4-23-2010

A Data Acquisition System For The NASA Specialized Center Of Research And Training Cuvette

Benjamin A. Riggs
Purdue University - Main Campus, bariggs@purdue.edu

Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Benjamin A. Riggs

Entitled A Data Acquisition System for the NASA Specialized Center of Research and Training Cuvette

For the degree of Master of Science

Is approved by the final examining committee:

Jeffrey Honchell
Chair

Cary Mitchell

Neal Widmer

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Jeffrey Honchell

Approved by: Gary Bertoline 04/19/2010
Head of the Graduate Program Date

Graduate School Form 20
(Revised 1/10)

PURDUE UNIVERSITY
GRADUATE SCHOOL

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:

A Data Acquisition System for the NASA Specialized Center of Research and Training Cuvette

For the degree of Master of Science

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Teaching, Research, and Outreach Policy on Research Misconduct (VIII.3.1)*, October 1, 2008.*

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

Benjamin A. Riggs

Printed Name and Signature of Candidate

4/13/2010

Date (month/day/year)

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/viii_3_1.html

A DATA ACQUISITION SYSTEM FOR THE NASA SPECIALIZED CENTER OF RESEARCH AND TRAINING CUVETTE

A Thesis

Submitted to the Faculty

of

Purdue University

by

Benjamin A. Riggs

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

May 2010

Purdue University

West Lafayette, Indiana

ACKNOWLEDGMENTS

I would like to begin by thanking my beautiful wife Armonda. Her love, support, and amazing patience helped guide me through this endeavor. Thank you for indulging me all of the times I solved a complex problem and had only a blinking light to show for it.

I would also like to thank my committee members: Professor Jeffrey Honchell, Dr. Cary Mitchell, and Professor Neal Widmer. Professor Honchell encouraged me to push myself further than I ever thought I could go and taught me how to find creative solutions to unconventional problems. Dr. Mitchell introduced me to the world of horticulture, thoroughly reviewed this thesis, and helped me reach outside of my field of study by “taking me outside my comfort zone.” Professor Widmer introduced me to the digital world and supported me throughout this thesis.

In addition, I would like to thank Dr. Gioia Masa and Keith Spence. Gioia provided much encouragement and was a great horticulture translator. Keith and I worked many long hours together on the cuvette system, and he often provided much needed support and ideas.

Thank you also to NASA, Jerry Shepard, Elaine Chase, Dr. Changhoo Chun, and the makers of Apache HTTP Server, Avacam, and ZedGraph.

Finally, I’d like to thank Purdue University and some of the faculty and staff in the Electrical and Computer Engineering Technology department for the past 7 years: Prof. Herrick, Nancy Tucker, Sandy Schnebly, Prof. Oxtoby, Prof. Blackwell, Prof. Robertson, Prof. Richardson, and Prof. Moss.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

LIST OF NOMENCLATURE

ADC – analog-to-digital converter

Advanced Life Support (ALS) – regenerative life-support systems (AMES Research Center)

Cuvette - “controlled environment chambers” used to determine “the net rate of carbon dioxide uptake by plants under different environmental regimes” (Bowman, 1968)

DAC – digital-to-analog converter

Data Acquisition system (DAQ) - “products and/or processes used to collect information to document or analyze some phenomenon” (Omega.com)

Datalogger – “A device that can read various types of electrical signals and store the data in internal memory for later download to a computer” (Omega.com)

DLL – dynamically linked library

Equivalent System Mass (ESM) – a transportation cost measure in ALS trade studies (Levri et al., 2003)

GTO – Geosynchronous Transfer Orbit (Futron Corporation, 2002)

GUI – graphical user interface

Hydroponics – "… a method of growing plants with their roots in a solution containing the mineral nutrients essential for plant growth." (Hoagland & Arnon, 1950)

Infrared Gas Analyzer (IRGA) – "an instrument that measures air samples for CO2 content" (Oak Ridge National Laboratory)

LED – Light Emitting Diode (Bourget, 2008)

LEO – Low Earth Orbit (Futron Corporation, 2002)

LGPL – Lesser General Public License

NiMH – Nickel Metal Hydride (Energizer Battery, 2001)

NSCORT – NASA Specialized Center of Research and Training

Photosynthesis – "the synthesis by organisms of organic chemical compounds, esp. carbohydrates, from carbon dioxide using energy obtained from light rather than the oxidation of chemical compounds." (Smith, 1997)

ppm – parts per million

RS232 – a standard for serial, binary data communication (Electronics Industries Association)

SCADA – Supervisory Control and Data Acquisition

Server – “a computer program that provides services to other computer programs (and their users) in the same or other computers” (WhatIs.com)

slm – Standard Liters per Minute

USB – Universal Serial Bus

Zigbee – “ZigBee is a wireless technology developed as an open global standard to address the unique needs of low-cost, low-power, wireless sensor networks.” (Digi.com)

ABSTRACT

Riggs, Benjamin A. M.S., Purdue University, May, 2010. A Data Acquisition System for the NASA Specialized Center of Research and Training Cuvette. Major Professor: Jeffrey Honchell.

This study explored the use of newer techniques to create a custom data acquisition system for horticultural purposes. Common horticultural lab equipment was used to measure environmental variables within a plant growth chamber, known as a cuvette. The cuvette was used to test various combinations of growing conditions and plant crops to gather data on the most feasible scheme for astronauts to remain in space in perpetuity. The lab equipment was networked to a common personal computer running custom software. The software developed is a multithreaded program making use of the Microsoft .Net framework. It periodically gathers and charts data from the horticulture lab equipment, saves the data to a Microsoft Excel spreadsheet, and disseminates the data via the Internet.

CHAPTER 1. INTRODUCTION

This section of the document discusses the scope and significance of the data acquisition system used for the cuvette (pronounced q-vette). It also investigates how this system was designed and implemented. Finally, any assumptions, delimitations, and limitations are listed.

In plant sciences, a cuvette is an enclosure for measuring real-time gas exchange of plants. The environmental variables inside the cuvette can be altered by the researchers to observe how different plants react to different conditions. The extent to which the environment can be controlled within the cuvette will determine the accuracy, precision, and amount of gas exchange (C. Mitchell, personal communication, December 8, 2009). In order to record and control these variables, a data acquisition system is needed.

1.1. Scope

When engineers, scientists, and other professionals need to reliably gather data from remote locations or over long periods of time, they often make use of a data acquisition system. This thesis explores the development of a data acquisition system for a small crop-stand cuvette. The construction and mechanics of the cuvette involved will largely not be covered. What will be covered are the various measurements and controls used in the cuvette, the networking involved in gathering the data, the data acquisition system used to gather and store the data, and the methods used to convey that data to users.

This thesis will cover the instruments used to measure temperature, humidity, mass airflow, and CO_2 and H_2O concentration in the system. Parameters to be controlled by the data acquisition system include adjustment of

the CO_2 concentration in the cuvette chamber using the mass air-flow controllers and control to adjust the angle of a camera within the cuvette. The camera will provide pictures of the plant canopy, including the hue of the leaves.

It will be necessary to coordinate the data gathering to a central location. This thesis will discuss a method to network the various measurement instruments and controls. It will also describe how the data will be collected, stored, and organized. Finally, the data will be accessible to users locally or remotely using the Internet.

1.2. Statement of Problem

The problem being addressed in this thesis is a need for a data acquisition system for a small crop-stand cuvette. Data needs to be gathered about the environment within the cuvette that the plants are growing in, how quickly they grow under different conditions, and their general health.

1.3. Significance of the Problem

The cuvette project is being implemented to explore the growth of plants in space by astronauts. These plants would provide the astronauts' nourishment. In addition, plants in space would act as natural CO_2 scrubbers, removing the CO_2 and replacing it with oxygen. Another benefit of plants in space would be to break down and dispose of all of the different kinds of waste produced by the astronauts.

Cuvettes are typically small in size and, when used for horticulture purposes, are used to enclose one leaf. The cuvette discussed in this thesis, however, houses a small stand of multiple plants. By enclosing the entire stand, the net photosynthetic activity of the crop can be measured, as opposed to just the photosynthetic rate of one leaf. This is accomplished by measuring the CO_2 consumption of the stand.

In similar situations, researchers would use commercial dataloggers to gather and control data. These devices can be left outside of buildings and store the data in memory until the researcher physically retrieves it. They are also quite expensive. The data acquisition system being employed for this cuvette runs on a standard personal computer and could potentially be migrated to a microcontroller integrated circuit, drastically reducing cost. In addition, the initial prototype would store the data in an easy to retrieve spreadsheet format. This data could be viewed locally via a graphical user interface or remotely via the Internet.

1.4. Statement of the Purpose

The purpose of this thesis is to design, assemble, and program a data acquisition system that can be used for a cuvette. Previously purchased horticulture equipment will be networked to a desktop computer. Some of this equipment includes a differential CO_2 analyzer, an absolute CO_2 analyzer, mass air-flow controllers, a camera, a two axis stage to move said camera, and various temperature and humidity probes. The data is collected using a custom program written in Visual C#. The program collects data from the networked equipment, organizes and formats the desired data from the raw data, and saves it into a spreadsheet format. Finally, the data is available via the Internet.

1.5. Assumptions

Assumptions for this project include:

- The software written for this project will be run only on modern Windows operating systems.
- The computer will have Microsoft Office 2003 or later installed.
- The computer will have Internet access.

1.6. Delimitations

Delimitations include:

- Only a brief summary of the mechanical system will be included.
- The software will be limited to Windows Operating Systems.
- The electrical system will not be controllable from the Internet, but data will be viewable.
- The camera used will use a separate freeware program.
- The electrical system will not operate wirelessly.

1.7. Limitations

Limitations for this project include:

- Some of the equipment was purchased prior to this researcher's involvement.
- The electronic system cannot be fully integrated and tested until the mechanical system is finished.

In this section, the scope and significance of the data acquisition system were discussed. The design and implementation were also investigated. Finally, any assumptions, delimitations, and limitations were listed.

CHAPTER 2. LITERATURE REVIEW

One of the greatest challenges facing scientists, engineers, and astronauts is creating and maintaining a habitable ecosystem in space. Considerations include carbon dioxide and oxygen exchange, disposal of waste products, and nutrition for the crew (Kliss, Heyenga, Hoehn,& Stodieck, 2000). All of these factors contribute to a balance between the cost of resupplying astronauts from Earth, having the astronauts resupply themselves by recycling products they have with them (bioregeneration), or various other combinations. To that end, Purdue University was given the task of leading research into methods of helping future astronauts maintain a viable, bioregenerative ecosystem in space (Venere, 2002), also known as a controlled ecological life-support system (Mitchell, 1994). To investigate these possible methods, a plant growth chamber, known as a cuvette (Bowman & Hand, 1968), was built at Purdue University. Various crops are placed inside, and it controls its internal environment while gathering internal environmental data utilizing a Data Acquisition system. This data allows scientists to characterize the net photosynthetic rates of the plants under different environmental conditions. This thesis will cover why this project as a whole is being undertaken, the purpose of cuvettes and how they are used, how data acquisition systems can be used to collect data and control systems, and concepts inherent to data acquisition systems.

2.1. Costs of space travel

The need for plants aboard spacecraft and bases comes from a need to reduce the cost of transporting goods to and from space. Monetary costs

associated with various spacecraft have been investigated by the Futron Corporation (2002). Tables 2-1, 2-2, and 2-3 demonstrate the operating costs involved with sending payloads into space, while Tables 2-4, 2-5, and 2-6 show the price per pound of Low Earth Orbits (LEO) and Geosynchronous Transfer Orbits (GTO).

Table 2-1
Launch cost for small launch vehicles (Futron, 2002)

Name	Athena 2	Cosmos	Pegasus XL	Rockot	START	Taurus
Country	USA	Russia	USA	Russia	Russia	USA
Launch Cost($mil)	24	13	13.5	13.5	7.5	19

Table 2-2
Launch cost for medium launch vehicles (Futron, 2002)

Name	Ariane 44L	Atlas 2AS	Delta 2	Dnepr	Long March 2E	Soyuz
Country	Europe	USA	USA	Russia	China	Russia
Launch Cost($mil)	112.5	97.5	55	15	50	37.5

Table 2-3
Launch cost for large launch vehicles (Futron, 2002)

Name	Ariane 5G	Long March 3B	Proton	Space Shuttle	Zenit 2	Zenit 3SL
Country	Europe	China	Russia	USA	Ukraine	Multinational
Launch Cost($mil)	165	60	85	300	42.5	85

Table 2-4
Price per pound for small launch vehicles (Futron, 2002)

Name	Athena 2	Cosmos	Pegasus XL	Rockot	START	Taurus
Country	USA	Russia	USA	Russia	Russia	USA
LEO price/lb.($)	5,310	3,939	13,832	3,313	5,388	6,258
GTO price/lb.($)	18,448	N/A	N/A	N/A	N/A	19,234

Table 2-5
Price per pound for medium launch vehicles (Futron, 2002)

Name	Ariane 44L	Atlas 2AS	Delta 2	Dnepr	Long March 2E	Soyuz
Country	Europe	USA	USA	Russia	China	Russia
LEO price/lb.($)	5,007	5,136	4,854	1,548	2,467	2,432
GTO price/lb. ($)	10,651	11,890	13,857	N/A	6,729	12,598

Table 2-6
Price per pound for large launch vehicles (Futron, 2002)

Name	Ariane 5G	Long March 3B	Proton	Space Shuttle	Zenit 2	Zenit 3SL
Country	Europe	China	Russia	USA	Ukraine	Multinational
LEO price/lb. ($)	4,162	2,003	1,953	4,729	1,404	2,431
GTO price/lb. ($)	11,004	5,233	8,326	23,060	N/A	7,343

For technical and political reasons, the cost of transporting goods to and from space is often measured using an equivalent system mass (ESM) calculation. ESM is a way of converting the various factors that increase the payload of spacecraft into measures of mass and simple example is shown in Equation 1 (Levri, Fisher, Jones, Drysdale, Ewert, Hanford, et al., 2003). By

using plants to nourish the astronauts and recycle waste, space missions will be able to decrease the ESM of each mission.

$$ESM = M + (V * V_{eq}) + (P * P_{eq}) + (C * C_{eq}) + (CT * D * CT_{eq}) \quad (1)$$

where

ESM = equivalent system mass value of the system of interest [kg],

M = total mass of the system [kg],

V = total pressurized volume of system [m^3],

V_{eq} = mass equivalency factor for the pressurized volume infrastructure [kg/m^3],

P = total power requirement of the system [kW],

P_{eq} = mass equivalency factor for the power generation infrastructure [kg/kW],

C = total cooling requirement of the system [kw],

C_{eq} = mass equivalency factor for the cooling infrastructure [kg/kW],

CT = total crewtime requirement of the system [CM-h/y],

D = duration of the mission segment of interest [y],

CT_{eq} = mass equivalency factor for the crewtime support [kg/CM-h].

The benefits for creating a bioregenerative ecosystem in space lie largely in the realm of long-term missions. Plants would largely consume harmful substances from the ecosystem and provide essential substances for the astronauts. These benefits would be offset by initial equivalent mass costs. Pressurized growth chambers and temperature control would be essential to house the plants and produce the maximal yield. Current studies have examined the feasibility of different types of life-support systems: direct resupply from Earth, physical/chemical systems, or varying degrees of bioregenerative systems. Findings are mixed with some predicting bioregenerative system breakeven points at 29 years (Jones, 2006), and others indicating breakeven

points at approximately three years (Drysdale, 2001; Ferl, Wheeler, Levine, & Paul, 2002). As space missions increase in length, it will be inevitable that some or all of the crew's resupply needs will be facilitated by plants.

In addition to mission length versus the type of life support system, crop selection will be crucial. Drysdale (2001) has shown that crop selection can determine the "goodness" of various crops over various mission lengths. This "goodness" number was defined "by dividing the supply ESM by the local-production ESM." The supply ESM is the value of supplying astronauts from Earth, while the local-production ESM is the value of astronauts supplying themselves on Mars. As the system mass for resupply from Earth increases, a factor directly related to mission duration, the above goodness number will at some point exceed the number one, meaning it is more efficient to grow locally than with resupply from Earth. In approximately half the crops listed (obtained from a draft of the KSC Crop Handbook or the University of Florida EDIS database), "goodness" exceeded one after missions lasting 600 days.

2.2. Cuvettes

One of the main issues with growing plants in space is providing consistent lighting for the plants to survive and flourish. While natural sunlight may be sporadically available, electrical lighting will almost certainly be needed for photosynthesis to occur. Using current lighting technology, LED's will most likely provide that light as they are more efficient and rugged than other lighting technologies (Bourget, 2006). LED's can provide light at various wavelengths essential to plant growth, without contributing to a greater heat load to the canopy of plants. Our cuvette utilizes LED banks (called "light-sicles") containing red and blue LED's suspended amongst the plants. This provides the maximal amount of red and blue light (approximately 640nm and 440nm respectively) needed for photosynthesis without exposing the environment of the cuvette to the additional heat load.

In order to test different lighting schemes, CO_2 concentrations, and other environmental variables on the types of plants destined for space, scientists make use of cuvettes here on Earth, using either soil or hydroponics as a growing medium. By varying variables such as light, temperature and CO_2 concentration, scientists can adjust the rate of photosynthesis. Methods used to measure photosynthesis have come a long way in the last few decades. Long, Farage, and Garcia (1996) describe how previously the only way to determine photosynthesis rates was with mobile laboratories. They go on to say:

> Measurement of photosynthesis then required an intricate knowledge of the infrared gas analyser, its daily or hourly calibration, flowmeters, properties of the materials used, and vigilant leak detection. Similarly, calculation of CO2 uptake from measured CO2 mole fractions, flow rate, pressure, temperature, humidity, leaf area, etc. would require an intricate knowledge of the equations and corrections, and probably access to a mainframe computer. (p. 1629)

In more recent times, much of this complexity has been simplified. Scientists are able to purchase off-the-shelf infrared CO_2 gas analyzers, LED arrays, mass airflow valves, etc. Initially, this led to largely mechanical cuvettes using electronic measuring devices (C.P. Akers, S.W. Akers, & Mitchell, 1985). As computers began to become more common in the 1980's, they began to assist scientists in controlling cuvettes (Knight, C.P. Akers, S.W. Akers, & Mitchell, 1988). Examples of both can be seen in Figures 2-1 and 2-2.

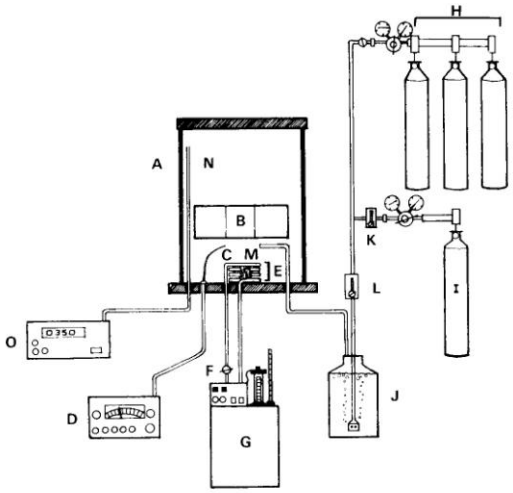

Fig. 1. Minitron chamber and support systems. **A.** chamber, **B.** plant container. Temperature monitoring: **C.** thermistor probe, **D.** telethermometer (YSI model 47, Yellow Springs Instrument Co.). Temperature control: **E.** heat exchange coils, **F.** needle valve, **G.** thermostatted, circulating water bath (Lauda model K-4R, Brinkmann Instruments, Inc.). Controlled atmosphere: **H.** compressed air (79% N, 21% O_2) cylinders on manifold, **I.** CO_2 cylinder, **J.** humidifier, **K.** rotameter (Matheson model 601), **L.** rotameter(Matheson model 603), **M.** atmosphere inlet. Atmosphere (CO_2) monitoring: **N.** atmosphere outlet, **O.** infrared gas analyzer (Infrared Industries model 705).

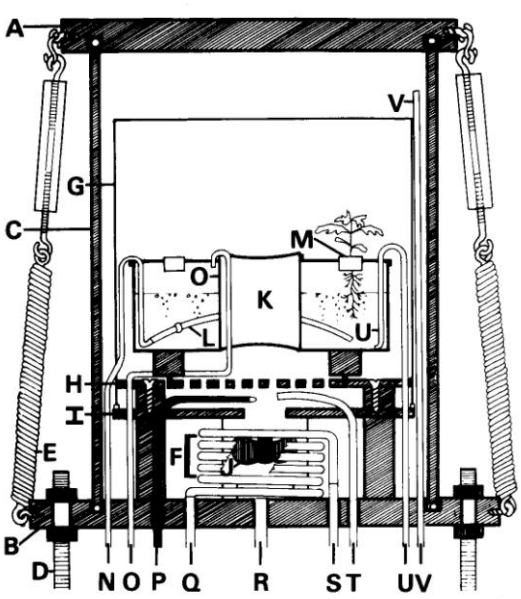

Fig. 2. Schematic side view of Minitron chamber. External parts: **A.** lid, **B.** base, **C.** cylinder. Both lid and base contain machined circular grooves (containing o-rings) into which the cylinder fits. Threaded rods (**D**) through each corner of the base support and level the chamber. Four tension-adjustable springs (**E**) connect the lid to the base and secure the cylinder in place. The cylinder is 30.59 cm OD, 40.78 cm high, and has an internal volume of 26.9 liters, excluding equipment and plant material. Omitted from diagram — lateral radiation shield surrounding bottom half of the outside of the cylinder. Internal parts: The copper heat exchange coils (**F**) are painted flat black to maximize heat exchange and have a wire screen (not illustrated) soldered to the outer edges to increase surface area. **G.** baffle, **H.** perforated platform, **I.** plate, **J.** fan (Rotron Sprite model SU 2-A1, Pioneer Inc.), **K.** plant growth (hydroponics) container, **L.** air wand, **M.** closed-cell foam plant holder. Portals in base: **N.** inlet for nutrient aeration, **O.** nutrient solution inlet or nutrient compartment headspace air outlet, **P.** thermistor probe, **Q.** coolant inlet, **R.** drain for condensate from heat exchange coils, **S.** coolant outlet, **T.** atmosphere inlet, **U.** nutrient solution outlet, and **V.** atmosphere outlet.

Figure 2-1 Internal and external workings of the Minitron[1]

[1] From “The Minitron System for Growth of Small Plants under Controlled Environment Conditions,” by C.P. Akers, S.W. Akers, and C.A.Mitchell, 1985, *Journal of the American Society for Horticultural Science*, *110*(3), p. 354. Reprinted with permission.

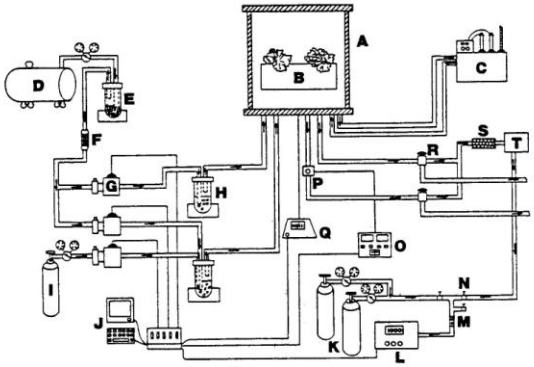

Fig. 1. Schematic of *Minitron II* environmental control and monitoring systems. *A*, chamber; *B*, root compartment; *C*, thermostatted, circulating water bath (*Lauda* model *K-4R*); *D*, air compressor (*Gast* model *5HCD*); *E*, air-drying cylinder containing $CaCl_2$ pellets; *F*, air-purifying tower containing *Purafil*; *G*, massflow controller (*MKS* model *1259A*); *H*, humidification cylinder; *I*, Carbon dioxide cylinder (99.95 % carbon dioxide); *J*, micro-computer (*Action Instr.* model *BC-3*); *K*, calibration gas standards; *L*, infrared gas analyzer (*Horiba* model *PIR-2000*); *M*, rotameter (*Matheson* model *603*); *N*, needle valve; *O*, dew-point hygrometer (*EG&G* model *911*); *P*, dew-point remote sensor; *Q*, scanning telethermometer (*YSI* model *47*); *R*, three-way valve; *S*, air-drying cylinder containing $Mg(ClO_4)_2$; *T*, booster pump (*Metal Bellows* model *MB-21*).

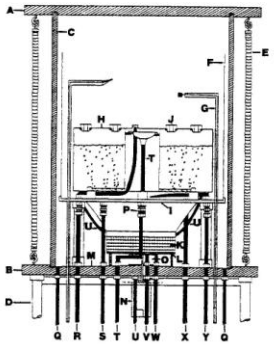

Fig. 2. Schematic side view of a *Minitron II* chamber. The cylinder is 63.5 cm high, 61.2 cm OD, and has an internal volume of 0.151 m^3, excluding internal accessories. *A*, lid; *B*, base; *C*, cylinder; *D*, height-adjustable, threaded metal legs; *E*, tension springs; *F*, air-flow baffle; *G*, height/position-adjustable thermistor; *H*, hydroponics container; *J*, hydroponics container platform; *J*, plant holder with closed-cell plug in middle; *K*, automobile core heat exchanger; *L*, heat-exchanger platform; *M*, outward-sloping floor; *N*, fan motor; *O*, fan blades; *P*, bulkhead fittings for portals in base; *Q*, condensate drain from chamber; *R*, rhizosphere outlet; *S*, nutrient solution inlet; *T*, shoot atmosphere outlet; *U*, inlet for nutrient solution aeration; *V*, coolant inlet; *W*, coolant outlet; *X*, nutrient solution outlet; *Y*, shoot atmosphere inlet.

Figure 2-2 Internal and external workings of the Minitron II[2]

Both Minitrons make use of an open system, where atmosphere is blown into the cuvette, after being thoroughly mixed, and then exits the cuvette. A similar system is a semi-open system. The semi-open system uses atmosphere and separate injection of CO_2, which maintains an enriched concentration. Both systems use differential gas analyzers to compare the reference air entering the cuvette with samples from the outlet of the cuvette to determine the net photosynthetic rate. Other systems include closed and semi-closed systems. These systems suffer from problems with H_2O absorption by the gas analyzers, making for inaccurate readings (Mitchell, 1992). The new cuvette developed for Purdue University is a semi-open system. Known amounts of CO_2 are mixed with ambient atmosphere, injected into the cuvette, and measured using gas analyzers on the outlet. Like the Minitrons, plants will be grown hydroponically.

---

[2] From “Minitron II System for Precise Control of the Plant Growth Environment,” by S.L. Knight, C.P. Akers, S.W. Akers, and C.A.Mitchell, 1988, *Photosynthetica 22*(1), p. 90-98. Reprinted with permission.

2.3. Data acquisition systems

Data acquisition systems (DAQ) gather data from various sources, format it into a desired manner, and present it for analysis. This section will discuss how data acquisition systems have been used in the realm of agriculture, have been used in other applications, and finally, some key concepts related to data acquisition systems.

2.3.1. Data acquisition in agriculture

The Minitrons and other early cuvettes showed continuing progress from the early days of calculating photosynthesis described by Long et. al (1996). In more modern times, scientists have begun utilizing data acquisition systems previously developed for monitoring greenhouses or agricultural fields. DAQs remove the need for personnel to constantly monitor and control these types of environments. However, there isn't just one approach to constructing or using a DAQ.

One approach includes multiplexing analog signals from multiple chambers and storing the data on a datalogger for later download to a personal computer (van Iersel & Bugbee, 2000; Wünsche & Palmer, 1997). As this method was used in multi-chambered projects, if signal wires are too long it is possible the analog signals could be corrupted by electromagnetic interference. This is largely avoided in our design by using shielded, digital signals. When analog signals are used, they are also shielded and passed through a buffer to provide proper impedance characteristics.

Timlin et al. (2006) performed experiments examining carbon partitioning using potato crops. The crops were placed in a transparent outdoor enclosure allowing the sun to provide light needed for photosynthesis. The potatoes were grown in that chamber using soil. Resistive heaters and heat exchangers were used to maintain constant humidity and temperature. CO_2 injectors utilized proportional-integral-derivative algorithms to control the amount of CO_2 injected during the day. The chamber's control and data collection were performed using

a SPARC 5 workstation. Our cuvette consists of an opaque indoor structure. Lighting, including luminous intensity and wavelength, can be precisely controlled by the operator via LED banks. We also make use of heat exchangers to maintain constant humidity and temperature. Mass air flow valves contain firmware that utilizes proportional-integral-derivative control, with setpoints entered by the user. As control and data collection are not computationally expensive in our project, an inexpensive personal computer will be used in lieu of an expensive workstation.

Mendoza-Jasso et al. (2005) focused on how to implement a DAQ that could concurrently gather data from several sensors in a greenhouse. This was accomplished with the use of a Field Programmable Gate Array (FPGA). This device is unique from other programmable electronic devices in that once programmed, it doesn't perform its function using lines of instructions. Rather, the program interconnects transistors inside to function like a hardwired circuit. This allows the device to process data much more quickly than traditional electronics, which rely on instructions that must be sequentially processed at a predetermined clock frequency. A similar implementation consisted of an FPGA using a fuzzy logic algorithm (Castañeda-Miranda, Ventura-Ramos, del Rocío Peniche-Vera, & Herrera-Ruiz, 2006). However, in agriculture there is rarely a need for concurrent data collection. As plants typically grow or change very little over relatively short periods of time, sampling periods need only be on the order of minutes. If multiple sensors are used, collecting their data sequentially within seconds of each other should not produce erroneous results.

Helmer, Ehret, & Bittman (2005) made use of commercial dataloggers, with included software, and customized software. Tomato crops were grown in a greenhouse, and the vines were suspended from overhead crossbeams using load cells. Load cells were also placed under the crop and, in concert with the upper load cells, measured the mass of the growing media. Commercial software, included with the datalogger, was used to measure and collect data from the load cells and other support sensors. Also, the software allowed users

to create and edit the datalogger data collection and control routines. Large data files could also be partitioned into smaller, more manageable files. It also contained a graphical user interface for displaying the data real time on a personal computer. Helmer et al. (2005) then created their own custom software, written in Visual Basic 6. These performed routine tasks such as file renaming and file copying and displaying. Our data acquisition system excludes dataloggers, in part, due to their expense. Also, dataloggers do not provide as much flexibility, as part or all of their software is previously written. Our data acquisition system is largely composed of custom software to fully meet the needs of the scientists involved. Support for Visual Basic 6 ended entirely by March 2008. Visual C# was used in our system due to continued support and the availability of the Microsoft .NET framework.

DAQs can also be used in agricultural fields to gather data; however, it is impractical in these situations to use wires to power and gather data from sensors. In these cases, data-gathering sensors need to be powered using batteries or nearby renewable energy. Morais et al. (2008) utilized three nearby energy sources for their sensors. Sunlight was gathered with a solar panel, wind energy was captured with a wind turbine, and water-flow energy was gathered from irrigation pipes using a water turbine. This energy charged a nickel-metal hydride (NiMH) battery pack and provided weather data without using additional sensors. These data were then transmitted wirelessly using Zigbee modems to a data collecting receiver. Vellidis, Tucker, Perry, Kvien, and Bednarz (2008) take a slightly different approach to a DAQ. Renewable energies were not used, instead relying on high energy density lithium batteries that last one full growing season. They also made unconventional use of commercial Radio Frequency IDentification (RFID) devices. Normally, RFID devices simply return their identification number when queried by a wireless transceiver. Vellidis et al. (2008) encoded environmental data from sensors into this identification number. Both strategies show different ways of using DAQs for agricultural purposes.

2.3.2. Other applications for data acquisition

The emerging field of renewable energy has also benefited from data acquisition systems. As renewable energy sources are often in remote locations, it is often beneficial to connect data measuring sensors, attached to the energy sources, wirelessly to data acquisition systems.

Kalaitzakis, Koutroulis, and Vlachos (2003) designed such a system with two key elements. The first involved using the JAVA platform. JAVA is a system and programming language that allows for cross-platform implementation (i.e. the programs can run independent of the hardware it's run on, using a virtual machine, and independent of the operating system used). In their system, they implemented a JAVA collection program, which gathered the data from the data acquisition systems for storage on a server. Their second element included using the server to make the data available to remote viewers who used a JAVA applet to access the server from anywhere in the world via the Internet. While the aspect of a cross-platform system could be appealing, it was deemed unnecessary for our applications. We did, however, implement an Internet server to allow remote viewers read-only access to the data.

Another use of data acquisition systems for renewable energy included the use of LabVIEW (Koutroulis & Kalaitzakis, 2003). LabVIEW is a proprietary product from National Instruments that allows scientists and engineers to create various programs, such as data acquisition and industrial automation, using a visual programming language (i.e. non-textual programming). This allows those not trained in traditional programming to develop programs using representations of equipment or objects they are familiar with into block diagrams, which then implement the algorithm of their program. Signals for this data acquisition system come from various sensors which must be properly interfaced with the data acquisition card installed in a personal computer. The DAQ card is purchased from National Instruments. This interface is accomplished using filters, amplifiers, and various other signal conditioning circuits. This approach can be an excellent alternative for non-traditional programmers. However, LabVIEW programs cannot

run independently of the LABVIEW development environment without purchasing additional components. Even when standalone programs are developed, most operating systems don't have the LabVIEW runtime library installed by default. This would cause problems if a new computer was used or the program was transferred to another researcher. The DAQ developed at Purdue University was developed using Visual C# and the Microsoft .NET framework. This will prevent the software from being used on computers using non-Windows operating systems, but that comprises less than 10% of personal computers (Net Applications, 2009), meaning excellent compatibility.

The previous two sections discussed applications of data acquisition systems. The next section will describe concepts relating to how they work.

2.3.3. Data acquisition concepts

Data acquisition systems are used to gather data, often from analog sensors, and store or process that data. To do this, the analog information must be converted into digital data. This task is frequently carried out by an analog-to-digital converter (ADC).

Most data we gather begins as an analog signal such as the one shown in Figure 2-3. In order for data to be saved or manipulated by digital machines, these continuous time signals must be sampled into finite, binary values. This process is called quantization and is shown in Figure 2-4. The dashed line represents the digital values stored each time the analog signal is sampled. The time between these samples is called the sampling period and inversely the sampling frequency. It is critical that the sampling frequency is at or above the Nyquist rate, defined as two times the highest frequency in the signal being measured. This prevents aliasing, an example of which is shown in Figure 2-5 (Burr-Brown, 1994). The asterisks indicate samples from the original signal at an inadequate sampling period, and the dotted line represents the perceived waveform.

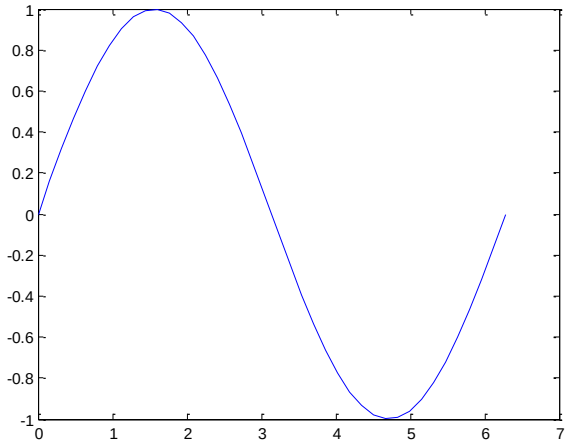

Figure 2-3 Analog sine wave

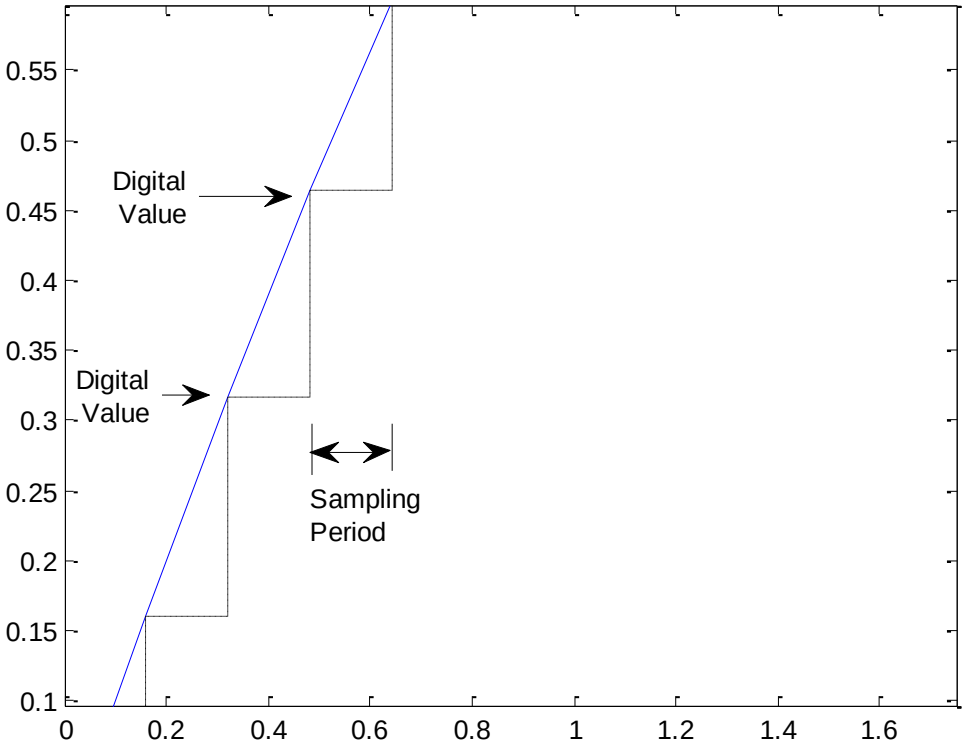

Figure 2-4 Quantization of sine wave

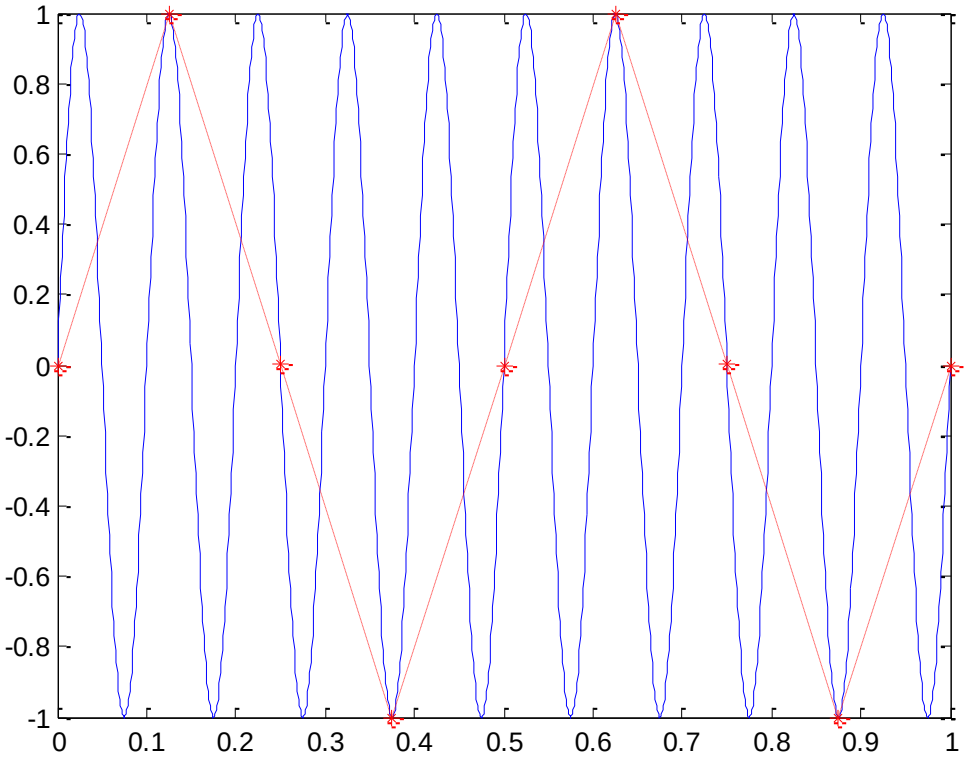

Figure 2-5 Aliasing (10 cycle signal and 8 sampling points per frame)

As mentioned previously, quantization converts continuous time signals into discrete time signals. However, the act of quantizing introduces errors. Any given ADC contains a finite set of binary values to approximate the continuous signal to. This can be seen in Figure 2-4 as the dotted line rarely equals the magnitude of the actual signal at any given point in time. The size of the set of binary values for an ADC is a function of its resolution, as seen in Equation 2 (Tocci, Widmer, & Moss, 2004).

$$RESOLUTION = V_{REFERENCE} / (2^n - 1) \quad (2)$$

where

$V_{REFERENCE}$ = voltage differential applied to the reference(s) of the ADC

n = number of binary bits used in the ADC

For example, in an eight bit ADC with 2.55VDC applied to the reference and 1V applied to the input, each binary bit in the binary number would represent 1mV, as shown in Equation 3, and the binary number stored would be 100, as shown in Equation 4.

$$2.55/(2^8 - 1) = 0.001V/bit \quad (3)$$

$$1V/0.001V/bit = 100_{10} = 01100100_2 \quad (4)$$

Typically, in order to quantize an analog signal, a sample is taken using a sample and hold circuit, so that the ADC can then calculate a proper binary value. This calculation is most commonly performed with a ramp ADC (also known as a counting ADC) or a successive approximation ADC. A ramp ADC uses a clock to increment a voltage from 0V by small steps until that voltage exceeds the input signal voltage, as seen in Figure 2-6. A successive approximation ADC uses each bit individually to narrow in on the input signal voltage. It begins by setting the most significant bit on a digital to analog convertor (DAC) high, corresponding to one-half of the reference voltage. This voltage is then compared to the input signal. If the DAC voltage is less than the signal voltage then the bit remains high, and vice versa. Then the next most

significant bit of the DAC is set high, corresponding to any previous voltage that remained plus one-quarter of the reference voltage, and the voltages are compared. This continues through to the least significant bit, as shown in Figure 2-7 (Tocci et al, 2004; Maxim, 2001).

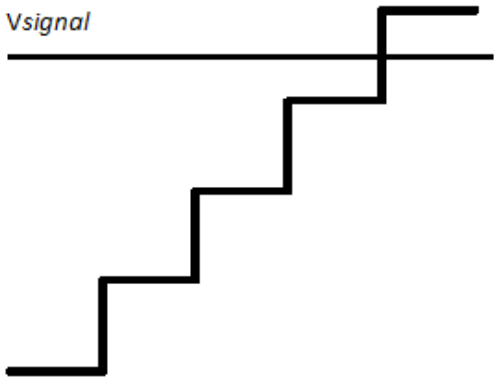

Figure 2-6 Counting ADC

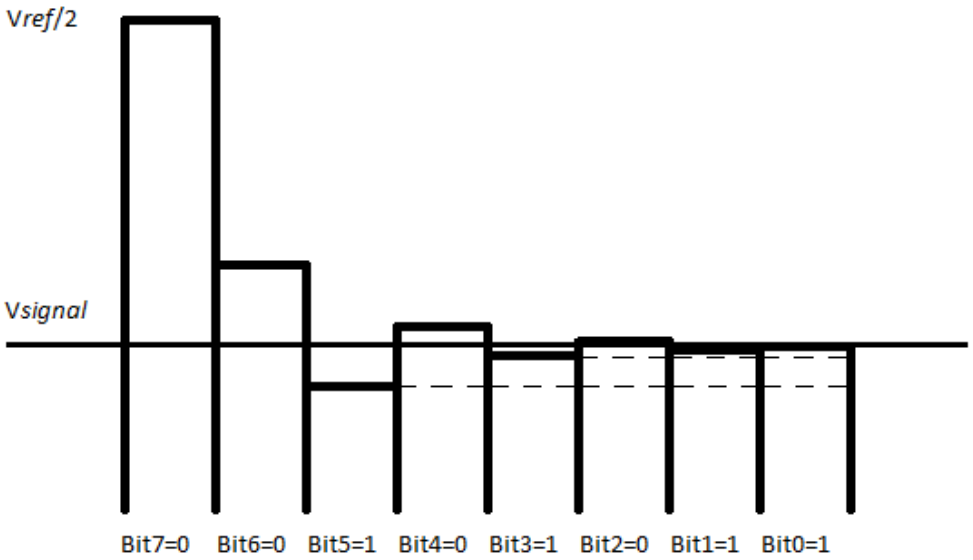

Figure 2-7 Successive Approximation ADC

This section described how analog information can be converted into digital data, in a process called quantization. In addition, techniques were discussed for how ADC's perform this conversion.

The previous three sections have shown how data acquisition systems are used to gather data from various sources, format it into a desired manner, and present it for analysis. They have also shown how data acquisition systems have been used in the realm of agriculture, have been used in other applications, and shown some concepts used in data acquisition systems.

In order for mankind to exist long periods of time in space, scientists and engineers will need to develop new methods of using plants to provide for our needs and remove our wastes. Plants will need to provide nutrition and oxygen to astronauts while removing carbon dioxide and various forms of waste. To develop these new methods, scientists will need to experiment on Earth using cuvettes. These cuvettes are able to precisely control plant photosynthesis rates by varying CO_2 levels, light levels, and other environmental variables. In order to measure these variables, a data acquisition system needs to be developed to properly gather data and control variables inside the cuvette. Different implementations of DAQs have been explored here with each suited to address the needs of the researchers. Various concepts associated with DAQs were also presented.

CHAPTER 3. METHODOLOGY

In plant sciences, a cuvette is an enclosure for measuring real-time gas exchange of plants. The extent to which the environment can be controlled within the cuvette will determine the accuracy, precision, and amount of gas exchange (C. Mitchell, personal communication, December 8, 2009).

Essentially, a cuvette is some type of container that surrounds a plant, part of a plant, or a crop of plants. The cuvette may be transparent or opaque, indoors or outside, manually controlled or automated. Its purpose is to measure the growth of a crop by determining its net photosynthetic rate under different environmental conditions. These conditions may include luminous intensity, temperature, humidity, soil composition, hydroponic mineral composition, or a myriad of other factors that affect photosynthesis. Photosynthesis involves capturing luminous energy and combining it with carbon dioxide and water to produce carbohydrates, as a form of energy for the plant, with oxygen being a byproduct. This is shown in Equation 5.

$$6CO_2 + 6H_20 + energy \rightarrow C_6H_{12}O_6 + O_2 \quad (5)$$

The most common way of determining a crop's net photosynthetic rate, or growth rate, is to determine the quantity of carbon dioxide consumed by the plants. If a known amount of carbon dioxide is pumped into the cuvette, the amount of carbon dioxide present at the outlet of the cuvette is an indicator of how much was consumed by the crop inside the cuvette. In order to exactly

control the input carbon dioxide and exactly measure the output carbon dioxide, the crop of plants must be isolated inside the cuvette to prevent contamination from atmospheric air. This means the cuvette must be as close to an air-tight container as possible, with a slight positive pressure added to prevent any possible inward atmospheric leakage. Pictures of the cuvette can be seen in Figures 3-1 and 3-2.

Figure 3-1 Cuvette front

Figure 3-2 Cuvette Side

As seen in the proceeding pictures, the cuvette is an opaque container and is also housed indoors. In order for photosynthesis to take place inside the cuvette, light must be supplied to the stand of plants inside. “Light-sicles” were designed by Ortbital Technologies Corporation and suspended within the cuvette to provide this light, as shown in Figures 3-3 and 3-4.

Figure 3-3 Unilluminated “light-sicles”

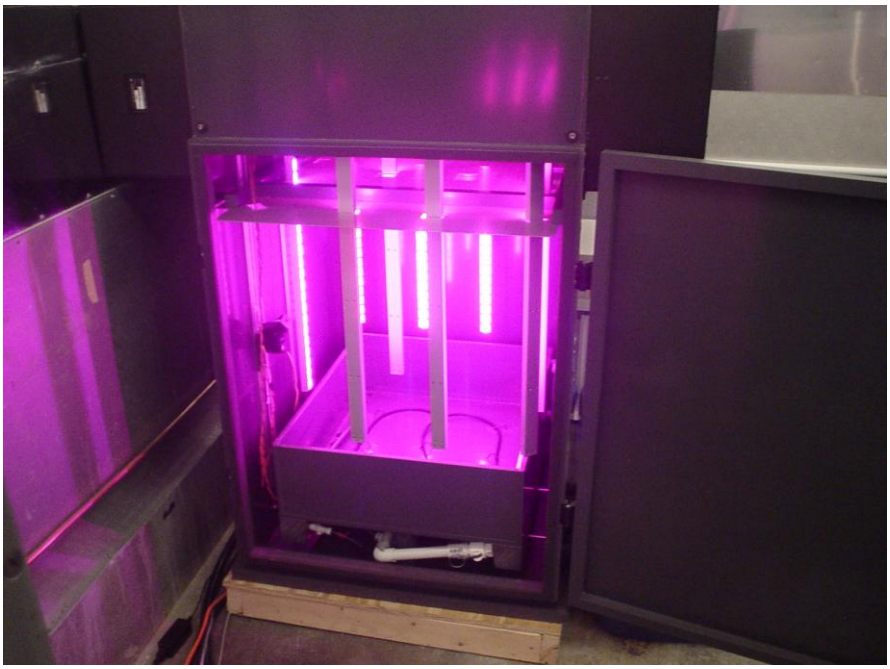

Figure 3-4 Illuminated “light-sicles”

3.1. Cuvette system

A simplified mechanical representation of the cuvette system is shown in Figure 3-5. The flow from a canister of pure CO_2 is controlled by a mass flow controller. The pure CO_2 then merges with ambient air collected from a blower. This enriched mixture is further controlled by another mass flow controller. This enriched mixture is split to pass through the cuvette and to bypass it. The bypassed mixture is then compared, using the differential gas analyzer, to the cuvette mixture to determine amount of CO_2 absorbed by plants inside. The absolute gas analyzer measures the bypassed mixture.

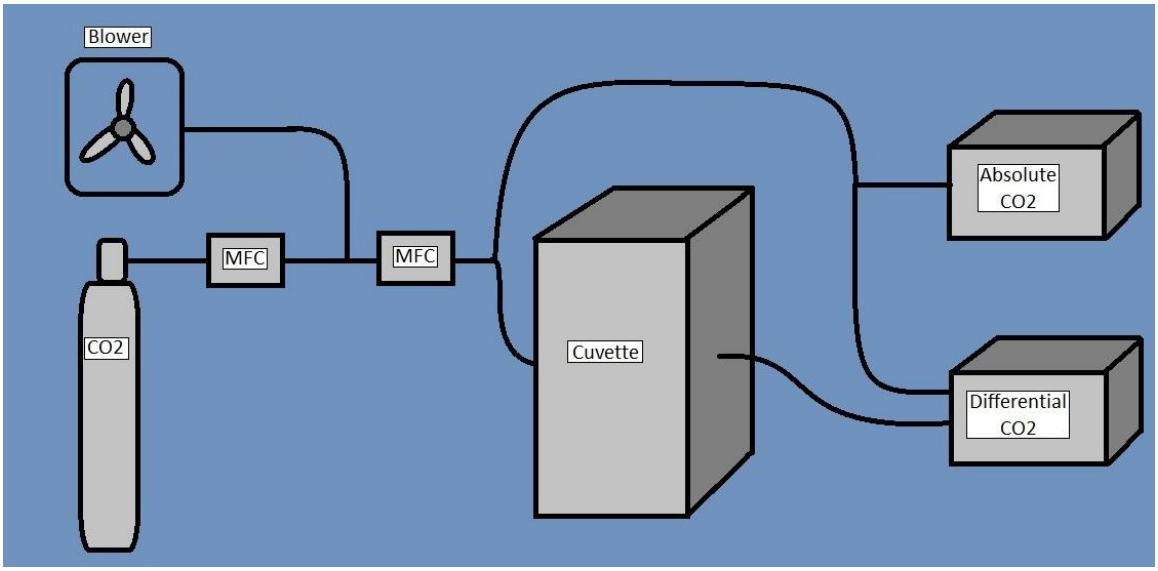

Figure 3-5 Cuvette system

3.2. Cuvette system components

The data acquisition and control system used for the cuvette is based on a Supervisory Control and Data Acquisition (SCADA) model. A wiring diagram of the system can be seen in Figure 3-6. A SCADA system includes several remote units that perform their respective tasks independently. When a supervisory unit requests data from a remote unit or sends control commands to the remote unit, the remote unit responds appropriately and then continues with its original task.

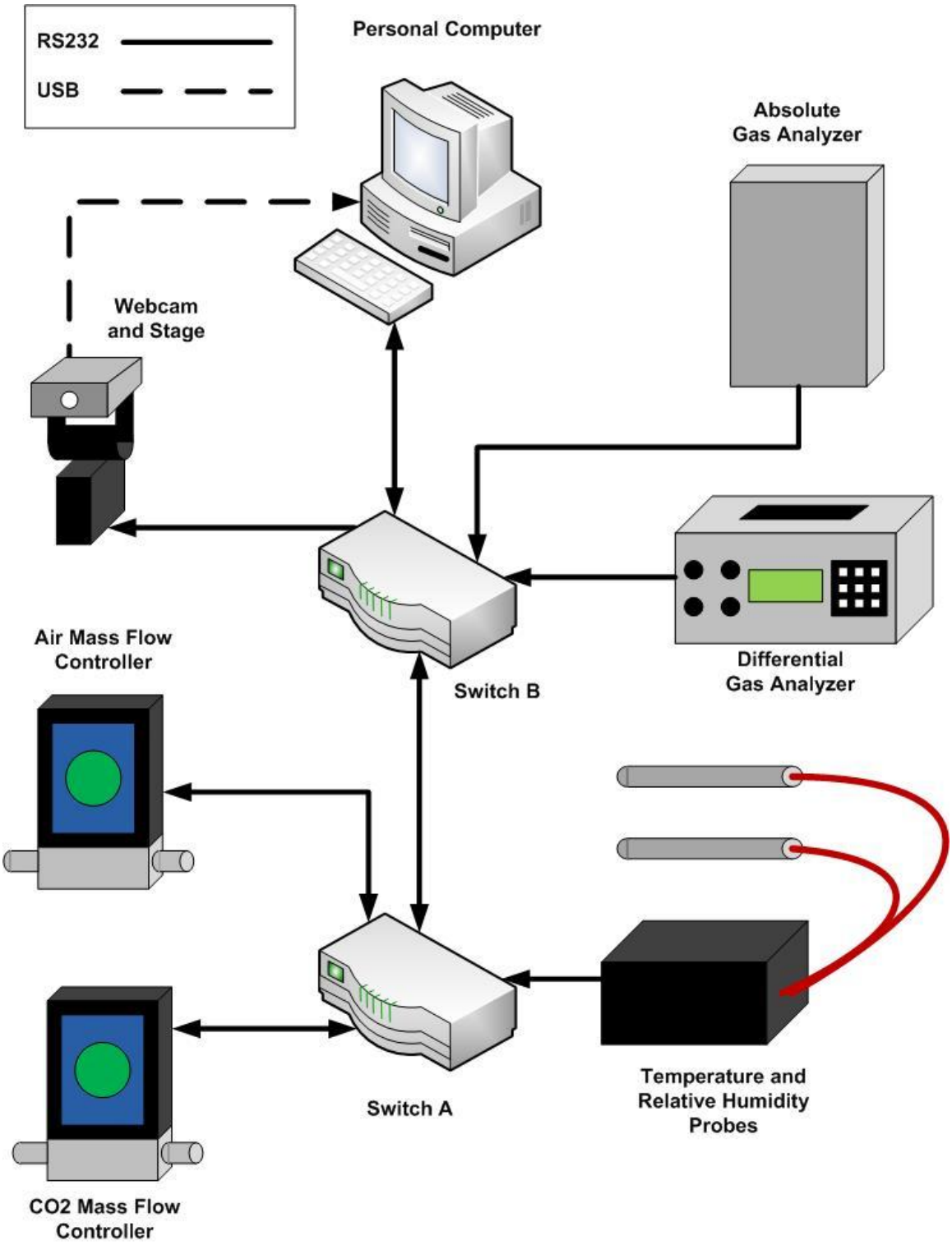

Figure 3-6 Cuvette network wiring diagram

3.2.1. Supervisory unit

A personal computer, running on Microsoft Windows XP, is used as the supervisory unit. This supervisory unit runs a custom written program that collects data and sends control signals, as necessary. In addition to the custom program, a freeware program was used with the webcamera to take periodic pictures inside the cuvette.

3.2.2. Serial Switches

Two serial switches, Figure 3-7, are used for communication between the supervisory unit and its remote units. These switches are five port, expandable smart switches (Model 232XS5) developed by B & B Electronics Manufacturing Company. Each switch contains a master port and five slave ports (Ports A-E). A host device (e.g. a personal computer or another switch) is connected to the master port. Up to 5 slave devices can be connected to the slave ports on a single switch. The switches can also be daisy-chained together by connecting the master port on one switch to Port C of a previous switch. Up to four switches can be daisy-chained to provide up to 17 serial ports (4 switches * 4 ports + 1 Port C). Various dipswitches and jumpers are physically set inside each switch to determine RS232 serial parameters (e.g. BAUD rate, number of data bits, and parity) and to determine the switch's expansion address.

In order to switch communications to a particular port, a series of characters, known as a preamble, must be sent to the switch to inform it that the following characters are a command for it to follow, as opposed to data to pass on to a slave device. Following the preamble is the expansion address character (A-E), if more than one switch is used. Finally, the character corresponding with the desired serial port is sent. The data format for the serial switches is shown in Table 3-1.

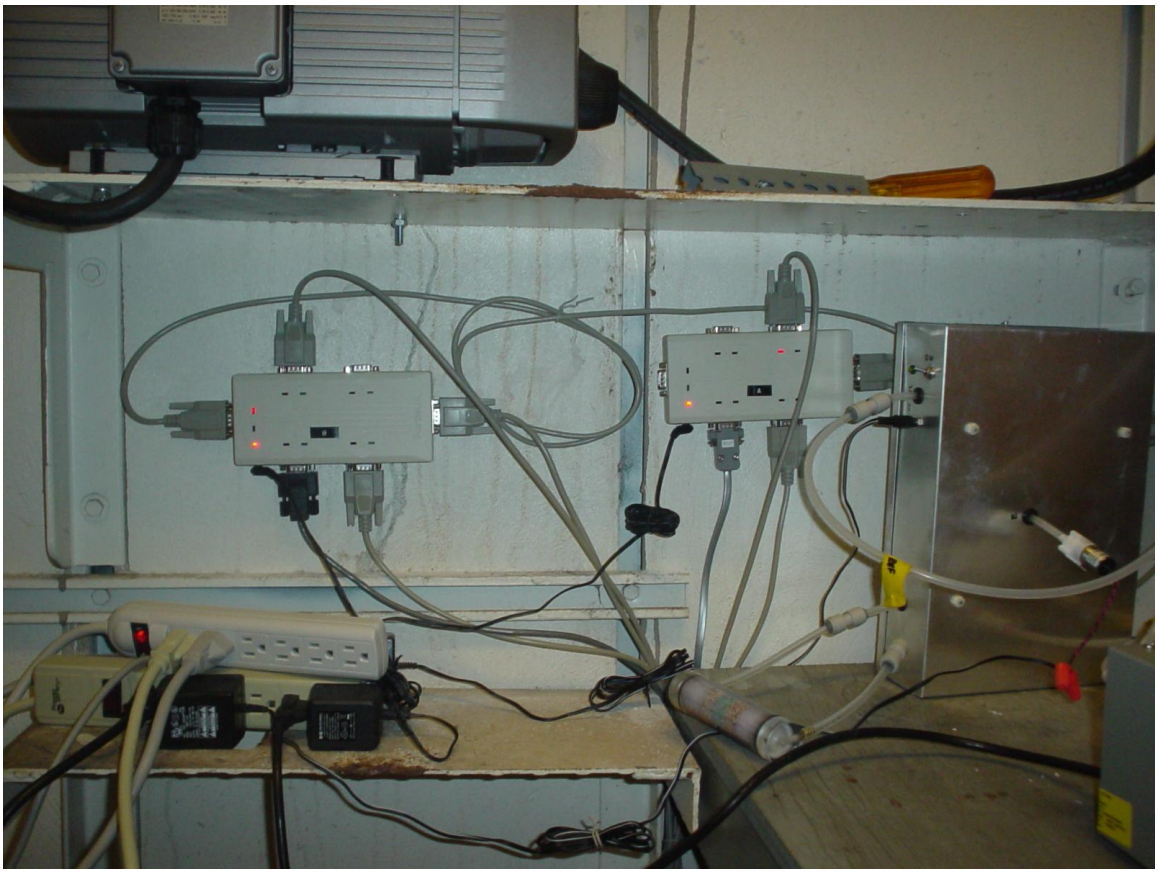

Figure 3-7 Serial switches

Table 3-1
Serial switch data format

Escape Preamble	User-defined Preamble	Expansion Address (optional)	Port
ESC	STX	<x>	<x>

where

ESC = ASCII escape character

STX = ASCII start of text character

<x> = ASCII capital letters A, B, C, D, or E

3.2.3. Webcam and stage

The webcam unit consists of a webcam, a two axis servo motor stage, and a servo controller. The webcam is a 3.0 megapixel USB webcam (Item: 460668) from Manhattan Computer Products. A freeware program is used to periodically or manually collect snapshots from the webcam. The two axis servo motor stage (Model Number: BPT-KT) and servo controller (Model Number: SSC-32), from Lynxmotion, are used to adjust the pan and tilt of the webcam, and can be seen in Figure 3-8. Serial commands (RS232) are sent, via the serial switches, to the servo controller to adjust the axes. These serial commands are converted to pulse-width modulated signals needed to command the servo motor stage to new positions. The positive pulse-width range for the servos is 500-2500 microseconds. The data format for the servo controller is shown in Table 3-2.

Figure 3-8 Two axis servo stage

Table 3-2
Servo controller data format

Servo Number	Pulse Width (µsec)	Carriage Return
#<ch>	<i>	<cr>

where

<ch> = channel number (0 or 1)

<i> = integer number between 500 and 2500

<cr> = ASCII carriage return

3.2.4. Absolute Gas Analyzer

The absolute infrared gas analyzer (IRGA) is an original equipment manufacturer, non-dispersive, CO_2 gas concentration analyzer (Model: SBA-4) developed by PP Systems. Since it was an original equipment manufacturer device (i.e. a circuit board) it was placed inside a metal enclosure with basic connections for power and communication, as shown in Figure 3-9. The absolute IRGA works on the premise that di-atomic molecules, such as CO_2, absorb photons in the infrared range. By placing a gas through a chamber with an infrared source and an infrared detector, measurements can be made about the photon absorption. The absolute IRGA contains an infrared source and an infrared detector sensitive to photons at 4.26 microns. The internal chamber is also heated to 55°C, providing constant temperature within the chamber. To help alleviate problems associated with changes in gas stagnation in the measuring chamber, atmospheric pressure, infrared detector sensitivity, etc., the absolute IRGA implements an Auto-Zero function. The Auto-Zero function activates a solenoid, as shown in Figure 3-10, which normally connects the CO_2 source gas to the measuring chamber, to temporarily connect a gas containing zero concentrations of CO_2. This allows any CO_2 in the chamber to be evacuated out of the outlet. When in measurement mode, the IRGA measures over a range of

0-5000 ppm. In the cuvette system, the absolute IRGA is used to measure the CO_2 concentration of the bypass stream. The data format for the absolute IRGA is shown in Table 3-3.

Figure 3-9 Absolute IRGA

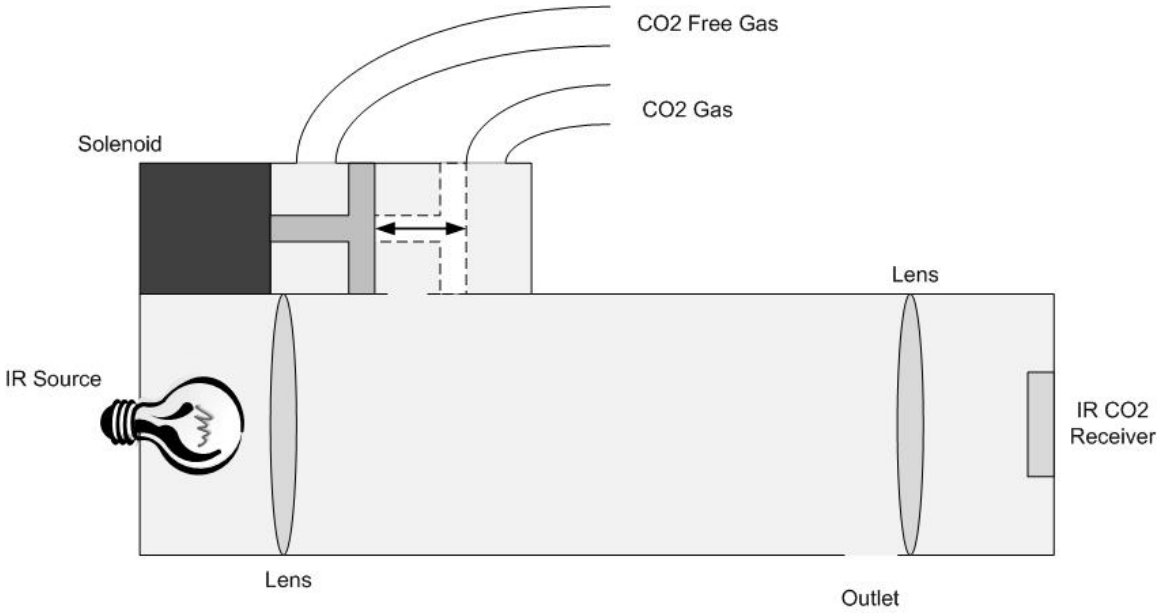

Figure 3-10 Absolute IRGA measurement chamber

Table 3-3
Absolute IRGA data format

M	ZERO	CO2M	CO2AV	ANLT	MB	HT	A	B	C	D	E	ATMP

where

M – represents the current mode (B-Begin, I-Initialization,…, M-Measurement)

ZERO – is a 16-bit value from the ADC the last time the IRGA Auto-Zeroed

CO2M – is a 16-bit value from the ADC at each sampling period

CO2AV – is a 16-bit running average of the CO2M readings

ANLT – is the temperature in Celsius of the measurement chamber

MB – not used in this experiment

HT – not used in this experiment

A through E – not used in this experiment

ATMP – barometric pressure (mbar)

3.2.5. Differential Gas Analyzer

The differential infrared gas analyzer (IRGA), Figure 3-11, is a differential, non-dispersive, CO_2 and H_2O gas concentration analyzer (Model: LI-6262) made by LI-COR. The differential IRGA works on a similar principle as the absolute IRGA in that di-atomic molecules, such as CO_2 and H_2O, absorb photons in the infrared range. However, in addition to measuring the absolute CO_2 concentration entering the cuvette, the differential IRGA also measures the CO_2 from the outlet of the cuvette and collects the difference of the two. It does this by placing an infrared source on one end of the chamber, as shown in Figure 3-12. Inside the chamber, gas samples pass through reference (cuvette inlet) and sample (cuvette outlet) cells. Since this is a CO_2 and H_2O IRGA, the radiation from the infrared source needs to be split in order to measure both parameters. At the other end of the chamber, the infrared radiation is passed through a dichroic beam splitter. This splitter allows half of the radiation to pass, while reflecting the remainder at a 45° angle. Half of the infrared light then passes through a 150nm bandpass optical filter so that the CO_2 detector can measure the 4.26 micron absorption band. The other half of the infrared light passes through a 50nm optical filter so that the H_2O detector can measure the 2.59 micron absorption band. The differential IRGA measures over a range of 0-3000 ppm.

Figure 3-11 Differential IRGA

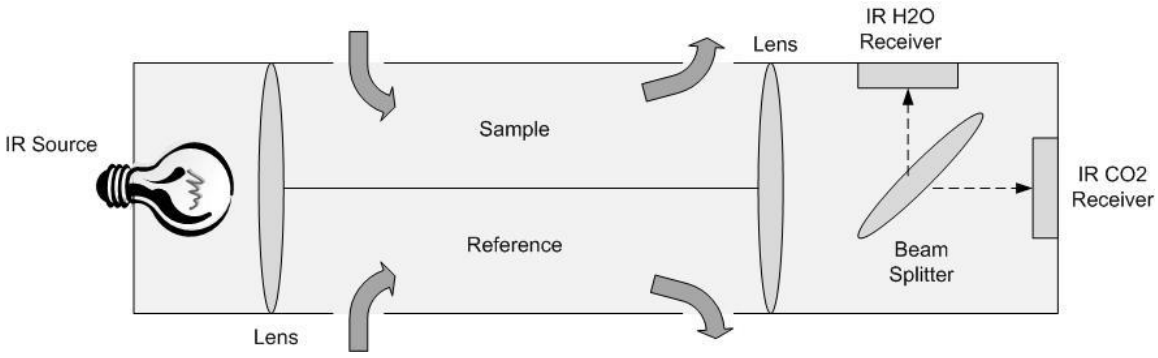

Figure 3-12 Differential IRGA measurement chamber

The data format for the differential IRGA is user-defined. It makes extensive use of “functions” organized into lists. These functions perform a variety of tasks from performing calibrations to setting the sampling period. They can also be used to set the units for the CO_2 and H_2O readings. In other words, the ADC values are converted into raw millivolts, µ(m)mol mol^{-1}, (k)Pa, µ(m)g g^{-1}, or several other desired units before being displayed or transmitted.

3.2.6. Temperature and relative humidity

The temperature and relative humidity (RH) unit consists of two identical temperature and RH probes and an analog-to-digital converter (ADC) module. The temperature and RH probes (Model: HMP50) are produced by Vaisala. The ADC module (Model: 232M1A0CT), Figure 3-13, is produced by Integrity Instruments. The temperature and RH probes consist of a platinum resistance thermometer (PRT) and a proprietary humidity probe. The PRT is classified as a pt1000, meaning it is 1000Ω at 0°C. It also has a positive temperature coefficient, meaning resistance rises with temperature. The temperature and RH probe measure -10°C to 60°C at ±0.6°C and measure 0-98% humidity at approximately ±4% humidity. The ADC module has eight channels containing 10-bit ADCs that convert analog signals to serial (RS232) communication. The temperature and RH probes are buffered to the ADC module using a voltage follower to insure proper impedance characteristics. The data format is shown in Table 3-4. The ADC module also contains eight digital-to-analog convertors and a pulse-width modulation channel for potential future use in the project.

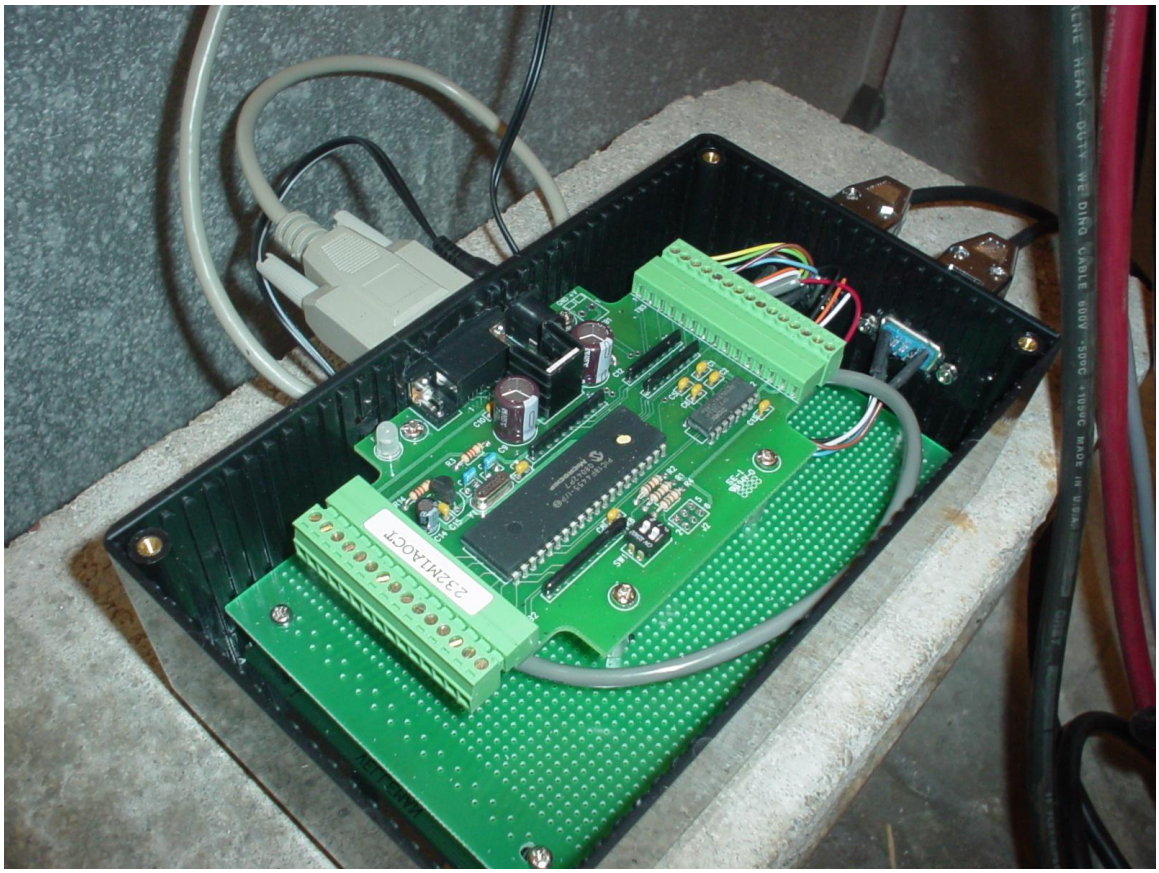

Figure 3-13 ADC module

Table 3-4
Analog-to-digital converter data format

Command Sent	Response Received
U<ch>	U<ch><value>

where

<ch> = channel number between 0 and 7

<value> = integer value between 0 and 1023

3.2.7. Mass Flow Controller

The mass flow controller (MFC) units are digital mass flow controllers (Model: HFC-D-302) developed by Teledyne Hastings. They are used to measure and control the amount of mass of a given gas that passes through it over a period of time. An MFC is shown in Figure 3-14. They differ from volumetric flow controllers in that MFC's control mass flow independently of changes in the flow pressure or temperature. However, volumetric units of flow rate are more familiar to most scientists and engineers, so the MFC's convert their mass units unto standard volumetric units. These volumetric units are considered standard as the internal temperature and pressure of the MFC are held constant.

Figure 3-14 Mass flow controller

The MFC's begin by taking a small sample from the inlet gas, as shown in Figure 3-15, while the remainder passes on to outlet. The sample is passed up a stainless steel tube called the shunt. The shunt passes through an aluminum block on both of its ends. The aluminum block, shown below the control circuitry, is in turn in direct contact with the steel chassis of the MFC. This ensures that the aluminum block, and in turn either end of the shunt, are constantly kept in equilibrium with ambient temperature. Coils are wrapped around the aluminum block in order to sense the ambient temperature, and therefore the temperature of the incoming gas. Symmetric coils of wire are wrapped around the center of the shunt and heated to 48°C above ambient temperature. These coils, and the coils wound around the aluminum block, are resistance temperature detectors. As temperature rises and falls, so too does the resistance of each coil. With no gas flowing through the shunt, both of its two heated coils of wire remain at 48°C above ambient, with more heat near the center, as shown in Figure 3-16. Once gas does begin to flow, the upstream coil begins to lose heat to the gas. This heat transfer is dependent on the type of gas. Some of the heat from gas is then returned to the downstream coil, causing it to heat up. This results in a change in the resistance both of the two coils, with one getting larger and the other getting smaller. These resistance temperature detectors are each connected in a Wheatstone bridge circuit along with one of the coils wrapped around the aluminum block. These circuits ensure that each shunt coil remains at 48°C above ambient temperature by adding or removing heat as necessary. The difference of these signals is then compared to a set point voltage defined by the user, using software. This comparison commands the control valve to open or close more, setting the flow rate at the outlet of the MFC (Sierra, 2008; Teledyne, 2005).

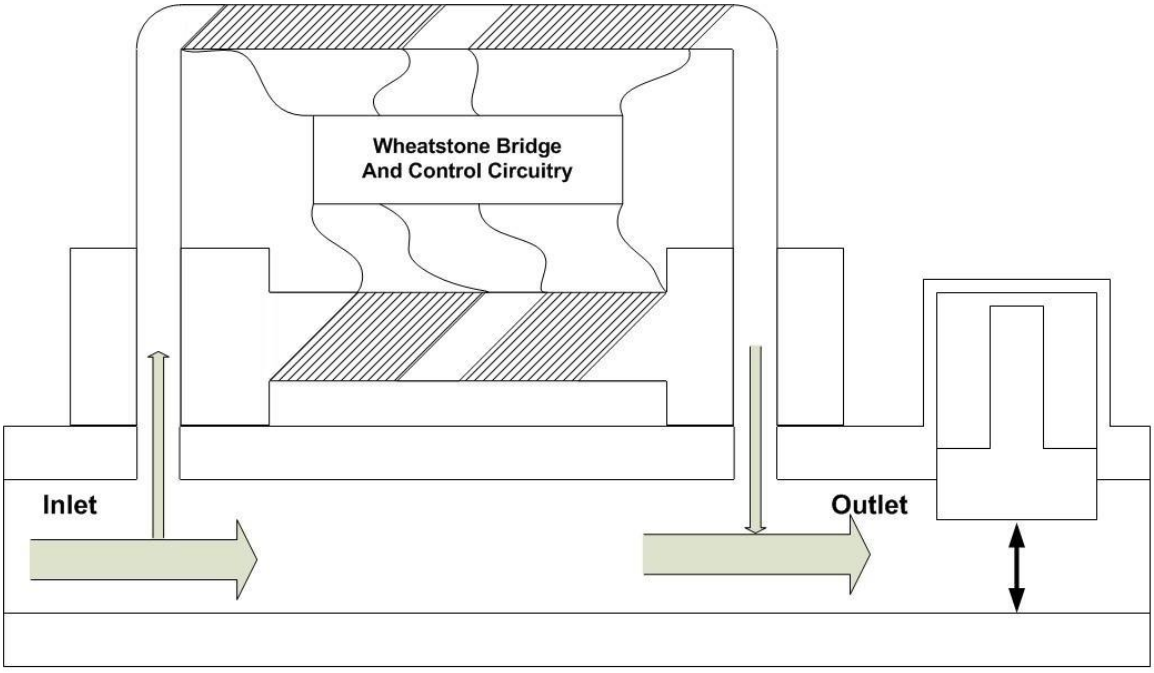

Figure 3-15 Mass flow controller operation

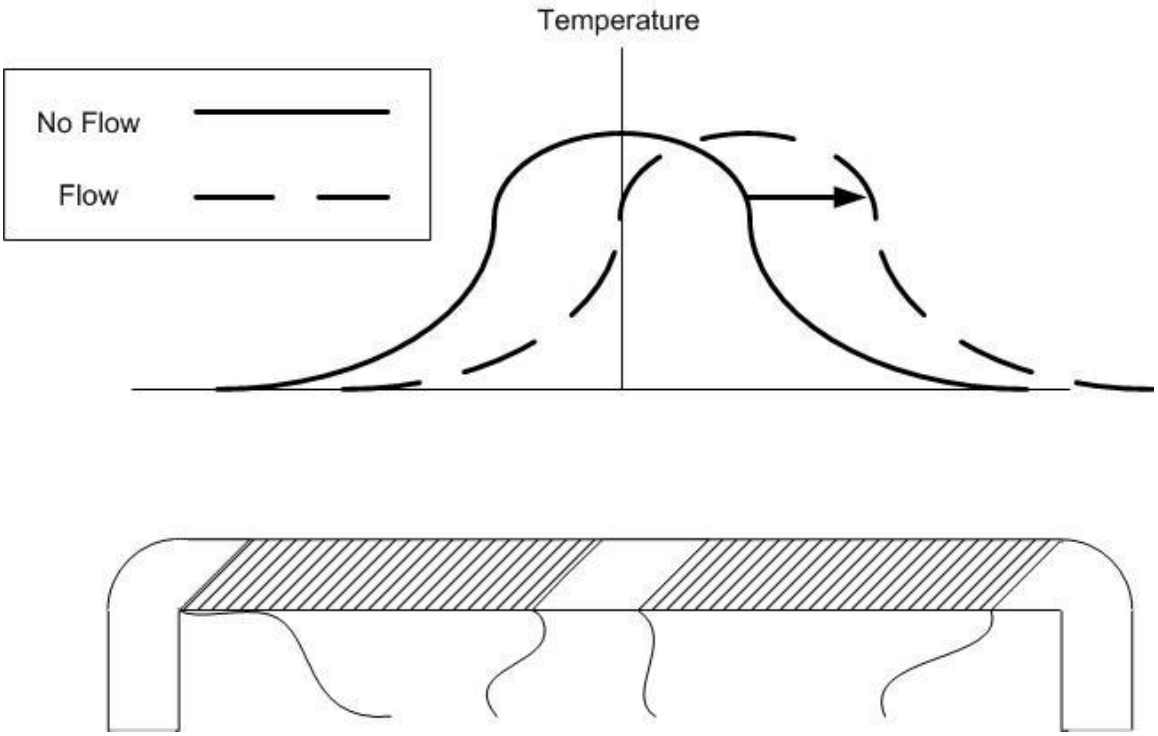

Figure 3-16 Temperature shift

Software is used extensively by the MFCs to read/set parameters and to read/clear error flags. These parameters are grouped into menus. These menus include the sensor, calibration, gas, valve, and mode menus. An example of the valve menu is shown in Figure 3-17. The sensor list includes generic information about the MFC including serial number, firmware version, etc. The calibration list consists of parameters defined by the manufacturer specific to each MFC. After calibration, the gas list is filled with parameters to convert raw data measured by the MFC into appropriate values for various different gases. This is needed as different gases will react differently to the heating coils and measurement devices. The valve list contains settings as to how the valve will react and to what stimuli. The valve's reaction can be altered by changing PID control coefficients. In addition, the valve list allows the user to determine appropriate source for commands. This can include using an external voltage signal or network defined setpoints. The mode menu contains information about current or latched warning and error flags set in internal registers.

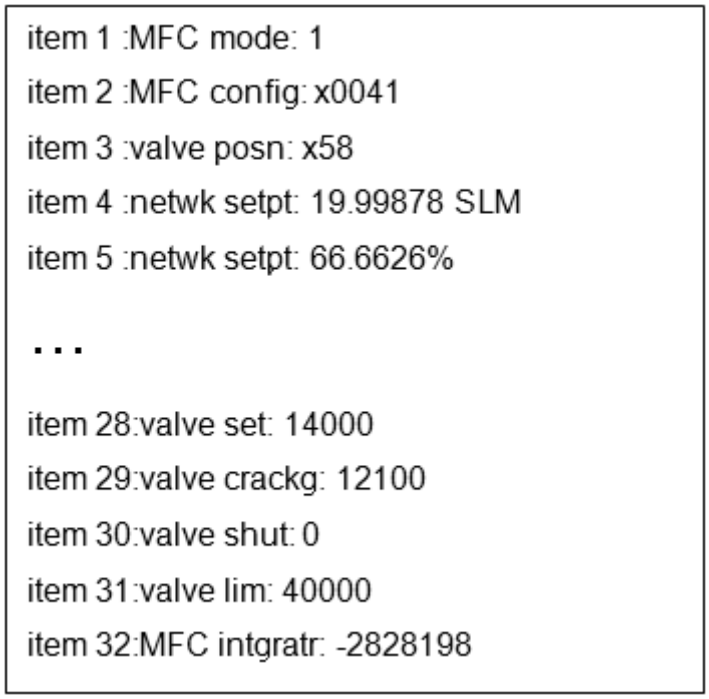

Figure 3-17 MFC valve menu

3.3. Cuvette system software

The software used for the cuvette data acquisition and control system was developed in Visual Studio and written in the Visual C# programming language. First, the graphical user interface is discussed, followed by the main software flow. A feedback loop added to the main software is then discussed. Additional software used by the cuvette system will be briefly discussed at the end.

3.3.1. Graphical User Interface

An image of the initial graphical user interface (GUI) is shown in Figure 3-18. Most of the remote units mentioned in earlier sections are represented in the cuvette program by tabs, as shown in the picture. The tabs for differential and absolute IRGAs are virtually identical with a graphing control on the left side and a textbox on the right side. The graphing control, known as ZedGraph, uses a dynamically linked library (DLL) that is licensed under the Lesser General Public License (LGPL). This license allows other users to freely use the software library as long as no changes are made to the library. The textbox is used to display the raw data being read from the IRGAs.

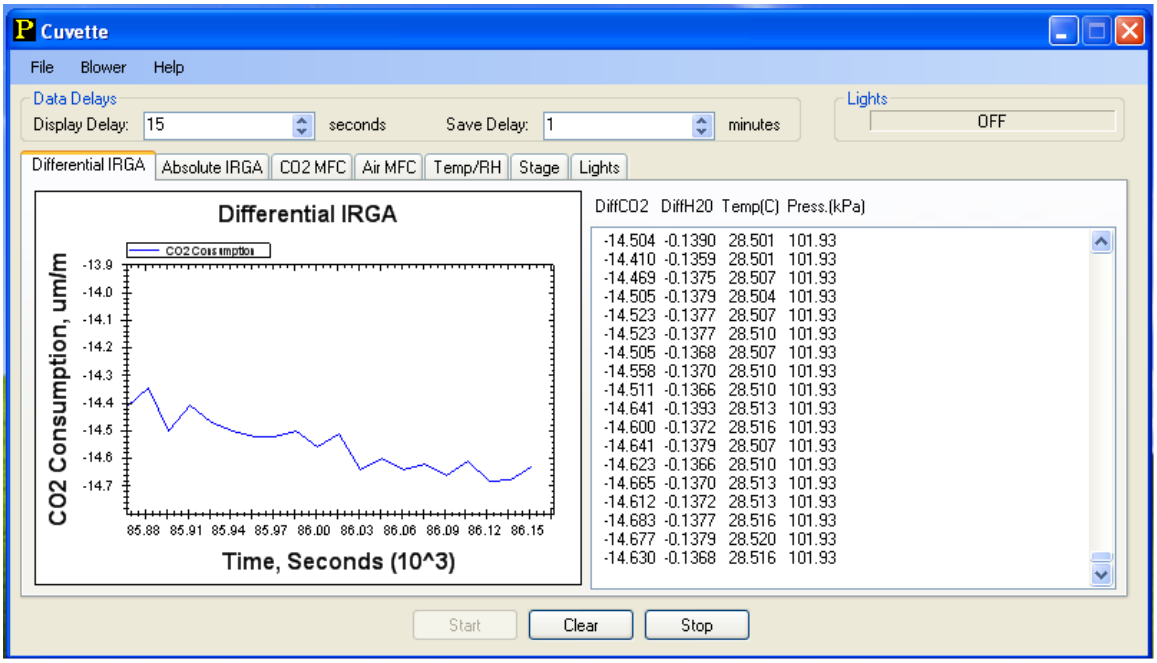

Figure 3-18 Initial GUI and IRGA tabs

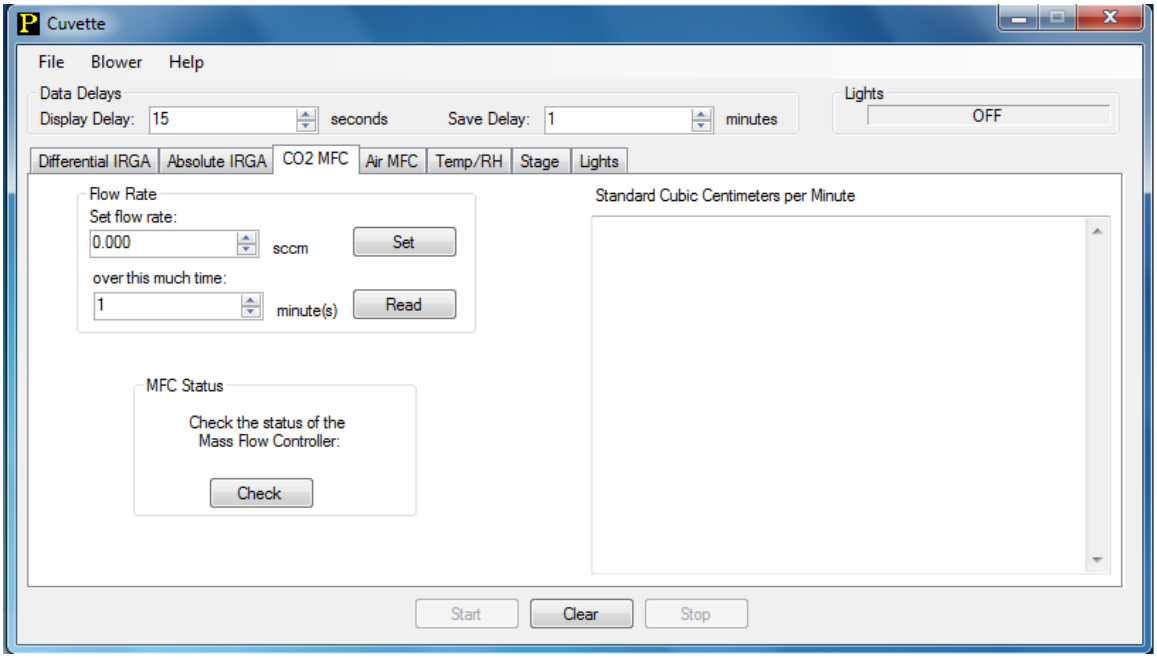

Figure 3-19 MFC tabs

The tabs for the CO_2 and air MFCs are also virtually identical and can be seen in Figure 3-19. There are controls on the upper left to read and set the setpoints for each MFC over a period of time. On the right is a textbox, which is used to display the raw data being read from the MFCs. The bottom left contains a button used to open the MFC Status window, as shown in Figure 3-20. The MFC are capable of storing and clearing present and latched warnings and alarms within registers. The MFC Status window queries and clears these registers in the upper portion of the window. User-defined setpoints, based on flow through the device, are used to determine when these warnings or alarms should be raised. These are set or read in the lower portion of the window.

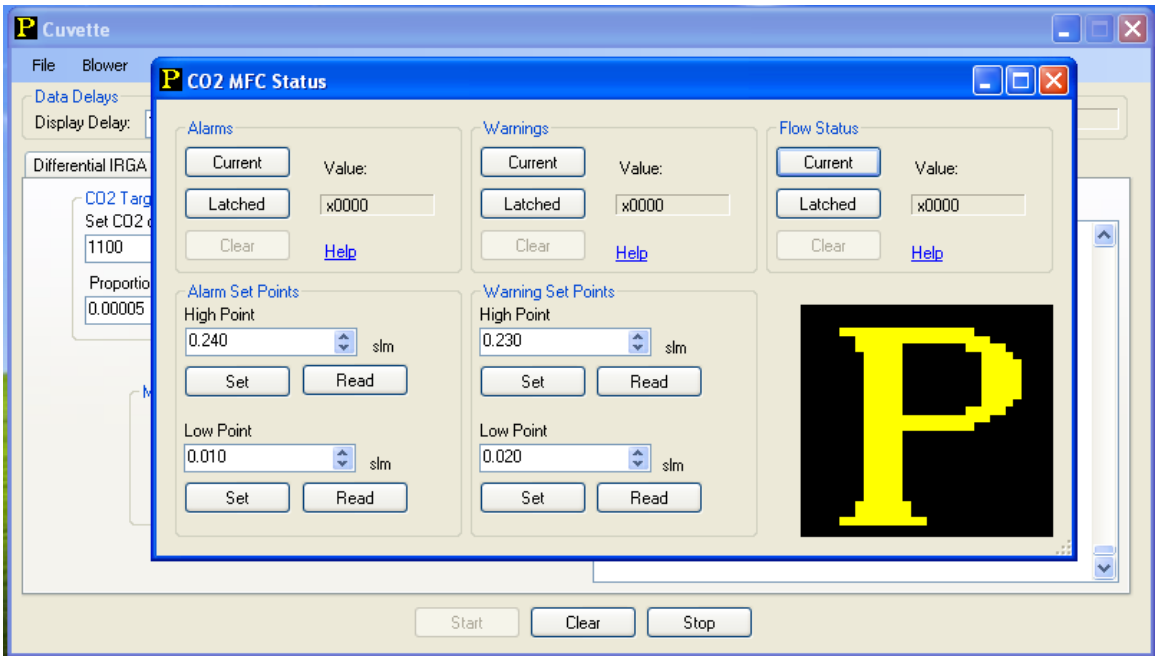

Figure 3-20 MFC Status window

The tab for the temperature and relative humidity (RH) probes is shown in Figure 3-21. The left portion of the tab displays the converted values of temperature and RH for each probe. The right portion of the tab displays the raw data being read from each probe.

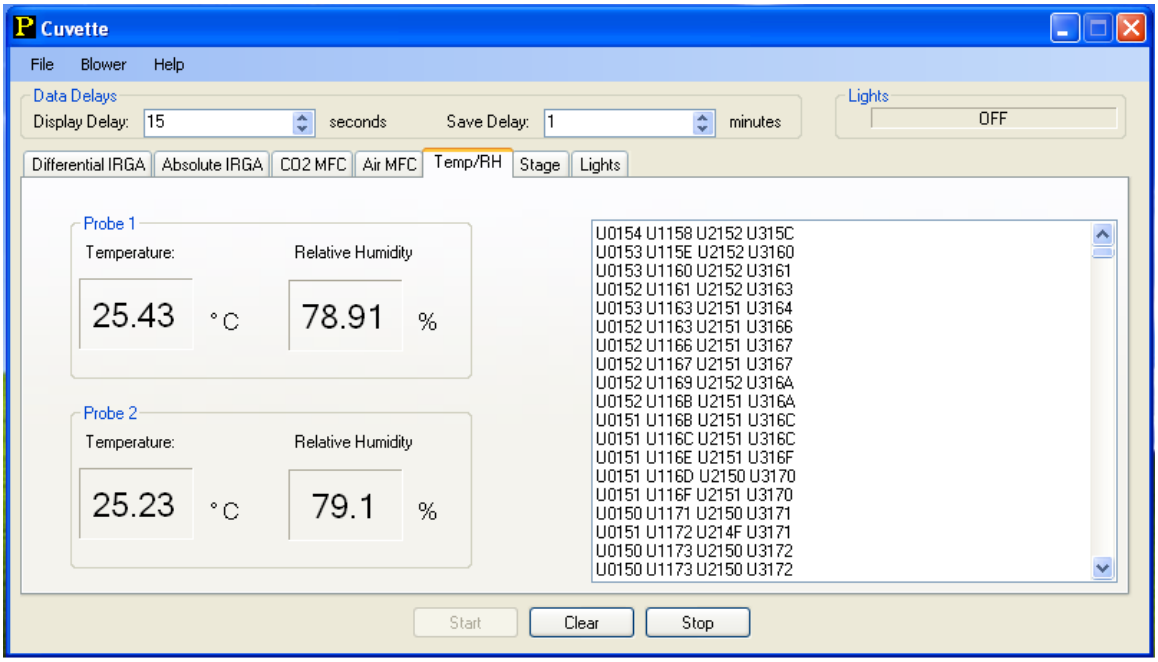

Figure 3-21 Temperature and RH tab

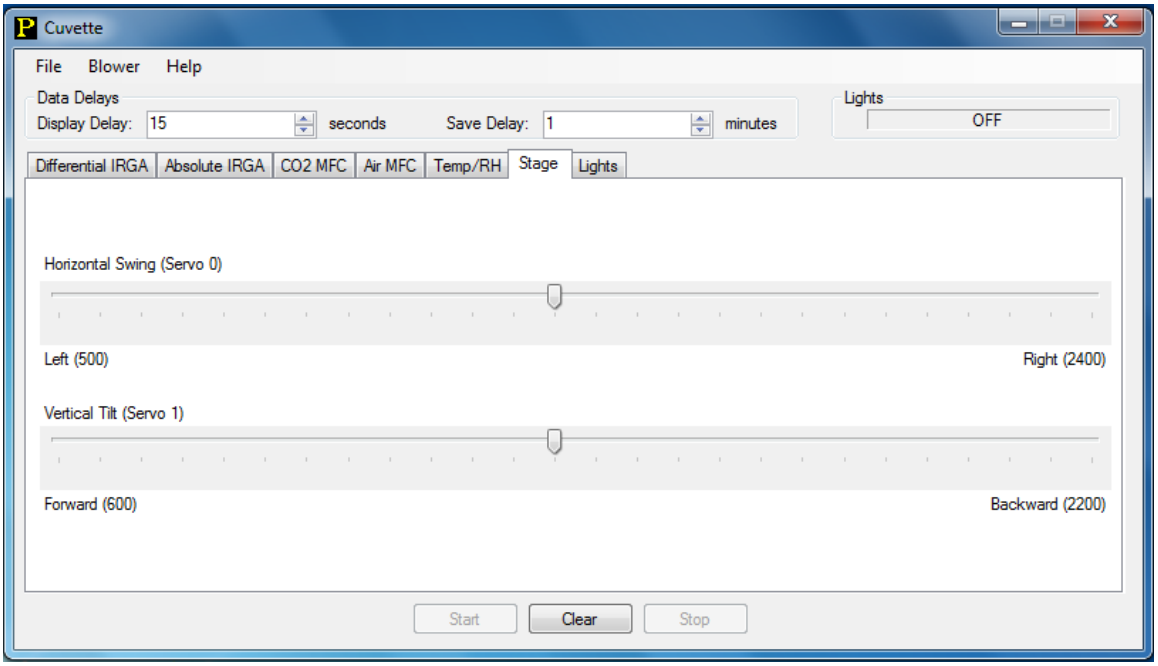

Figure 3-22 Webcam stage tab

The tab for the webcam stage is shown in Figure 3-22. It contains two trackbars used for adjusting the stage forward/backward or right/left. Each time a trackbar is moved its relative position is used to calculate the new position for the appropriate servo on the stage.

One of features requested by scientists was an indicator showing when lights inside of the cuvette should or should not be on. This was accomplished using the Lights tab shown in Figure 3-23. The user sets each setpoint to create the desired schedule. Each time data is collected, the cuvette program finds the most recent setpoint time in the past or present, compares it to the current PC time, and determines from that setpoint's button whether the lights should be on or off and displays this in the top right of the main GUI.

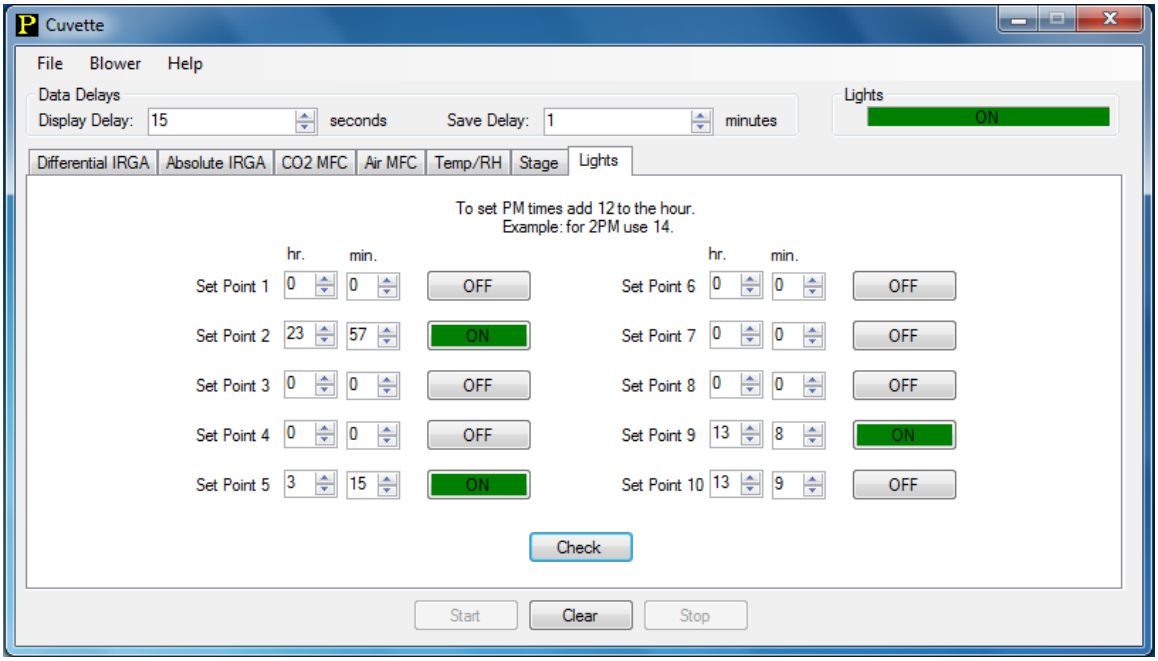

Figure 3-23 Lights tab (PC time = 1:08 p.m.)

Other features of the GUI include a pdf help file, about window, and message box displaying how long the cuvette blower has been on. The blower time is determined from the cumulative time the program has been running since

the last time it was reset by the user. While the program is not running, this value is saved as an application setting on the user's computer.

3.3.2. Main software flow

The cuvette program is composed of two main topics: data collection and storage. Data collection is accomplished by periodically querying or reading each data device in the system and updating the GUI. Data storage is accomplished by storing the each new interval of data into an Excel spreadsheet format for later analysis.

Before data collection can be discussed it is necessary to first describe its storage. Figure 3-24 shows a flowchart for the load event. It should be noticed that the very first thing the cuvette program does is query the operating system if there are any processes called Excel and, if so, with the user's permission, kill those processes. This is necessary because if the cuvette program is running (using Excel) and the user opens another spreadsheet file it will be opened in the current instance of Excel, rather than opening a new instance. This becomes a problem if the user inadvertently closes the Excel instance. The cuvette program would no longer have an Excel instance to write to, causing crashes. So, Excel should not be used by the user while the program is running. After this check, a new workbook is opened, and the ZedGraph controls are initialized.

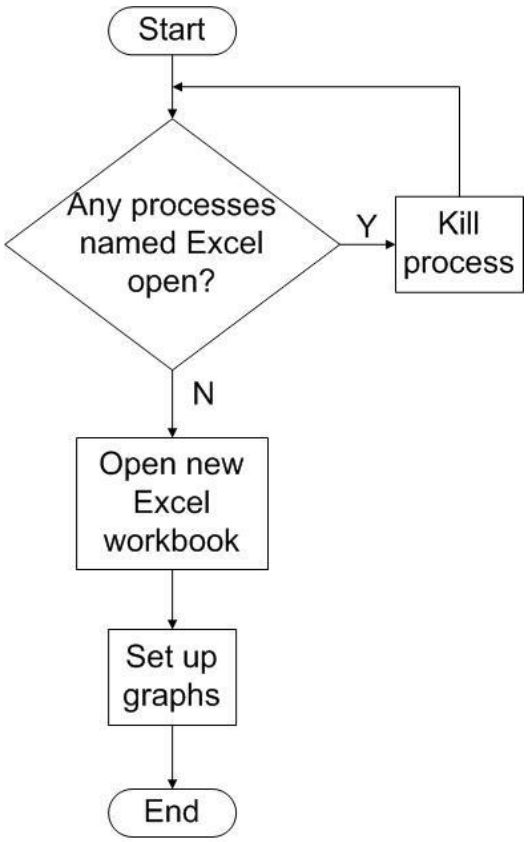

Figure 3-24 Load flowchart

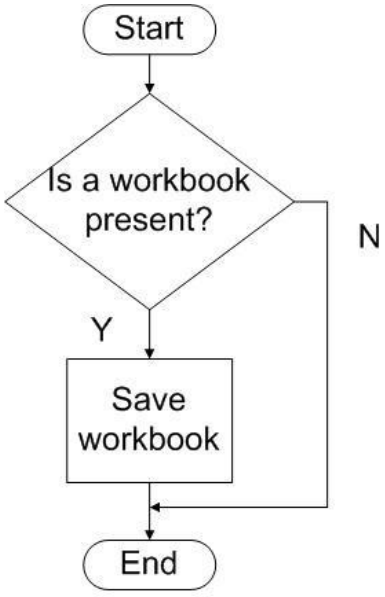

Figure 3-25 Excel save flowchart

A couple of methods used throughout the cuvette program are saveExcel and saveAsExcel, shown in Figures 3-25 and 3-26. The method saveExcel ensures there is a workbook to save before saving. The method saveAsExcel opens a file dialog to have the user save the filename. The cuvette program uses separate worksheets for each of the five data devices. As will be below, these workbooks may be new or preexisting. If they are new, Excel spreadsheets default to only three worksheets. The cuvette program adds extra worksheets, if they are needed, and then selects the first worksheet to be ready for data collection.

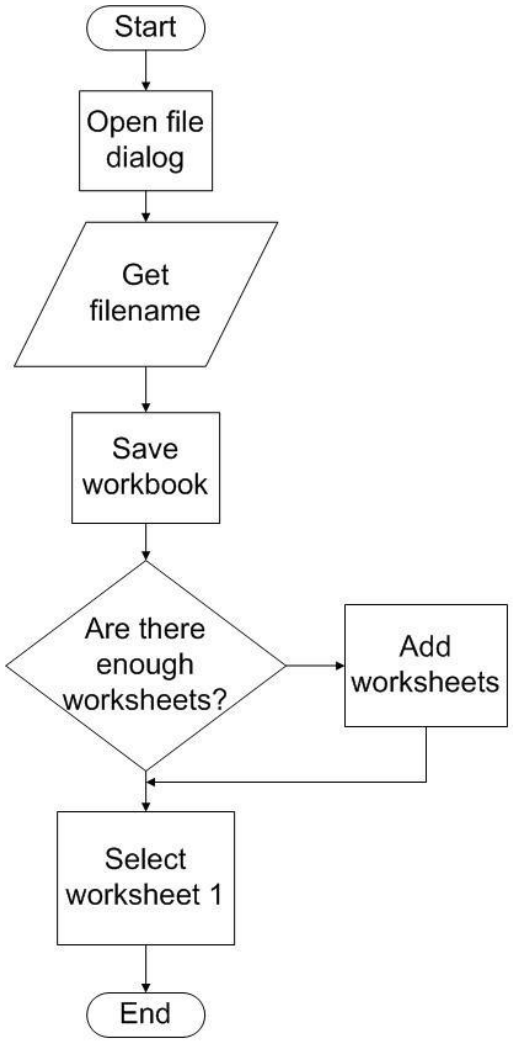

Figure 3-26 Excel saveas flowchart

Before saving the data to a worksheet, the cuvette program must know whether to start a new file or use an existing file. Figure 3-27 shows the program flow if the user elects to start a new spreadsheet. First, the previously defined saveAsExcel method is called. During this method the user has the option of cancelling during the open file dialog. So, the program once again checks to see that a workbook is present. It then activates the file's hidden attribute in an attempt to keep the user from manually opening and closing the file, for similar reasons as the load event method. Then various buttons are enabled or disabled to allow the program to begin data collection.

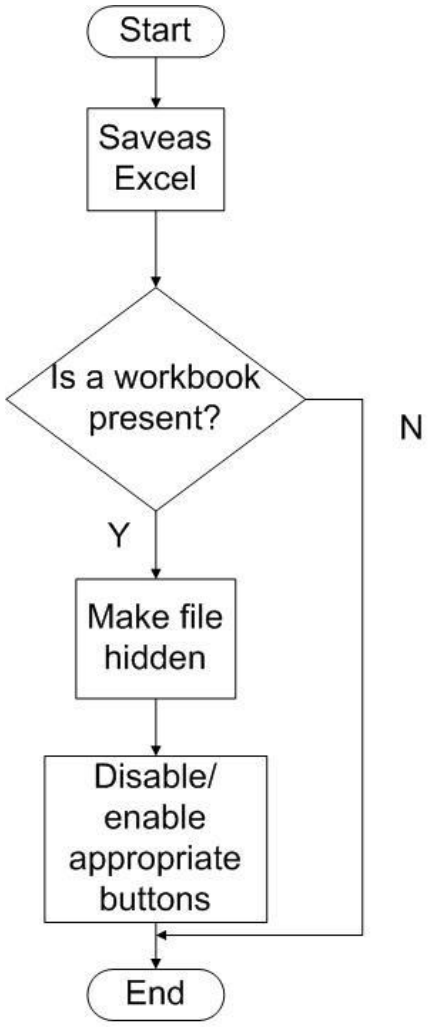

Figure 3-27 Excel new file flowchart

Figures 3-28 and 3-29 show the program flow to append new data onto a file with existing data. A file dialog is opened to get the filename. The filename is then checked to see if it's empty. If the file exists, it's then checked to see if it is already in use by another user. As data files can get rather large, it may take some time to find end of the file, so a "waiting" window is opened and the main window minimized to discourage the user from closing the program during this time. The spreadsheet file is then opened and checked to ensure it has the proper number of worksheets. It then selects the first worksheet and selects cell A1 as its range. The cell range is the collection of cells that can be manipulated by giving a command or, in other words, the currently selected cells. The cuvette program then sequentially checks each cell in column A for values. It continues until an empty cell is found. That row is then used on all worksheets to begin writing new data. The "waiting" window is then closed and the main window is restored.

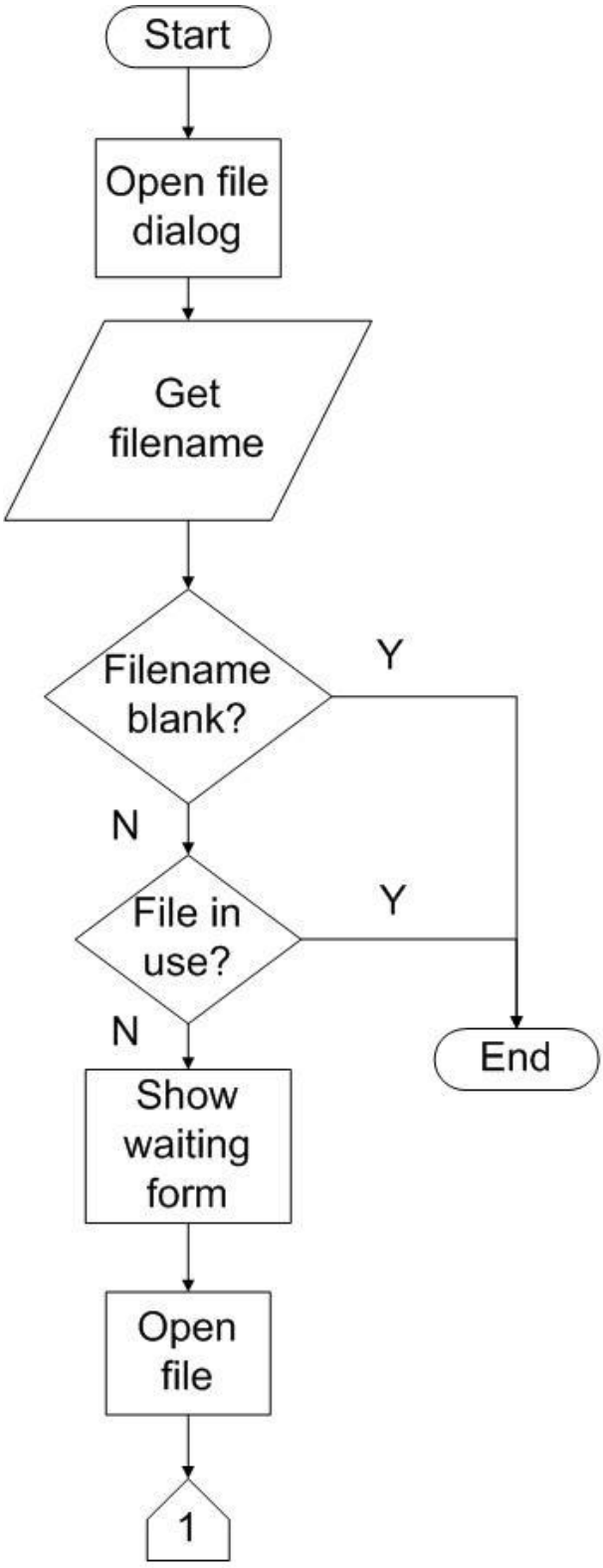

Figure 3-28 First Excel append file flowchart

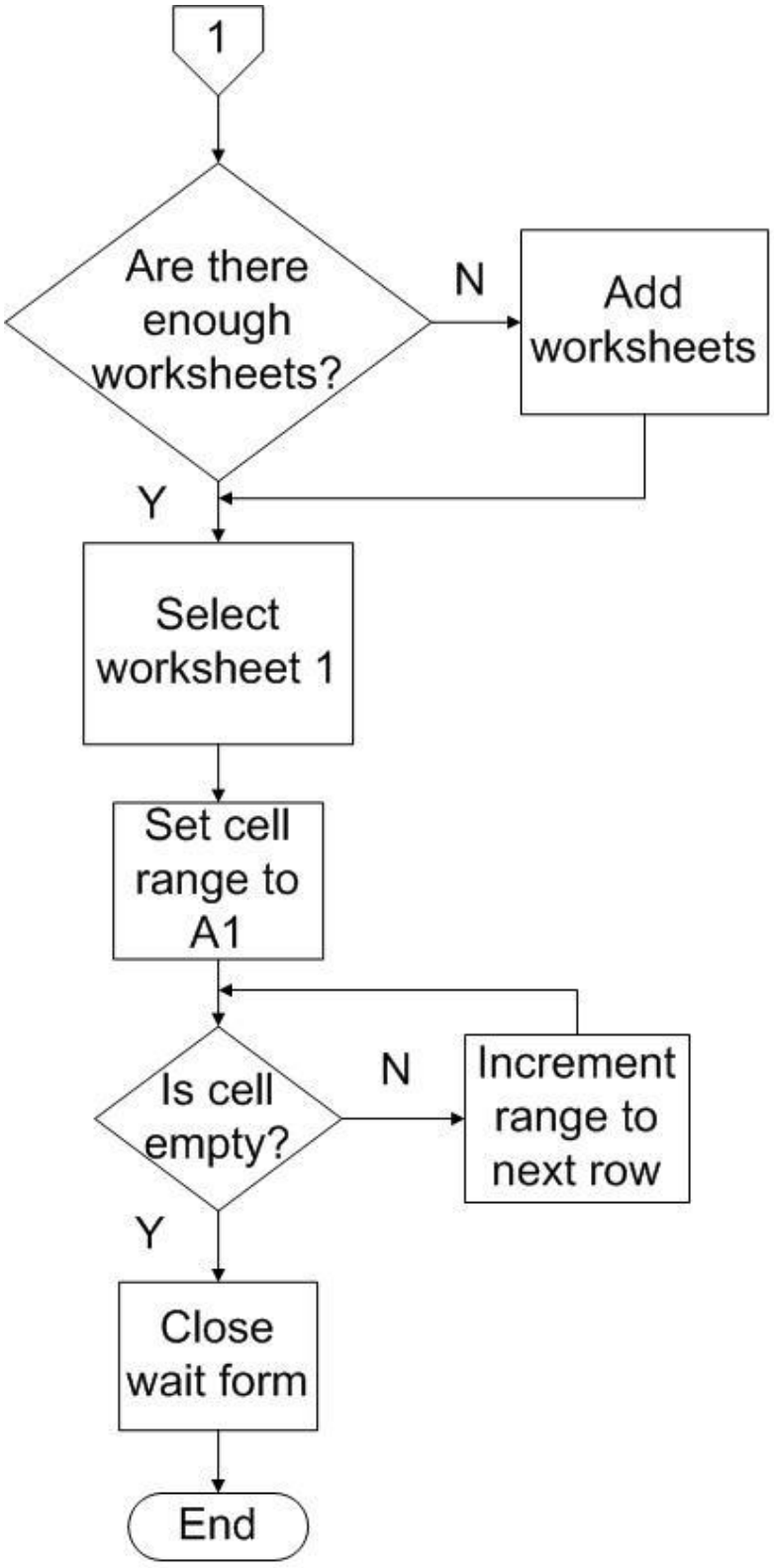

Figure 3-29 Second Excel append file flowchart

After the data storage is completed, the Excel instance must be closed. This is shown in Figure 3-30. The serial port is closed, if not closed already. The spreadsheet's hidden attribute is turned off. The workbooks and instance of Excel are closed and told to quit. All of the Excel objects created in the load method are set to null to allow garbage collection to reclaim their memory. It has been noticed on various PC configurations (various Windows OS's, versions of Office, etc.) that this is not sufficient to close the Excel process. This can be seen in the Windows Task Manager. Similar to the load method, at closing it is necessary to close all open processes called Excel to ensure a proper exit of the cuvette program.

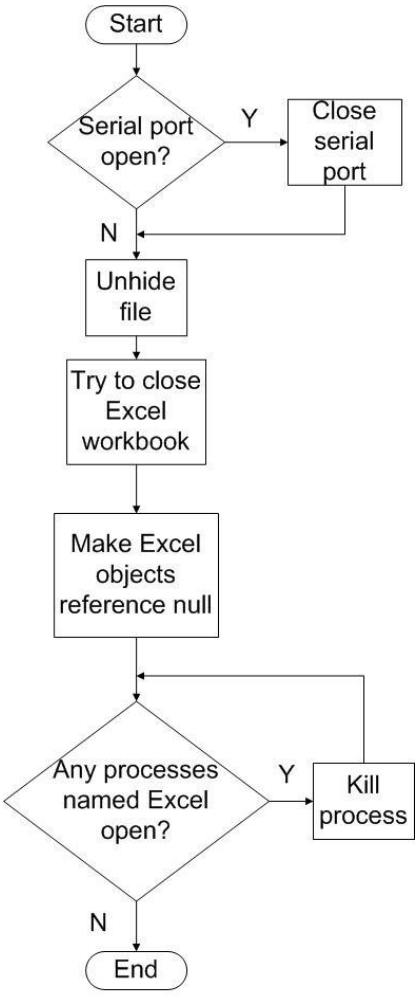

Figure 3-30 Excel closing flowchart

Now that data storage has been discussed, it's appropriate to discuss the data collection procedure. As mentioned previously, the data collection procedure is accomplished by periodically querying or reading each data device in the system and updating the GUI. This sounds deceptively straightforward, as will be shown in a moment.

After the user has started a new spreadsheet or appended to an existing spreadsheet, the user begins data collection by pressing the Start button at the bottom of the main window, as shown in Figure 3-31. This enables a timer within the program. The timer's period is determined by the Display Delay value at the top of the window. Each time the timer completes a period data are collected and displayed in the respective textboxes on each tab. The data are actually saved in the spreadsheet less frequently. This "save" period is determined by the Save Delay value at the top of the main window. The timer can be stopped by either pressing the Stop button at the bottom of the main window or exiting the program.

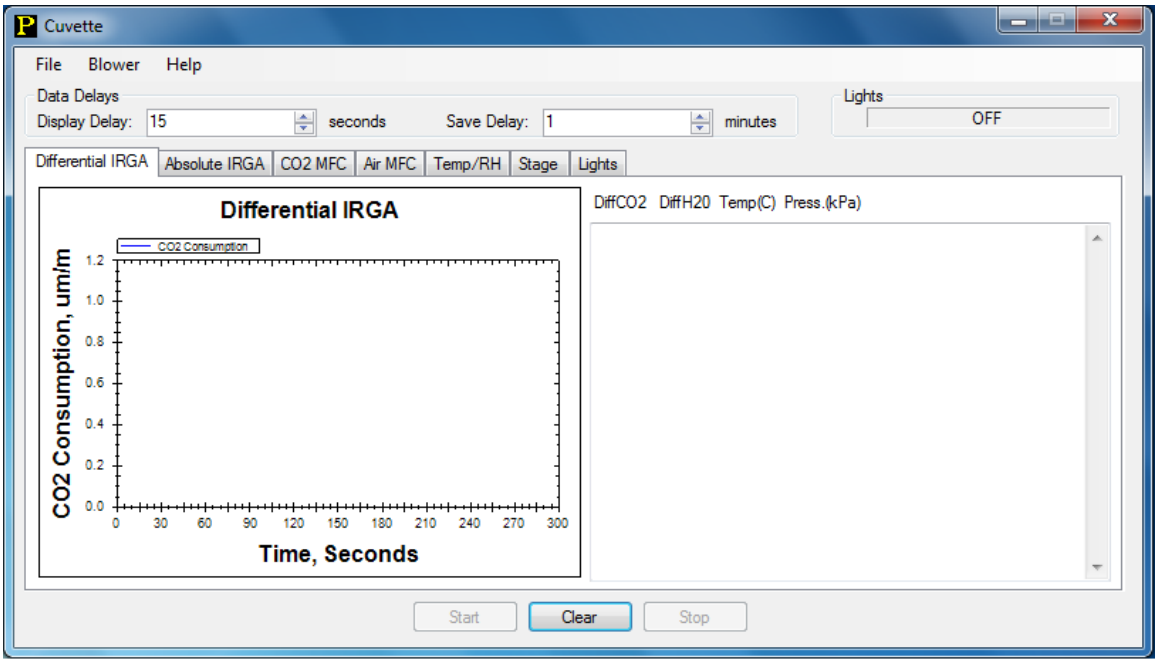

Figure 3-31 Main GUI window

The first issue that arises from attempting to collect data is an unresponsive GUI. The data devices communicate through a serial port on the personal computer. Serial ports are relatively slow to begin with. Add to that the fact that one of the data devices, the absolute IRGA, is a read-only device that transmits its data at a fixed period. This means the device can't simply be queried for its data. The probability is high that when the serial switches transfer to the absolute IRGA and data are read that the absolute IRGA will not be preparing to send its next message. It will already be at some stage of sending a data message. If this is true, the first data message read will be a fragment of data and must be thrown away. The cuvette program must then wait for the next full string of data. The timer that initiates the data collection procedure uses the same thread that the GUI uses, meaning it can't do both things at the same time. So, when the timer completes its period and the data collection procedure is being run, the GUI of the cuvette program will be unresponsive. If the timer period is long enough this would still allow the user to use the GUI, but only between data collection procedures.

So, in order to allow the GUI to remain responsive at all times, the timer must spawn a new thread for the data collection procedure to run on. Each thread will take turns using the computer's processor so that both can run essentially at same time. However, multithreading introduces two new problems: resource sharing and thread communication.

Since the timer spawns a new thread each time it completes a period, there is the potential for more than one data collection procedure to be running at once. This could happen when the computer is given a heavy workload, such as opening another program. If the first data collection thread is using the serial port and a second data collection thread tried to use it too, it would corrupt the data. Part of the data message would be read to the first thread, with the remainder going to the second thread. This section of the program is called a critical section. To protect a critical section from other threads, a software lock is placed around this section of the program. If the code is in use by a thread, no other

thread can access that part of the program until the lock is removed by the original locking thread.

At this point we have the GUI thread and at least one data collection thread. So, after the data collection thread is finished it should update the controls on the GUI. Wrong! The controls on the GUI are not thread safe, meaning if multiple threads try to change a control the results are unpredictable. The only thread that should manipulate the GUI controls is the GUI thread. In order for the data collection thread to safely change the GUI controls it must invoke the changes through the GUI thread (H. M. Deitel & P. J. Deitel, 2006). Wergerson (2007) provides an excellent example of how to make changes to the GUI controls from other threads.

Figures 3-32, 3-33, and 3-34 show the program flow for the data collection procedure. After the first worksheet is selected, the differential IRGA is read. As reading from each of the five data devices requires similar steps, they will be covered in general after the overall program flow has been discussed. The ZedGraph for the differential IRGA is then updated by sending the data to the GUI thread. Worksheet two is selected, the absolute IRGA is read, and the ZedGraph for the absolute IRGA is updated. Then, the CO_2 mass flow controller is read and saved to worksheet three, the air mass flow controller is read and saved to worksheet four, and the temperature and relative humidity probes are read (through their ADC) and saved to worksheet five.

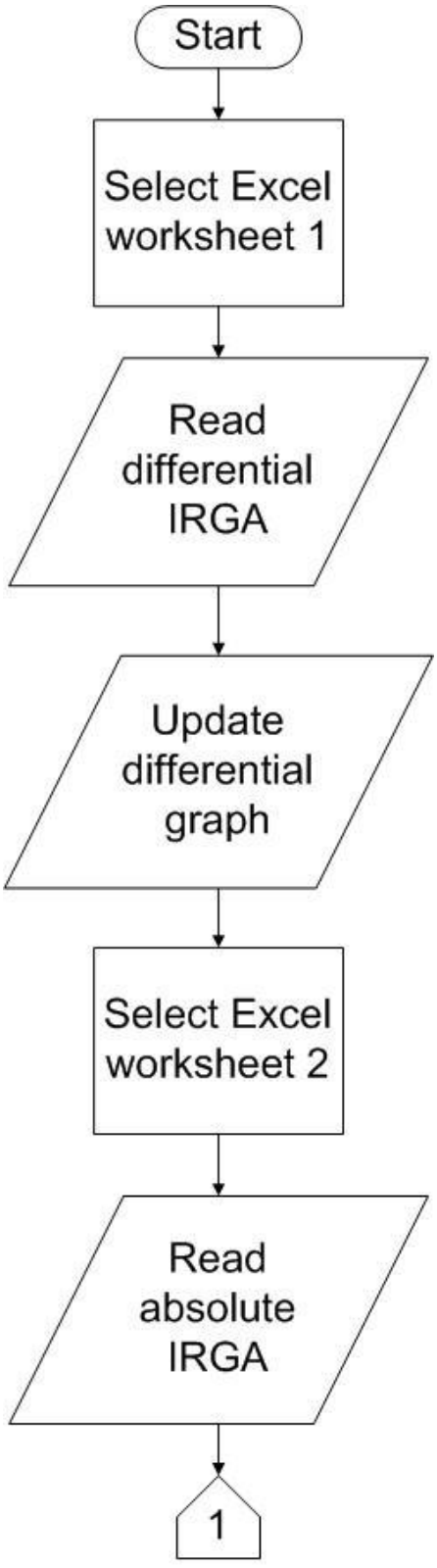

Figure 3-32 First data collection flowchart

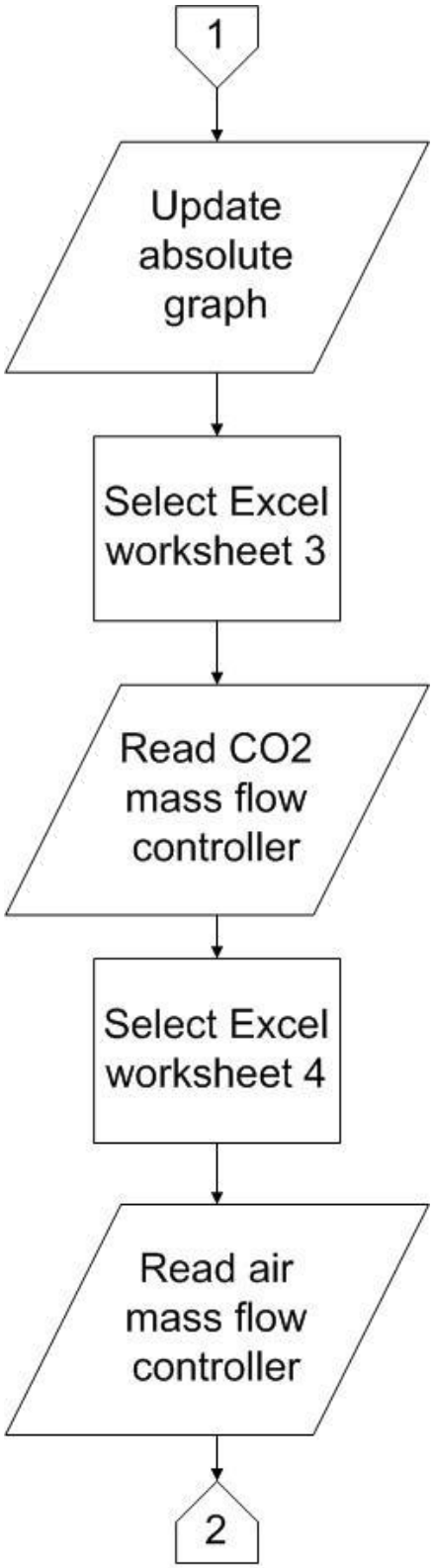

Figure 3-33 Second data collection flowchart

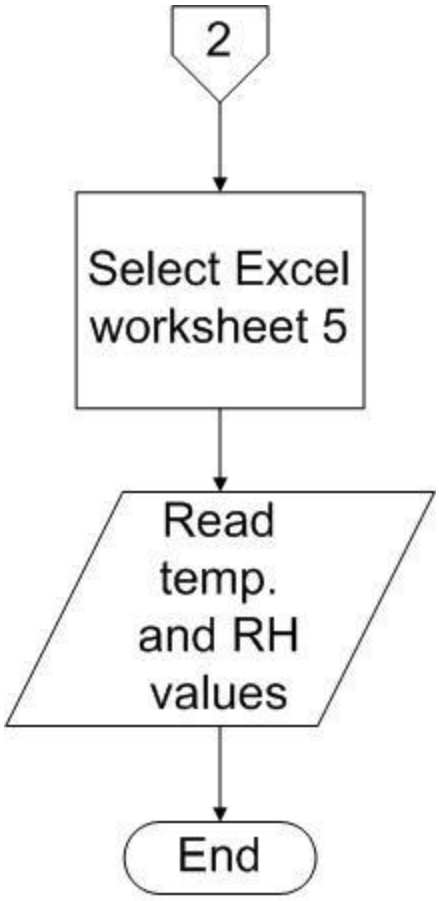

Figure 3-34 Third data collection flowchart

Each of the five data devices is read sequentially and has its data saved to an appropriate worksheet. The general program flow for reading each device is shown in Figures 3-35, 3-36, and 3-37. As mentioned previously, when the serial port is being used it must be locked to prevent other threads from interfering. The serial port is opened, if not open already. A command is send to the serial switches to open the appropriate port to communicate with the desired data device. The serial port on the computer has a buffer that stores incoming data until it is used. Any data in there when the serial port is first opened is not valid, so the buffer is cleared. A line of data (denoted by a new line character) is read from the data device. As mentioned previously, this first data message is most likely a data fragment, so it is discarded. Another line of data is then read from the device to get a complete data message. The port on the serial switch is closed, and the lock on the computer serial port is disabled.

When the data message is read, it is in the form of one large string of data. First, the individual pieces of data within the large data string must be broken up or parsed. Each device has a unique set of delimiting characters that separate the individual pieces of data. For example, the delimiting characters for the absolute IRGA are a single space, double space, triple space, quadruple space, quintuple space, carriage return, and newline character. When these delimiting characters are removed, an array of individual data pieces is left. However, it is still an array of strings. After a time stamp (from the computer clock) is inserted at the beginning of the array, the remainder of the array's string data is converted into numerical data. This is necessary to store the data correctly in the spreadsheet. The cell range is selected, based on the length of the numerical array, and saved to the respective worksheets. Finally, the data are sent to the GUI thread so that it can update the respective textboxes.

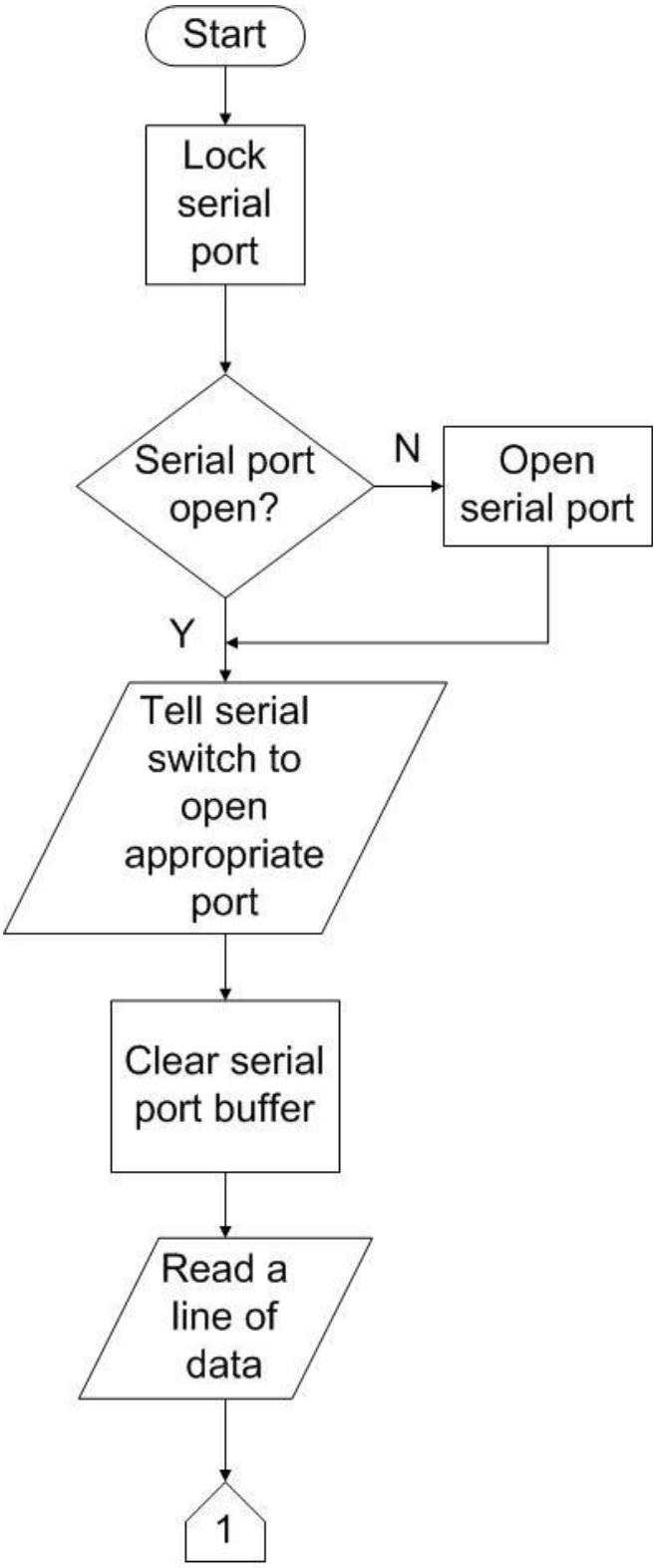

Figure 3-35 First general data read flowchart

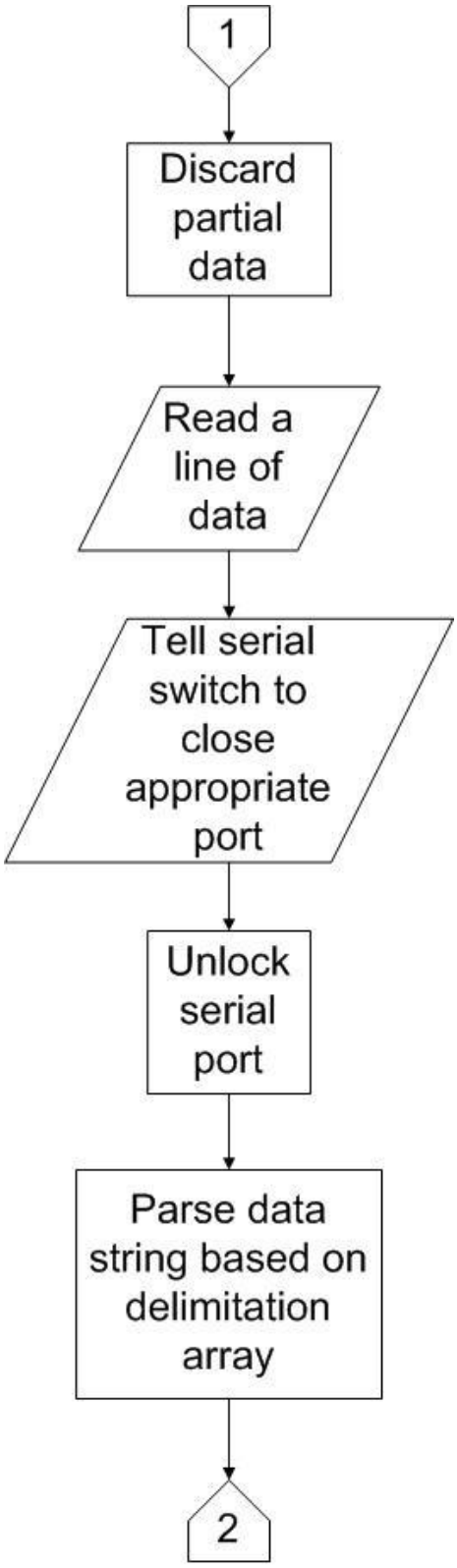

Figure 3-36 Second general data read flowchart

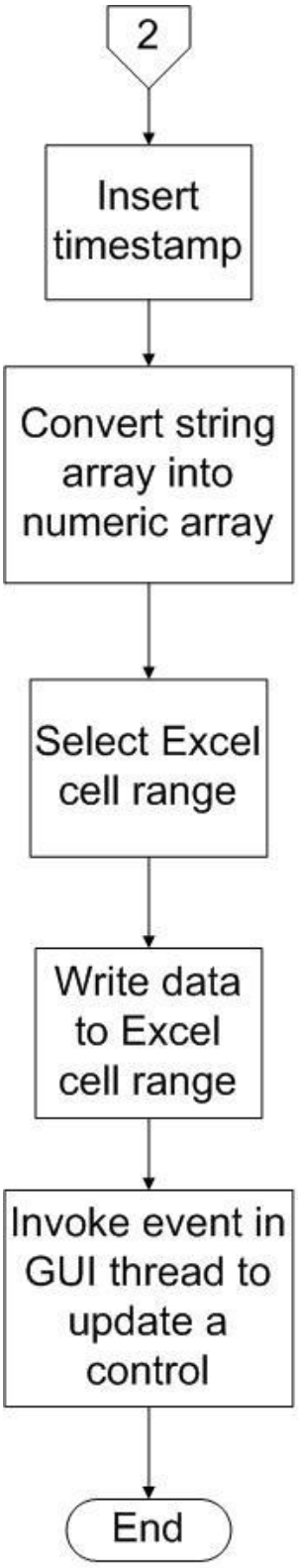

Figure 3-37 Third general data read flowchart

3.3.3. Feedback loop addition

Recent modifications to the main portion of the software have been implemented to improve multithreading stability and to introduce a feedback loop.

In Figures 3-35, 3-36, and 3-37, for each device the serial port is locked, its data read, the serial port is unlocked, and then the data is written to its respective worksheet. However, since the timer is still periodically spawning data collection threads, there exists the potential that one of the collection threads might not finish before another begins. This could lead to multiple threads reading properly from devices, but saving the data to the wrong worksheet on a different thread. This potential issue has been rectified by modifying the main software flow as shown in Figure 3-38.

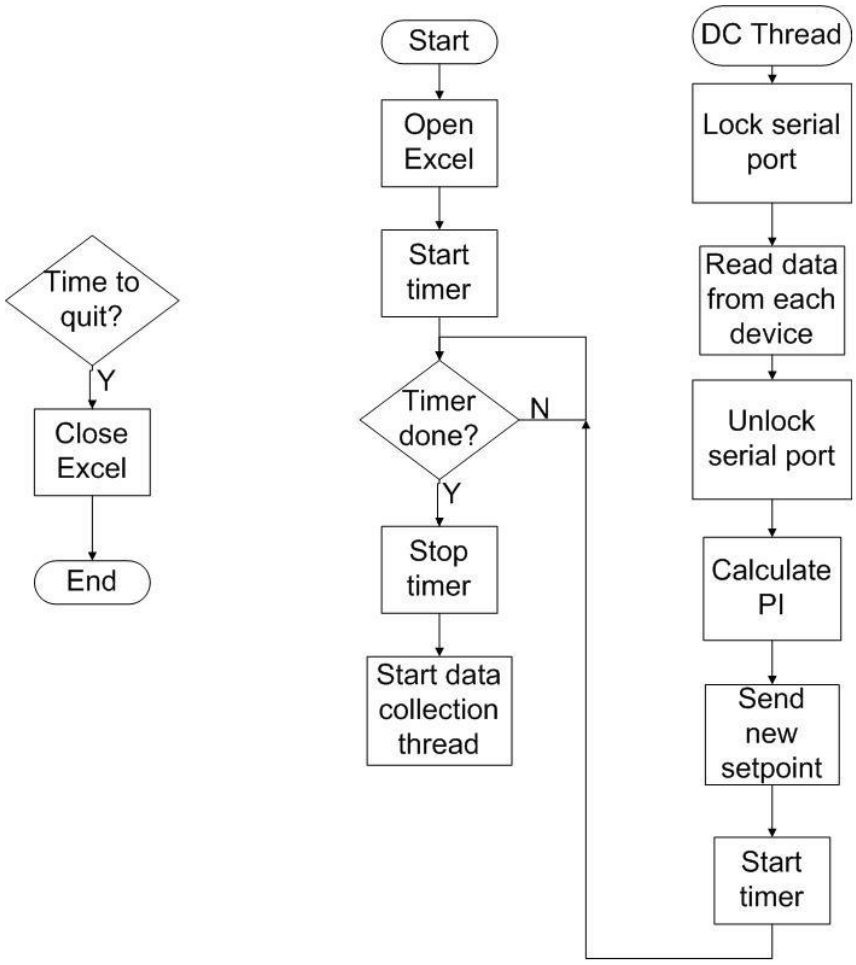

Figure 3-38 Modified overall software flow

This modification to the main software structure stops the timer after the timer event occurs to prevent new threads from spawning. After the data collection thread is initiated, the serial port is locked until all of the devices have been read and their values written to their proper worksheets. After the data collection thread has finished, the timer is restarted. It may be necessary in the future to implement a watchdog timer, in case the data collection thread never finishes, but the most current testing has not shown a need for it.

A feedback loop was also introduced recently. This feedback loop makes partial use of a PID algorithm (Proportional-Integral-Derivative), using just the P and I terms for our loop. The feedback loop reads from the absolute infrared gas analyzer to determine the current concentration of CO_2 being sent to the cuvette chamber. It then calculates the error between the current concentration and a setpoint the user is attempting to achieve. The error is used to command the mass flow controller connected to the CO_2 canister using Equation 6.

$$x[n]_{CO2\ MFC} = x[n\text{-}1]_{CO2\ MFC} + K_P E + K_I E_I \qquad (6)$$

where

$x[n]_{CO2\ MFC}$ = the next CO_2 MFC value

$x[n\text{-}1]_{CO2\ MFC}$ = the last CO_2 MFC value

K_P = Proportional Gain (user defined)

E = desired setpoint – current CO_2 concentration

K_I = Integral Gain (user defined)

$$\mathrm{E_I} = \sum_{0}^{n-1} E_I[n-1] + E$$

3.3.4. Additional cuvette system software

In order for the data files to be accessible via the Internet, a simple webserver was installed and setup on the personal computer. Apache HTTP Server was chosen due to its ease of use and long history of reliability. It can be obtained from the Apache Server Foundation website. The server can be further configured by altering the “httpd” file. When initially setting up the server, all firewalls should be turned off.

The webcams used in this project needed an application program that could periodically (like the data devices) capture images of the plants within the cuvette. A freeware program called AvaCam was used for this project. This program allows users to manually take pictures, have pictures taken when motion occurs, or take pictures periodically. It also records video. More than one webcam can be used with this program, provided two instances of the program are running. If the webcams are the same model, Windows may get confused by having two devices with the same driver/name. This may be overcome, as it was in the cuvette system, by connecting to different USB hubs on the computer.

CHAPTER 4. FINDINGS

The purpose of this thesis was to design, assemble, and program a data acquisition system that can be used for a cuvette. The following findings demonstrate the ability of the system to carry out these goals.

4.1. Initial Findings

Data was collected from the system without plants over a period of approximately a week. The data needed to be normalized due to erroneous data. The erroneous data are shown as spikes in the differential gas analyzer (IRGA) data, as seen in Figure 4-1.

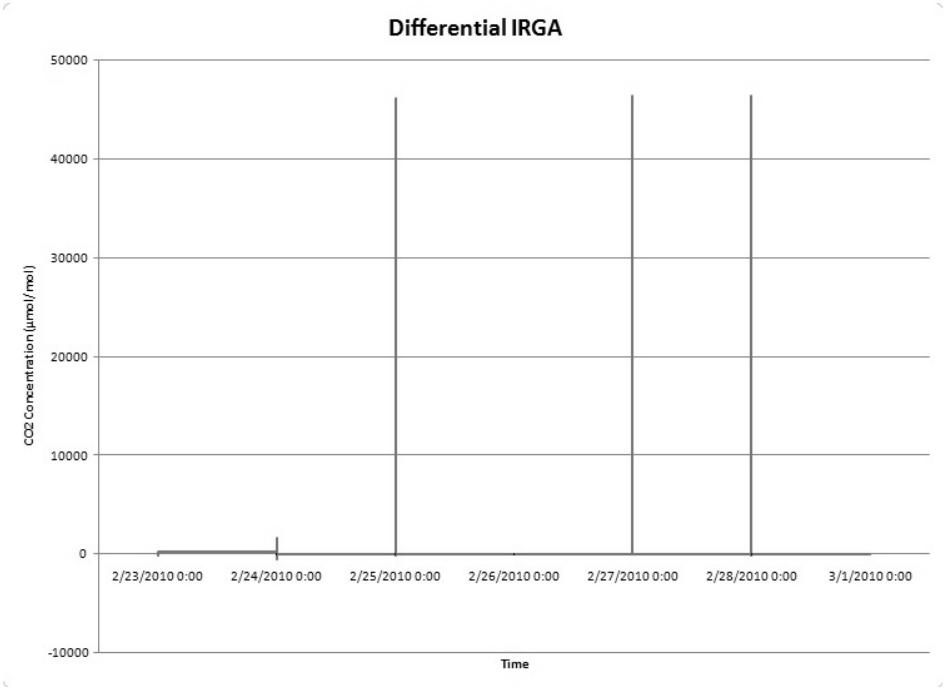

Figure 4-1 Differential data with erroneous data

After investigation, it was determined that data from the various pieces of lab equipment were occasionally being written to the incorrect worksheet in Excel. Nearly every day at approximately 2 a.m. to 3 a.m., one to two rows of data were written to the incorrect worksheet. While the exact cause remains unknown, the normalized differential data can be seen in Figure 4-2.

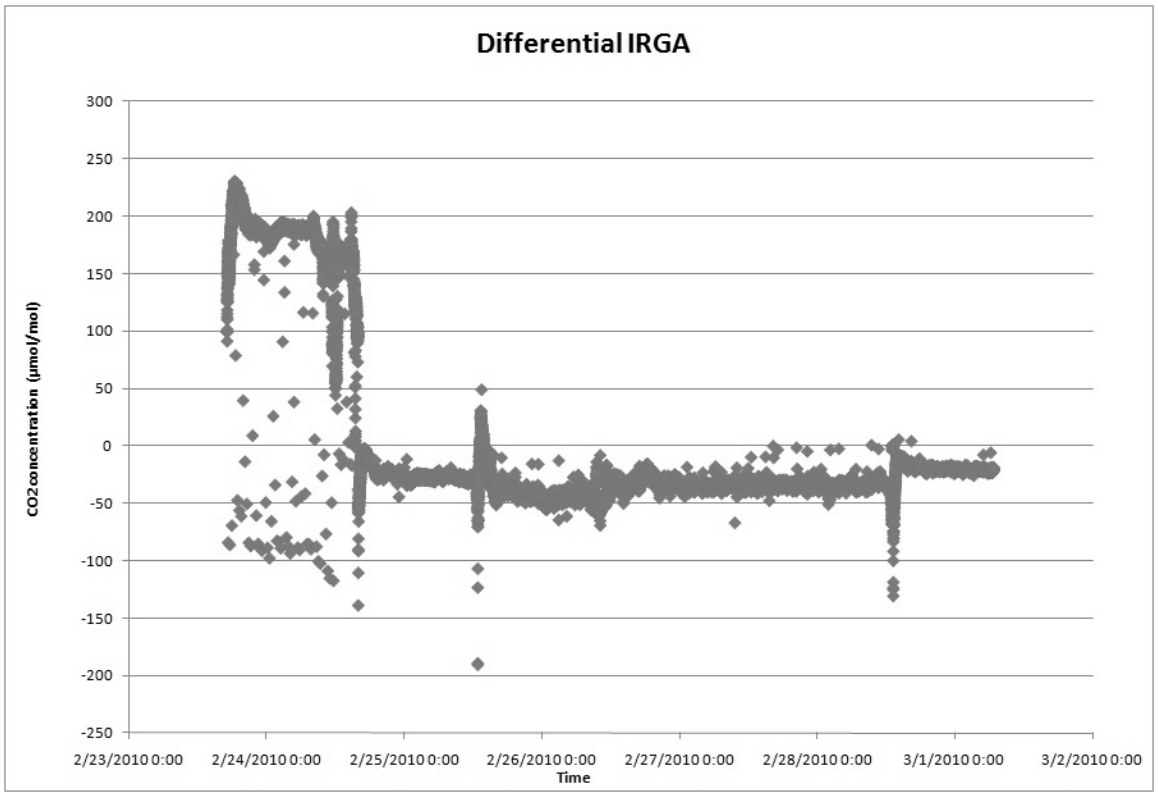

Figure 4-2 Normalized differential data

The initial large disturbance was a setup period for the cuvette system. After that point in time, the differential measurements enter a steady state at approximately -35µmol/mol. The two spikes, seen after the setup period, occurred when personnel entered and exited the room, adding CO_2 to the ambient CO_2 concentration. Since there were no plants in the cuvette to absorb CO_2, the expected steady state value was anticipated to be around 0µmol/mol. This would have indicated that the CO_2 concentration exiting the cuvette was

identical to the concentration that bypassed the cuvette. The offset in our results was caused by a peristaltic pump, used to push the gasses to the gas analyzers, which heated some of those gasses. Although an offset is present, Figure 4-3 demonstrates that between the two spikes the differential CO_2 concentration is fairly steady. Table 4-1 shows an average offset of -36.1µmol/mol with a standard deviation of 6.3µmol/mol.

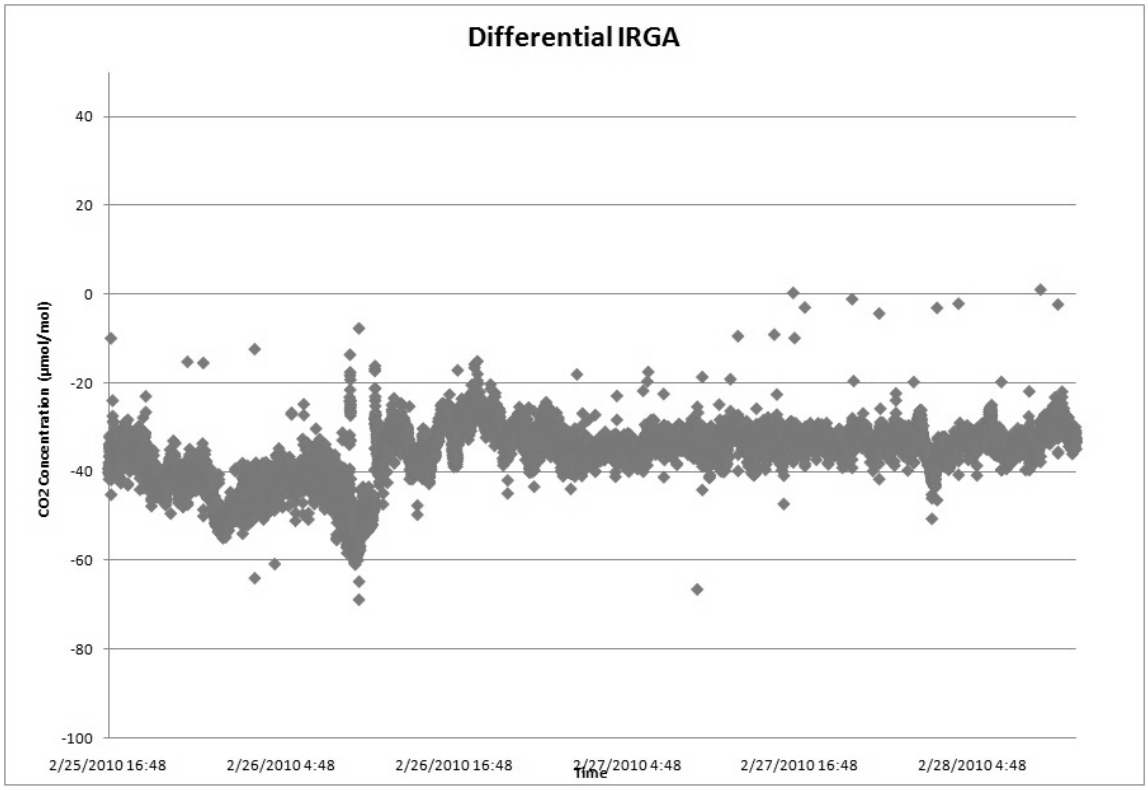

Figure 4-3 Steady differential concentration

Table 4-1
Differential IRGA statistics

Mean (µmol/mol)	Standard Deviation (µmol/mol)	Variance (µmol/mol)	95% Confidence Interval (µmol/mol)
-36.1009	6.2810	39.4508	0.1005

Figure 4-4 shows data from the absolute gas analyzer over the course of the week long test. Similar to Figure 4-2, there is a setup time, followed by two spikes. Figure 4-5 shows the steady state between the two disturbances. The absolute gas analyzer measures the enriched ambient CO_2 concentration, which varies as people move in and out of the building. Table 4-2 shows the average ambient CO_2 concentration at 419.7µmol/mol with a standard deviation of 12.0µmol/mol.

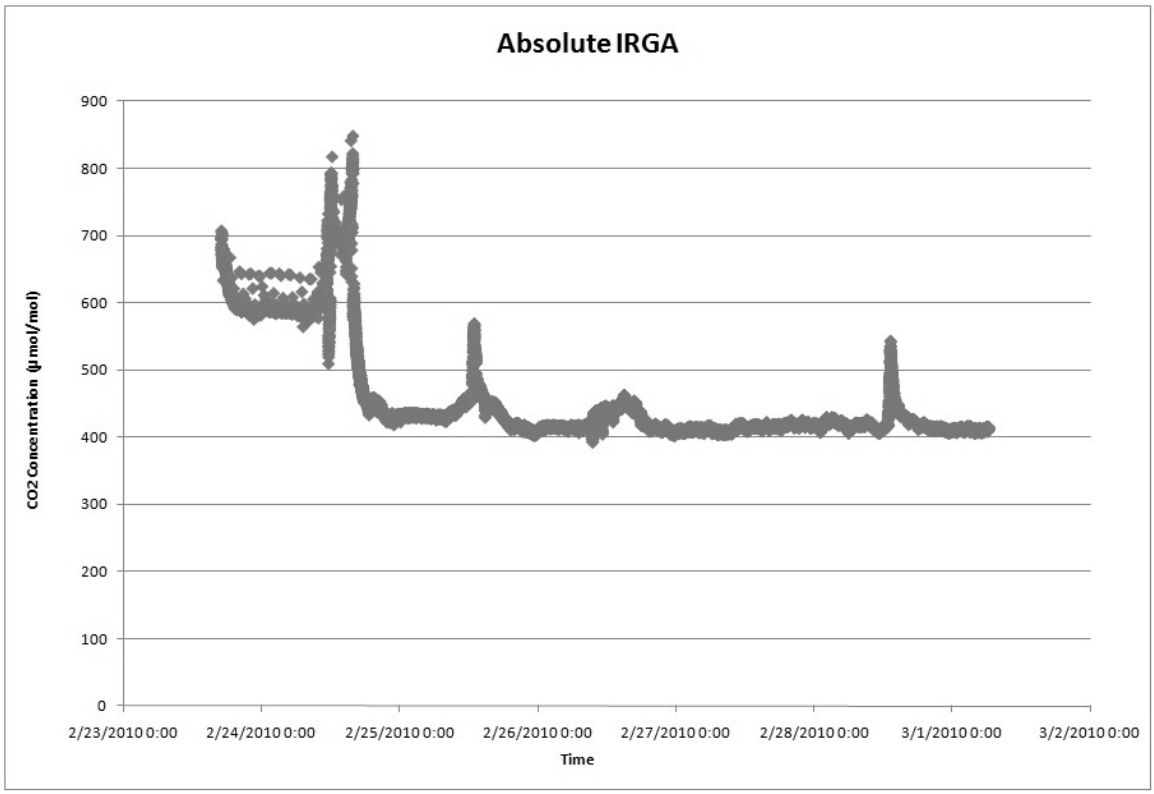

Figure 4-4 Normalized absolute data

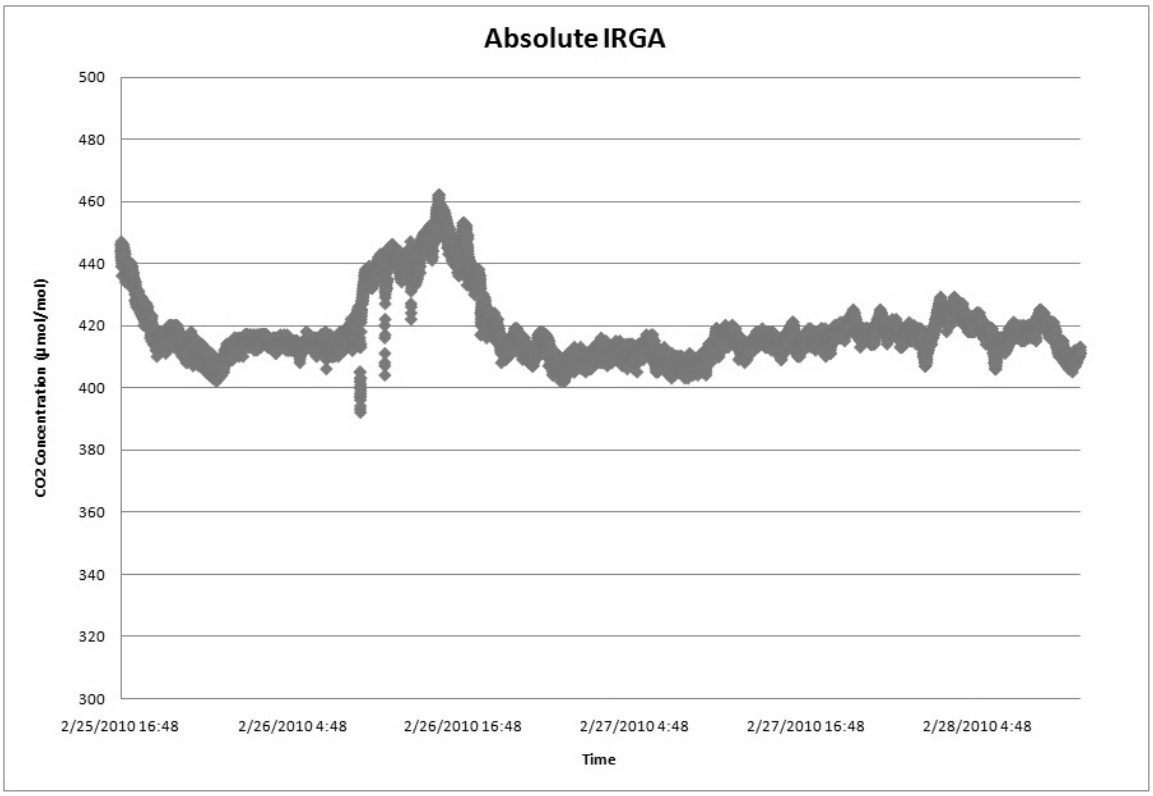

Figure 4-5 Steady absolute concentration

Table 4-2
Absolute IRGA statistics

Mean (µmol/mol)	Standard Deviation (µmol/mol)	Variance (µmol/mol)	95% Confidence Interval (µmol/mol)
419.7009	12.0424	145.0190	0.1943

The mass flow controllers (MFC) maintained flow rates of either pure CO_2 or ambient air mixed with pure CO_2. Figure 4-6 shows the week long results for the pure CO_2 MFC. The setpoint for the pure CO_2 MFC was changed from 0.6slm to 1.0slm and then back to 0.6slm.

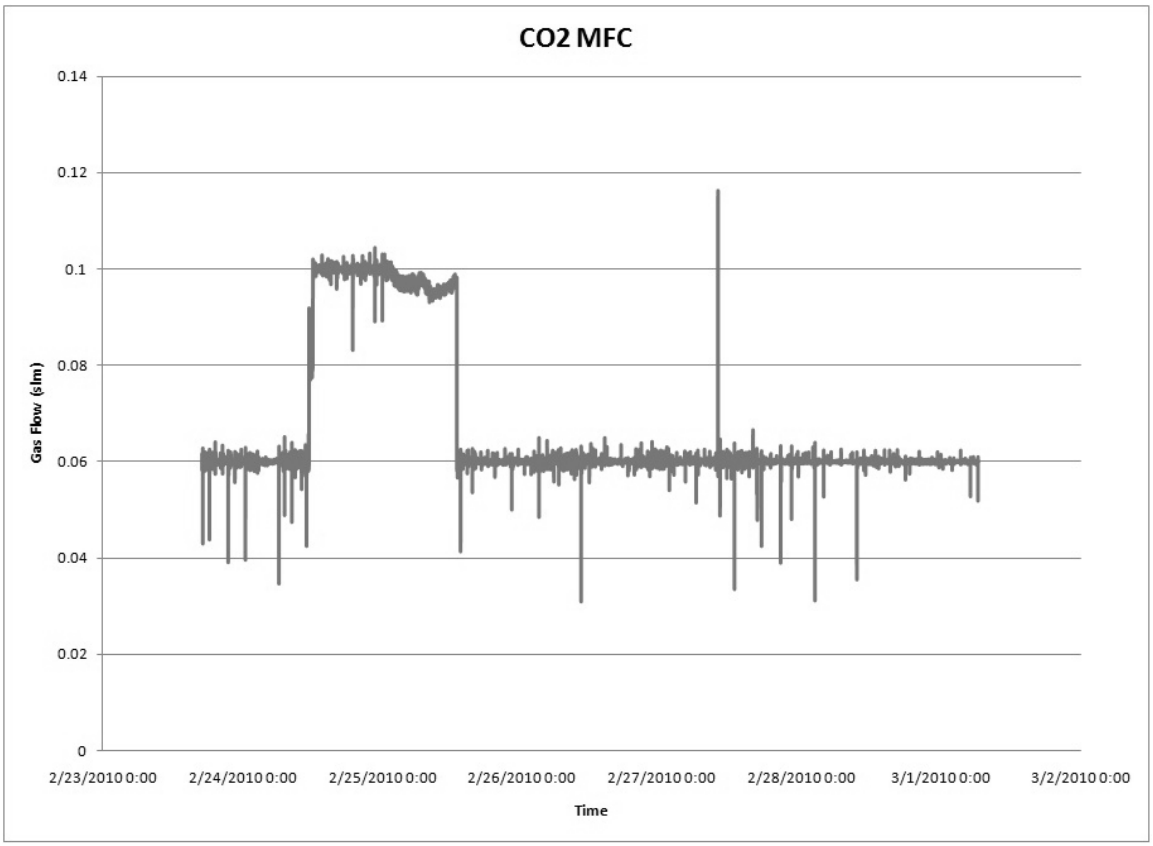

Figure 4-6 CO_2 MFC data

To ensure that changes in pure CO_2 added to the mixed air stream do not saturate the cuvette chamber, changes in MFC setpoints are time limited. The setpoint time delay is set by dividing the time delay by the data collection period. In this case, the time delay was one minute with a 15 second setpoint time delay. Figure 4-7 shows this stepped response. Table 4-3 shows an average value of 0.6slm and a standard deviation of 0.9E-3slm during the 0.6slm steady state period.

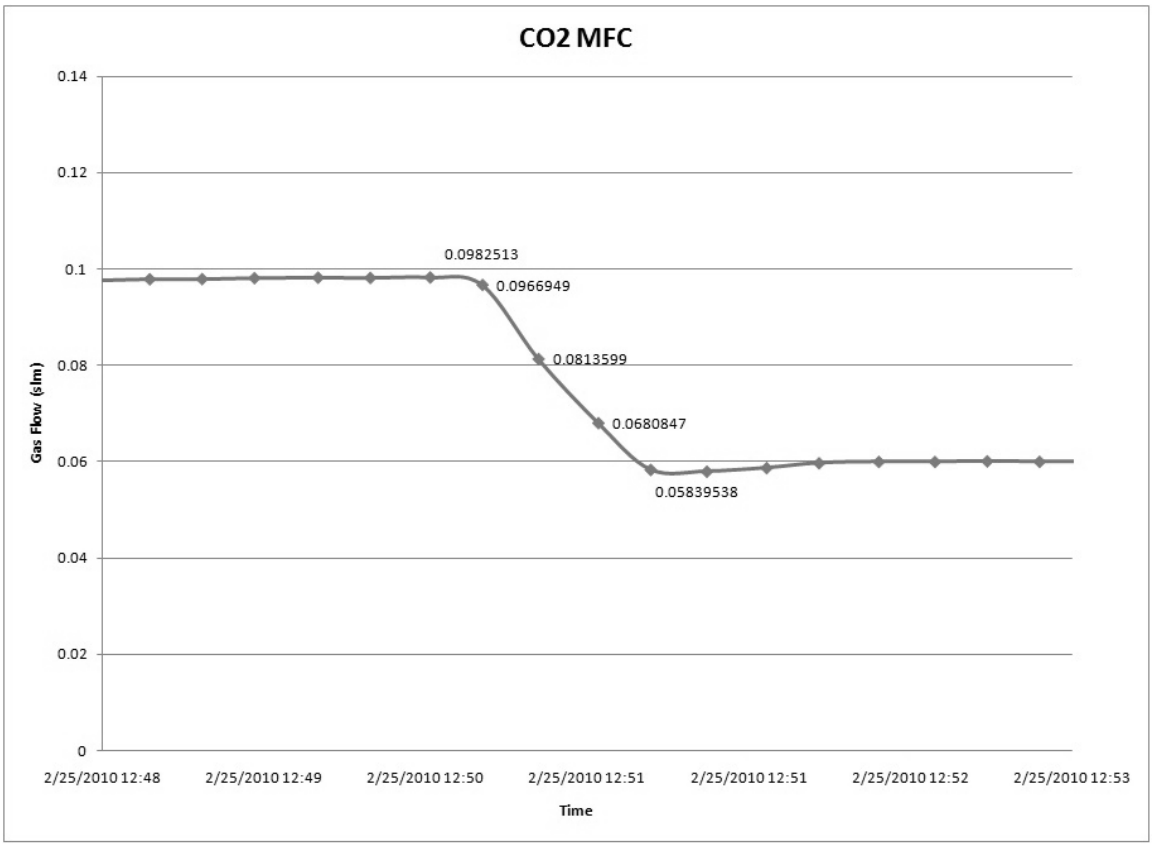

Figure 4-7 CO_2 MFC setpoint delay

Table 4-3
CO_2 MFC statistics

Mean (slm)	Standard Deviation (slm)	Variance (slm)	95% Confidence Interval (slm)
0.6000	0.0009	<0.0001	<0.0001

The mixed air MFC data is shown in Figure 4-8. A setpoint delay is also implemented; however, the change from 15.0slm to 28slm occurred during the system setup period. Therefore, the stepped setpoints are not uniformly spaced, but the desired setpoint is achieved within four steps, as shown in Figure 4-9. Table 4-4 shows an average value of 28.0slm and a standard deviation of 23.0E-3slm during the 28.0slm steady state period.

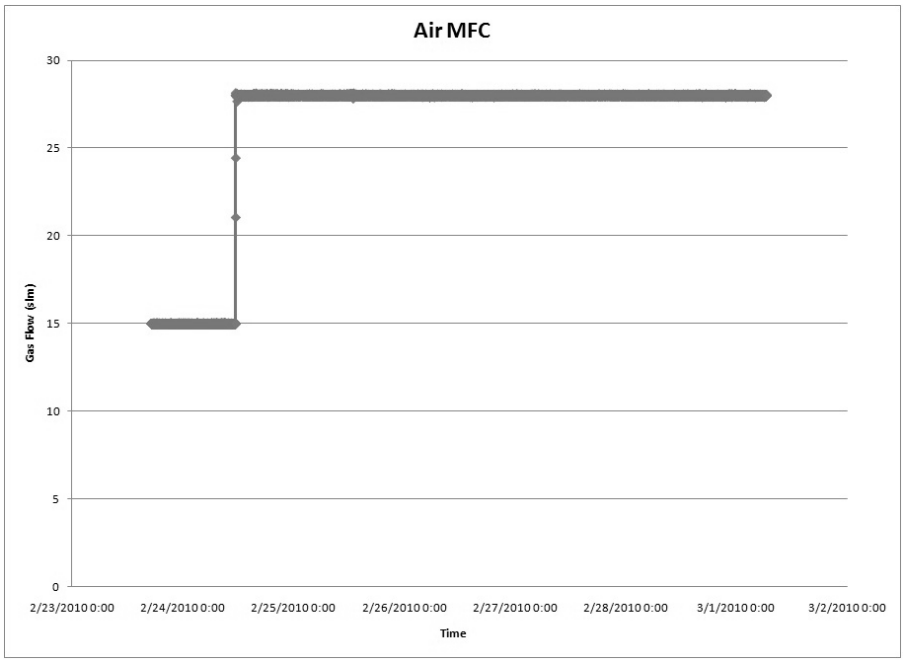

Figure 4-8 Air MFC data

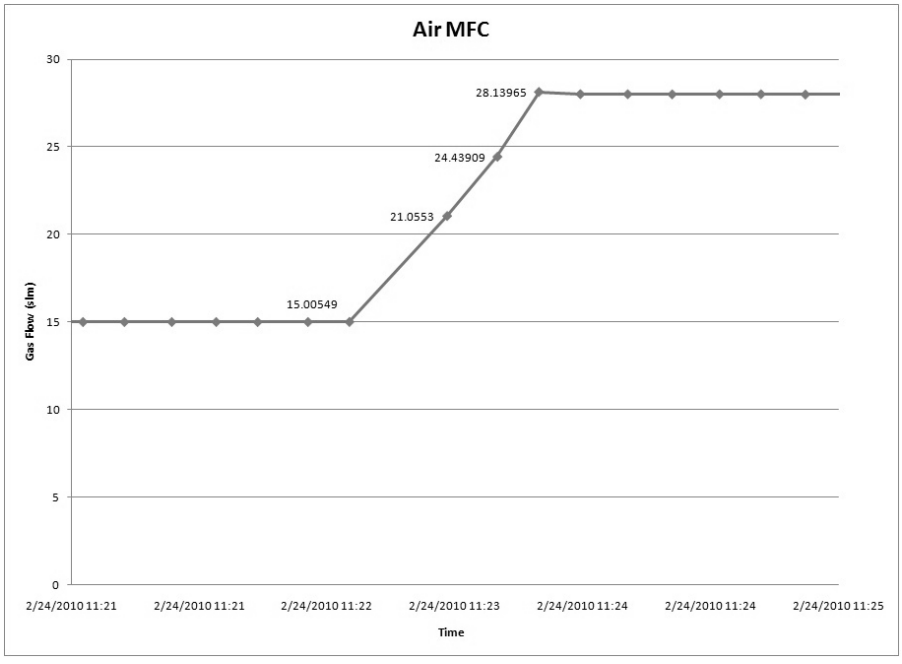

Figure 4-9 Air MFC setpoint delay

Table 4-4
Air MFC statistics

Mean (slm)	Standard Deviation (slm)	Variance (slm)	95% Confidence Interval (slm)
27.9986	0.0230	0.0005	0.0004

The camera and two axis stage work in concert to allow scientists to view the crop of plants within the cuvette. The stage has enough freedom of movement across both axes to allow the camera to take pictures from nearly any angle. Examples are shown in Figures 4-10 and 4-11 using the freeware program AvaCam. The figures show the bottom of the cuvette with the hydroponics chamber underneath. The plants would grow through the holes shown. Also visible are the bottom of the "light-sicles" or their reflections.

Figure 4-10 First webcam picture

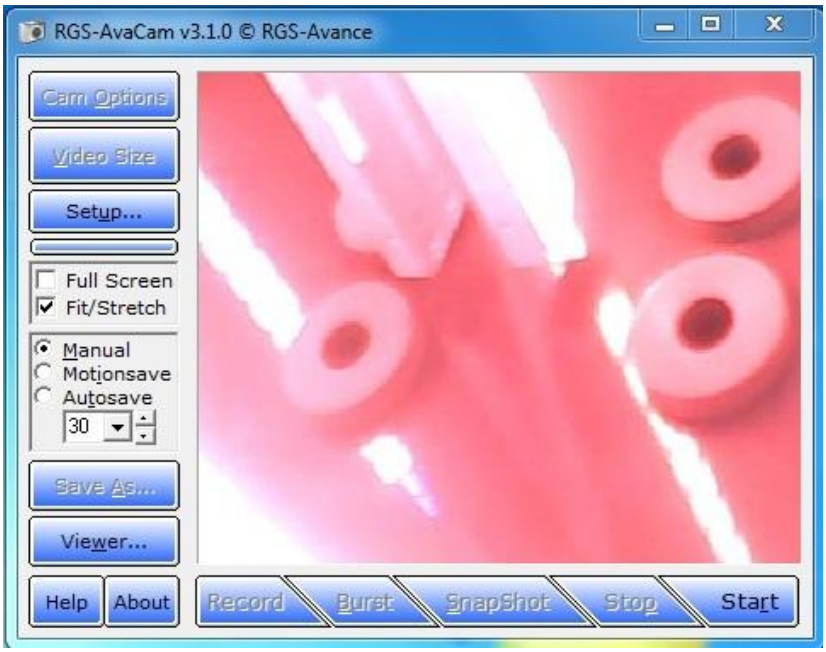

Figure 4-11 Second webcam picture

While the week-long test was being performed, the “light-sicles” were periodically cycled on and off. This had an effect on the temperature and relative humidity within the cuvette chamber. This can be seen in Figures 4-12 and 4-13. The two series of data in each figure represent either temperature or humidity for probes one and two. The two probes performed nearly identically with a temperature correlation of 0.9988 and a humidity correlation of 0.9985, shown in Table 4-5. Comparing Figures 4-12 and 4-13 one can also see the inverse relationship between temperature and humidity within the cuvette.

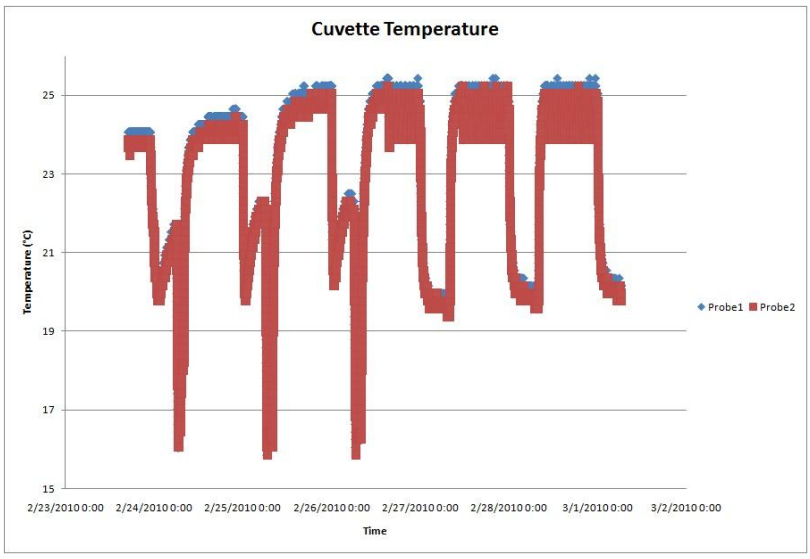

Figure 4-12 Cuvette temperature

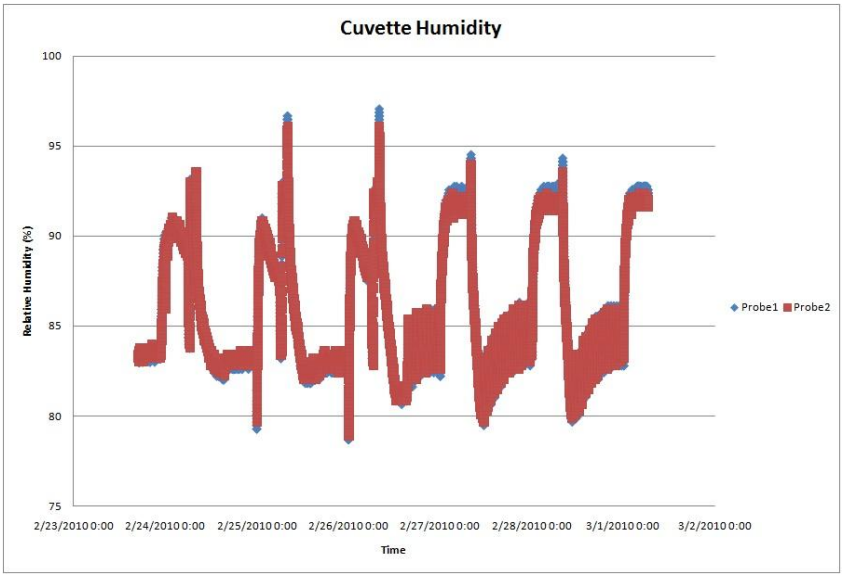

Figure 4-13 Cuvette humidity

Table 4-5
Temperature and relative humidity correlation

Temperature (°C)	Relative Humidity (%)
0.9988	0.9985

4.2. Findings with a feedback loop

After the addition of the PI feedback loop, the results for the absolute gas analyzer and the pure CO_2 mass flow controller were changed, as seen in Figures 4-14 and 4-15. For approximately two-thirds of the experiment the CO_2 concentration setpoint was set to 1000µmol/mol, and it was then set to 600µmol/mol for the remainder. Additionally, Figures 4-16 and 4-17 show the system's response to the change in setpoint, reaching steady state within approximately 5 minutes. Table 4-6 shows the average ambient CO_2 concentration at 999.8µmol/mol with a standard deviation of 3.7µmol/mol during the 1000µmol/mol setpoint. Table 4-7 shows the average value of 0.14slm with a standard deviation of 6.6E-3slm during the 1000µmol/mol setpoint.

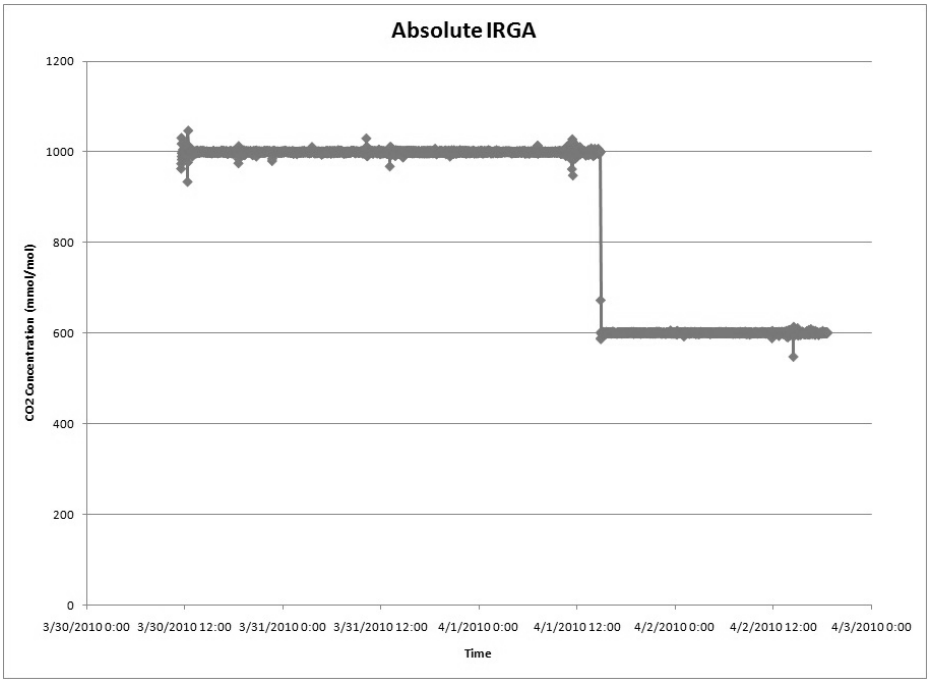

Figure 4-14 Absolute IRGA with feedback

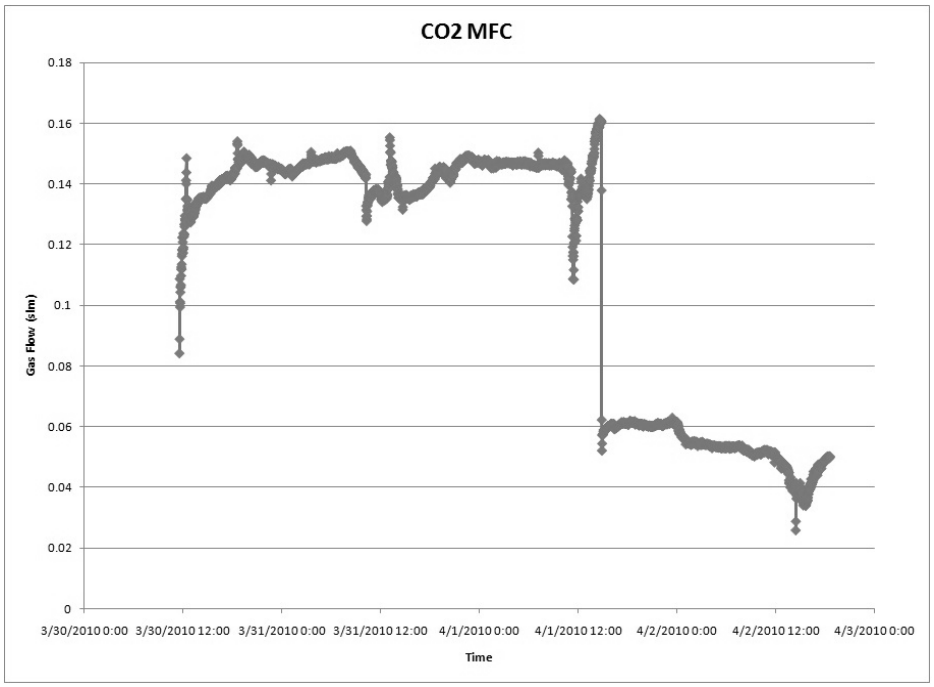

Figure 4-15 CO_2 MFC with feedback

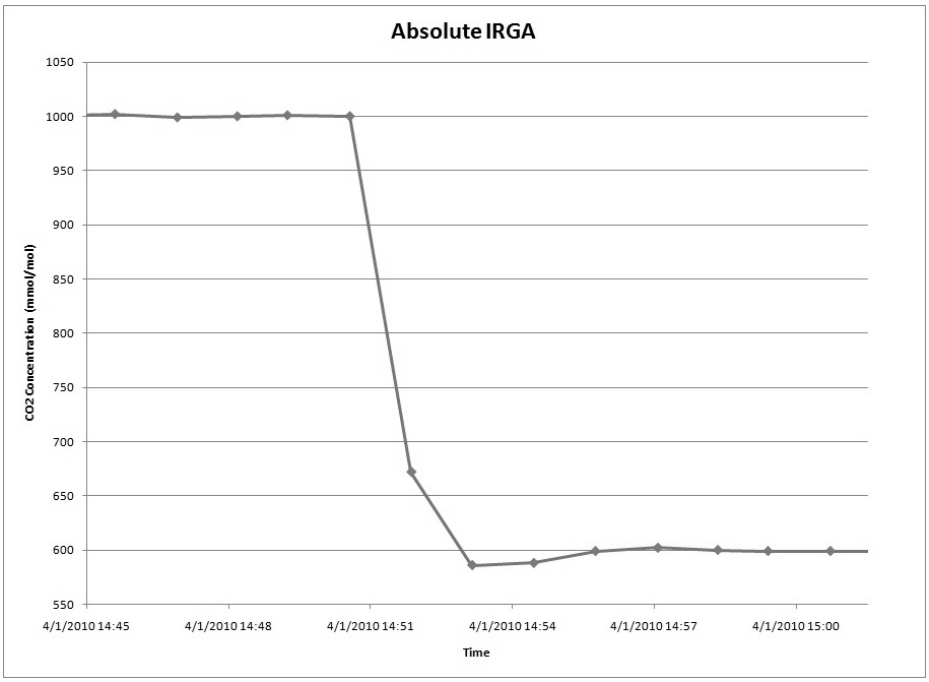

Figure 4-16 Zoomed in Absolute IRGA with feedback

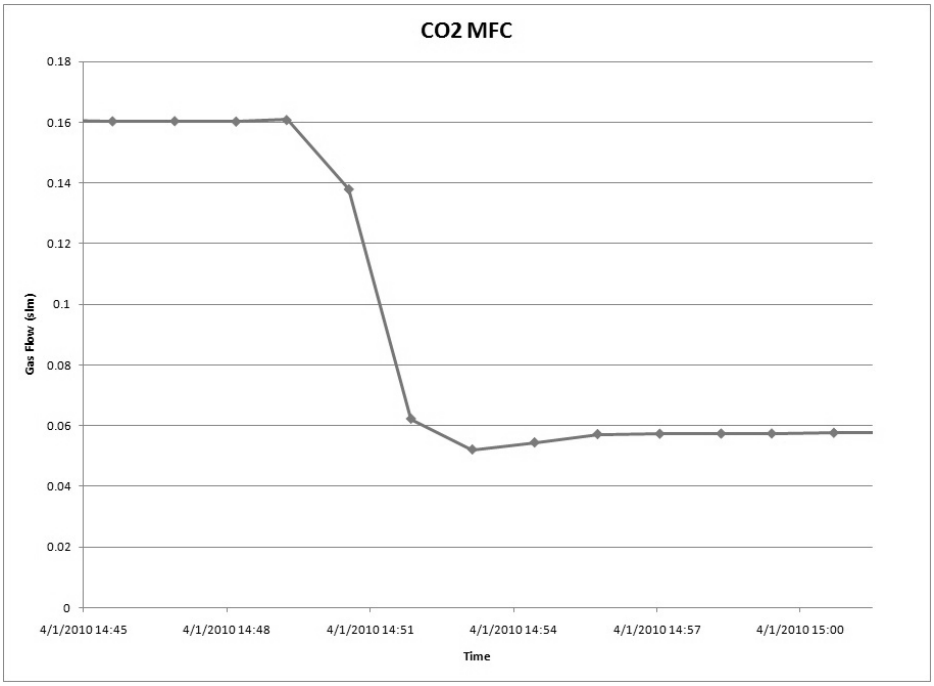

Figure 4-17 Zoomed in CO_2 MFC with feedback

Table 4-6
Absolute IRGA statistics during 1000µmol/mol setpoint

Mean (µmol/mol)	Standard Deviation (µmol/mol)	Variance (µmol/mol)	95% Confidence Interval (µmol/mol)
999.8468	3.7042	13.7208	0.1453

Table 4-7
CO_2 MFC statistics during 1000µmol/mol setpoint

Mean (slm)	Standard Deviation (slm)	Variance (slm)	95% Confidence Interval (slm)
0.1430	0.0066	<0.0001	0.0003

CHAPTER 5. CONCLUSIONS, DISCUSSION, AND RECOMMENDATIONS

This thesis has discussed why it is important to study the effects of environmental variables on crops of plants, how data has been previously collected and used for these kinds of studies, and how new techniques have been developed to acquire data and control these environments. Along the way, several lessons have been learned and future contributions may be possible.

Future space exploration cannot in perpetuity rely on Earth for resupplying astronauts' needs. Eventually, the missions' lengths will make this unfeasible. In order to maximize the chances of future mission success, scientists must study means of allowing astronauts to become self-reliant in space or on other planets. This is accomplished by studying the net photosynthetic rate of plants on Earth, using cuvettes, to determine which plants under what conditions would best supply astronauts' needs. In past studies, researchers carried out experiments by tediously recording measurements and manually making adjustments to growing environments. This cuvette data acquisition system has largely replaced this work with an automated system, using common lab equipment and a personal computer.

In order to automate the task of collecting data from the pre-existing lab equipment, a common form of data conveyance needed to be employed. Amongst the "smart" equipment, the only common protocol was RS232 serial communication. "Dumb" devices, such as the temperature and relative humidity probes, were converted from analog signals to RS232, so all devices communicated using the same network. Using a more modern communication protocol, such as USB, may have improved data communication speeds and improved the upgradeability of the system, but converting the pre-existing

equipments' data protocol would have added another layer of complexity to the hardware network and software. Additionally, converting between USB and RS232 would still limit communication speeds to that of RS232 speeds. Also, as physiological changes within the crop of plants don't occur at high speeds, this system doesn't warrant that volume of data collection.

In order to provide the greatest amount of flexibility to scientists for data collection, a custom program was written in the Visual C# programming language and used the Microsoft .NET framework, available on all Windows OS computers. The Excel 2003 spreadsheet format was chosen as a receptacle for the collected data as it is a common file format capable of being statistically analyzed in Excel or imported to common statistical packages, such as SAS. It was also chosen as an alternative to flat file formats, such as comma-separated values (.csv) or text (.txt) files, in order to store data from multiple sources within one file. Using a flat file would have required multiple flat files or different fixed locations within a single file. In addition to the custom program, a free webcam program was used to periodically capture images, and a free webserver program was used to present data via the Internet. These new hardware and software techniques have allowed for a more automated and flexible data acquisition system.

Creating this data acquisition system presented several challenges. Two of these were software challenges: programmatic interaction with Excel and multithreading.

Programmatically opening, interacting, and especially closing Excel can be a very tricky task as noted by Read (2007) and Wills (2008). As Read points out, interaction with Excel is carried out using Microsoft COM, a precursor to the Microsoft .NET framework. As such, the interaction is not native to the .NET framework, introducing a host of potential unseen conflicts. While not implemented in this thesis, Read is likely correct in that using Visual Basic .NET would be a better alternative to using Visual C#. This could simply be a separate class written in VB.NET or a separate dynamically linked library. However, even

when VB.NET is used, there are many programming traps to beware of. Wills explains many of the common mistakes that can prevent COM objects, like Excel, from closing correctly.

Another software challenge in this project was the use of multithreading. While multithreading is a common way for tasks to share processor time, great care must be taken to prevent unexpected results. Wegerson (2007) provides an excellent example for initiating worker threads separate of the GUI thread and updating the GUI thread from those worker threads. This is the method employed in this thesis, but is a somewhat complex process. Instead, as Wergerson points out, a background worker component would likely make this a more manageable task to complete.

Using newer techniques for data acquisition systems can help reduce the cost, time, and overhead involved in gathering data from cuvettes and other systems. This thesis has demonstrated how using modern software, a personal computer, and common lab equipment can create an automated data acquisition system used to gather and analyze data that may one day assist in space exploration and colonization.

LIST OF REFERENCES

LIST OF REFERENCES

Akers, C. P., Akers, S. W., & Mitchell, C. A. (1985). The Minitron System for Growth of Small Plants under Controlled Environmnet Conditions. *Journal of the American Society for Horticultural Science*, *110*(3), 353-357.

Bourget, C. M. (2008). An Introduction to light-emitting diodes [Electronic version]. *HortScience*, *43*(7), 1944-1946.

Bowman, G.E. & Hand, D.W. (1968). A Cuvette Glasshouse for Specifying the Environmental Requirements of Glasshouse Crops*. Acta Horticulturae, 7*, 49-60.

Burr-Brown (1994). PRINCIPLES OF DATA ACQUISITION AND CONVERSION. Retrieved January 12, 2010, http://focus.ti.com/lit/an/sbaa051/sbaa051.pdf

Castañeda-Miranda, R., Ventura-Ramos, E., del Rocío Peniche, R., & Herrera-Ruiz, G. (2006). Fuzzy Greenhouse Climate Control System based on a Field Programmable Gate Array. *Biosystems Engineering*, *94*(2), 165-177.

Deitel, H. M., & Deitel, P. J. (2006). *C# for Programmers* (2nd ed., p. 601). Upper Saddle River, NJ: Pearson Education, Inc.

Drysdale, Alan. (2001) *Life Support Trade Studies Involving Plants*. Retrieved April 4, 2009, http://www.sae.org/technical/papers/ 2001-01-2362

Energizer Battery (2001, November 6). *NICKEL-METAL HYDRIDE*. Retrieved April 11, 2009, http://data.energizer.com/PDFs/nickelmetalhydride_appman.pdf

Ferl, R., Wheeler, R., Levine, H., & Paul, A. (2002). Plants in space. *Current Opinion in Plant Biology*, *5*, 258-263.

Futron Corporation. (2002, September). Space Transportation Costs: Trends in Price Per Pound to Orbit 1990-2000.

Jones, Harry. (2006). *Comparison of Bioregenerative and Physical/Chemical Life Support Systems*. Retrieved April 4, 2009, http://www.sae.org/technical/papers/2006-01-2082

Hoagland, D. R. & Arnon, D. I. (1938, Revised by Arnon 1950). *The Water-Culture Method for Growing Plants without Soil (Circ. 347)*. University of California Berkeley, California Agricultural Experiment Station.

Kalaitzakis, K., Koutroulis, E., & Vlachos, V. (2003). Development of a data acquisition system for remote monitoring of renewable energy systems. *Measurement*, *34*, 75-83.

Kliss, M., Heyenga, G., Hoehn, A., & Stodieck, L. (2000). *Toward the Development of a "Salad Machine"*. Retrieved April 7, 2009, http://www.sae.org/technical/papers/2000-01-2476

Knight, S. L., Akers, C. P., Akers, S. W., & Mitchell, C. A. (1988). Minitron II System for Precise Control of the Plant Growth Environments. *Photosynthetica*, *22*(1), 90-98.

Koutroulis, E., & Kalaitzakis, K. (2003). Development of an integrated data-acquisition system for renewable energy sources systems monitoring. *Renewable Energy*, *28*, 139-152.

Levri, J. A., Fisher, J. W., Jones, H. W., Drysdale, A. E., Ewert, A. K., Hanford, A. J., et al. (2003). *Advanced Life Support Equivalent System Mass Guidelines Document*. Retrieved April 7, 2009, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20040021355_20040 08092.pdf

Long, S. P., Farage, P. K., & Garcia, R. L. (1996). Measurement of leaf and canopy photosynthetic C02 exchange in the field. *Journal of Experimental Botany*, *47*(304), 1629-1642.

Maxim (2001, March 1). *APPLICATION NOTE 1080 Understanding SAR ADCs*. Retrieved January 13, 2010, http://pdfserv.maxim-ic.com/en/an/AN1080.pdf

Mendoza-Jasso, J., Ornelas-Vargas, G., Castañeda-Miran, R., Ventura-Ramos, E., Zepeda-Garrido, E., & Herrera-Ruiz, G. (2005). FPGA-based real-time remote monitoring system. *Computers and Electronics in Agriculture, 49*(2), 272-285. from ScienceDirect.

Mitchell, C. (1992). Measurement of Photosynthetic Gas Exchange in Controlled Environments. *HortScience, 27*(7), 764-767.

Mitchell, C. (1994). Bioregenerative life-support systems. *Am J Clin Nutr*, *60*(5), 820S-824S.

Morais, R., Matos, S., Fernandes, M., Valente, A., Soares, A., Ferreira, P., et al. (2008). Sun, wind and water flow as energy supply for small stationary data acquisition platforms. *Computers and Electronics in Agriculture, 64*, 120-132. from ScienceDirect.

Net Applications. (2009, November). *Operating System Market Share*. Retrieved December 31, 2009, http://marketshare.hitslink.com/operating-system-market-share.aspx?qprid=8&qpmr=100&qpdt=1&qpct=3&qptimeframe=M&qpsp=130&qpnp=1

Read, D. (2007, March 7). *Lessons Learned Automating Excel from .NET*. Retrieved July 15, 208, from http://www.developerdotstar.com/community/automate_excel_dotnet

Sierra Instruments. (2008). *Precise Gas Flow Control: The Sierra Way* [Online video]. Sierra Instruments. Retrieved January 28, 2010, from http://sierratechsupport.com/video/flow_control.html

Smith, A. L. (1997). Photosynthesis. In *Oxford Dictionary of Biochemistry and Molecular Biology* (p. 508).

Teledyne Hastings Instruments (2005, August). *HFM-D-300 / HFC-D-302 FLOWMETER/CONTROLLER*. Retrieved January 28, 2010, from http://www.teledyne-hi.com/Manual/Flow/157-112009_HFM-D-300-HFC-D-302.pdf

Timlin, D., Lutfor Rahman, S. M., Baker, J., Reddy, V. R., Fleisher, D., & Quebedeaux, B. (2006). Whole Plant Photosynthesis, Development, and Carbon Partitioning in Potato as a Function of Temperature. *Agronomy Journal*, *98*, 1195-1203.

Tocci, R. J., Widmer, N. S., & Moss, G. L. (2004). *Digital Systems: Principles and Applications* (9th ed., pp. 748-757). Upper Saddle River, NJ: Pearson Education Inc.

vanIersel, M. W., & Bugbee, B. (2000). A Multiple Chamber, Semicontinuous, Crop Carbon Dioxide Exchange System: Design, Calibration, and Data Interpretation. *Journal of the American Society for Horticultural Science,* 25(1), 86-92.

Vellidis, G., Tucker, M., Perry, C., Kvien, C., & Bednarz, C. (2008). A real-time wireless smart sensor array for scheduling irrigation. *Computers and Electronics in Agriculture, 61*, 44-50. from ScienceDirect

Venere, E. (2002, March 12). Purdue to help NASA create life-supporting ecosystem in space. *Purdue News Service*. Retrieved April 11, 2009, http://news.uns.purdue.edu/UNS/html4ever/020312.Mitchell.NASAcenter.html

Wegerson, W. (2007, September 1). *Safely Update .Net Winform from Threads or Timer Ticks*. Retrieved February 11, 2010, from http://www.omegacoder.com/?p=117

Wills, M. (2008, May 15). *COM Interop*. Retrieved July 15, 2008, from http://ausdotnet.wordpress.com/2008/05/15/com-interop-welcome-to-pain

Wünsche, J., & Palmer, J. (1997). Portable Through-flow Cuvette System for Measuring Whole-canopy Gas Exchange of Apple Trees in the Field. *HortScience*, *32*(4).

APPENDIX

APPENDIX .

Main.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Diagnostics;
using System.IO;
using System.Reflection;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading;
using System.Windows.Forms;
using Excel = Microsoft.Office.Interop.Excel;
using ZedGraph;


namespace Cuvette
{
    public partial class mainForm : Form
    {
        #region declarations
        static Excel.Application excelApp;
        static Excel.Workbooks excelBooks;
        static Excel.Workbook excelBook;
        static Excel.Sheets excelSheets;
        static Excel.Worksheet excelSheet;
        static Excel.Range excelRange;

        const byte STX = 0x02;
        const byte EOT = 0x04;
        const byte ESC = 0x1b;
        const byte A = 0x41;
        const byte B = 0x42;
        const byte C = 0x43;
        const byte D = 0x44;
        const byte E = 0x45;

        int diffTickStart = 0;
        double diffGraphValue = 0;
        int absTickStart = 0;
        double absGraphValue = 0;
```

```
        byte[] closePort = { ESC, STX, EOT };

        bool firstThreadCreated = false;
        bool extraProcessAtStartup = false;
        int excelCurrentRow = 1;
        string excelFilename = null;
        string excelCurrentDirectory = Application.StartupPath;

        decimal co2MFCNewValue = 0;
        decimal co2AbsSetValue = 0;
        decimal co2AbsNewValue = 0;
        decimal co2AbsErrorValue = 0;
        decimal co2ProportionalConstant = 0;
        decimal co2IntegralConstant = 0;
        decimal co2Integral = 0;
        decimal co2Sum = 0;
        decimal airSpacing = 0;
        decimal airSum = 0;
        int airIterations = 1;
        int airNumIterations = 0;

        DateTime nextSaveTime;
        DateTime lastSaveTime;
        Thread readThread;
        DateTime lastBlowerTimeUpdate;
        #endregion

        public mainForm()
        {
            InitializeComponent();

            //subscribe to thread event
            btnMFC += new buttonMFC(MFCbtn_readTick);
            readDiff += new readTimer(DiffIRGA_readTick);
            readAbs += new readTimer(AbsIRGA_readTick);
            readCO2 += new readTimer(CO2MFV_readTick);
            readAir += new readTimer(AirMFV_readTick);
            readTRH += new readTimer(ReadTempRH_readTick);
            disTRH += new displayTRH(DisplayTempRH_readTick);
        }
        #region Excel
            private void newFileToolStripMenuItem_Click(object sender,
EventArgs e)
            {
                try
                {
                    saveAsExcel();
                }
                catch (Exception theException)
                {
                    String errorMessage;
                    errorMessage = "Error: ";
                    errorMessage = String.Concat(errorMessage,
theException.Message);
```

```
                    errorMessage = String.Concat(errorMessage, " Line:
");
                    errorMessage = String.Concat(errorMessage,
theException.Source);

                    MessageBox.Show(errorMessage, "Error");
                }
                //if user didn't hit cancel
                //hide file so user doesn't open file in use
                //enable/disable components
                if (excelFilename != null)
                {
                    File.SetAttributes(excelFilename,
FileAttributes.Hidden);
                    newFileToolStripMenuItem.Enabled = false;
                    appendExistingFileToolStripMenuItem.Enabled =
false;
                    btnStart.Enabled = true;
                    btnStart.Focus();
                }
            }
            private void
appendExistingFileToolStripMenuItem_Click(object sender, EventArgs e)
            {
                DialogResult openResult = ofdAppend.ShowDialog();

                //exit if user cancels
                if (openResult == DialogResult.Cancel)
                    return;

                excelFilename = ofdAppend.FileName;

                if (excelFilename == "" || excelFilename == null)
                {
                    MessageBox.Show("File does not exist!", "Error",
                        MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
                else
                {
                    excelFilename = ofdAppend.FileName;
                    try
                    {
                        //see if file is already open
                        using (FileStream fs =
File.OpenWrite(excelFilename))
                        {
                        }
                        File.SetAttributes(excelFilename,
FileAttributes.Hidden);
                    }
                    catch
                    {
                        //file is open by another app
                        MessageBox.Show("File already in use. Please
close the file and try again.\r\n\r\n" +
```

```
                            "Note: If you don't have Excel open check
the Processes tab \r\n" +
                            "in Windows Task Manager and end any
processes called EXCEL.EXE",
                            "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                        return;
                    }
                    try
                    {
                        /***************************************
                         * if the file to append to is too long
                         * the program will look like it crashed
                         * so this is added to to reassure user
                         * and hopefully prevent user from closing
                         * the program
                         * *************************************/
                        Form pleaseWait = new Form();
                        pleaseWait.Size = new Size(200, 150);
                        pleaseWait.Text = " ";
                        pleaseWait.ControlBox = false;
                        pleaseWait.StartPosition =
FormStartPosition.CenterScreen;
                        pleaseWait.FormBorderStyle =
FormBorderStyle.FixedToolWindow;
                        pleaseWait.ShowInTaskbar = false;
                        System.Windows.Forms.Label lblWait = new
System.Windows.Forms.Label();
                        lblWait.Parent = pleaseWait;
                        lblWait.Location = new Point(65, 25);
                        lblWait.Size = new Size(70, 45);
                        lblWait.Text = "   Loading \r\n\r\nPlease
wait...";
                        ProgressBar pgrWait = new ProgressBar();
                        pgrWait.Parent = pleaseWait;
                        pgrWait.Location = new Point(25, 90);
                        pgrWait.Size = new Size(150, 20);
                        pgrWait.Style = ProgressBarStyle.Marquee;
                        pgrWait.Step = 10;
                        pgrWait.MarqueeAnimationSpeed = 20;
                        pleaseWait.Show();


                        //minimize main form
                        this.WindowState = FormWindowState.Minimized;

                        //put focus on please wait form
                        //and repaint form
                        pleaseWait.Activate();
                        Application.DoEvents();

                        //open file
                        excelBook =
excelApp.Workbooks.Open(excelFilename, Missing.Value, Missing.Value,
```

```
                            Missing.Value, Missing.Value,
Missing.Value, Missing.Value, Missing.Value,
                            Missing.Value, Missing.Value,
Missing.Value, Missing.Value, Missing.Value,
                            Missing.Value, Missing.Value);
                        excelSheets = excelBook.Worksheets;
                        if (excelSheets.Count != 5)
                        {
                            if (excelSheets.Count != 4)
                            {
                                excelSheets.Add(Missing.Value,
excelSheets.get_Item(3), Missing.Value, Missing.Value);
                            }
                            excelSheets.Add(Missing.Value,
excelSheets.get_Item(4), Missing.Value, Missing.Value);
                        }
                        excelSheet =
(Excel.Worksheet)excelSheets.get_Item(1);

                        //find first empty cell in column a and start
writing from there
                        excelRange = excelSheet.get_Range("A" +
excelCurrentRow, Missing.Value);
                        while (excelRange.Value2 != null)
                        {
                            Application.DoEvents();
                            excelCurrentRow++;
                            excelRange = excelSheet.get_Range("A" +
excelCurrentRow, Missing.Value);
                        }

                        //close please wait form
                        //and restore main form
                        pleaseWait.Close();
                        this.WindowState = FormWindowState.Normal;

                        //Return control of Excel to the user.
                        excelApp.UserControl = true;
                    }
                    catch (Exception theException)
                    {
                        String errorMessage;
                        errorMessage = "Error: ";
                        errorMessage = String.Concat(errorMessage,
theException.Message);
                        errorMessage = String.Concat(errorMessage, "
Line: ");
                        errorMessage = String.Concat(errorMessage,
theException.Source);

                        MessageBox.Show(errorMessage, "Error");
                    }
                }
                newFileToolStripMenuItem.Enabled = false;
                appendExistingFileToolStripMenuItem.Enabled = false;
```

```
                btnStart.Enabled = true;
                btnStart.Focus();
            }
            private void saveAsExcel()
            {
                DialogResult saveResult = sfdExit.ShowDialog();

                //exit if user cancels
                if (saveResult == DialogResult.Cancel)
                    return;

                excelFilename = sfdExit.FileName;
                excelBook = excelBooks.Add(Missing.Value);
                try
                {
                    excelBook.SaveAs(sfdExit.FileName,
Excel.XlFileFormat.xlWorkbookNormal, Missing.Value,
                        Missing.Value, Missing.Value, Missing.Value,
Excel.XlSaveAsAccessMode.xlNoChange,
                        Missing.Value, Missing.Value, Missing.Value,
Missing.Value, Missing.Value);
                    excelSheets = excelBook.Worksheets;
                    excelSheets.Add(Missing.Value,
excelSheets.get_Item(3), Missing.Value, Missing.Value);
                    excelSheets.Add(Missing.Value,
excelSheets.get_Item(4), Missing.Value, Missing.Value);
                    excelSheet =
(Excel.Worksheet)excelSheets.get_Item(1);
                }
                catch (Exception theException)
                {
                    String errorMessage;
                    errorMessage = "Error: ";
                    errorMessage = String.Concat(errorMessage,
theException.Message);
                    errorMessage = String.Concat(errorMessage, " Line:
");
                    errorMessage = String.Concat(errorMessage,
theException.Source);

                    MessageBox.Show(errorMessage, "Error");
                }
            }
            private void saveExcel()
            {
                //make sure there is something to save to
                if (excelBook == null)
                {
                    MessageBox.Show("Please start or open a file
first.", "Error",
                        MessageBoxButtons.OK, MessageBoxIcon.Error);
                    return;
                }
                else
                {
```

```
                    try
                    {
                        excelBook.Save();
                    }
                    catch (Exception theException)
                    {
                        String errorMessage;
                        errorMessage = "Error: ";
                        errorMessage = String.Concat(errorMessage,
theException.Message);
                        errorMessage = String.Concat(errorMessage, "
Line: ");
                        errorMessage = String.Concat(errorMessage,
theException.Source);

                        MessageBox.Show(errorMessage, "Error");
                    }
                }
            }
        #endregion
        #region buttons
        private void btnStart_Click(object sender, EventArgs e)
        {
            lastSaveTime = DateTime.Now;
            nextSaveTime =
DateTime.Now.AddMinutes((double)nudSaveDataDelay.Value);
            // Save the beginning time for reference
            diffTickStart = Environment.TickCount;
            absTickStart = Environment.TickCount;
            //don't let user stop thread if
            //one hasn't been created
            if (firstThreadCreated == true)
            {
                btnStop.Enabled = true;
                btnStop.Focus();
            }
            btnStart.Enabled = false;
            if (sp1.IsOpen == false)
                sp1.Open();
            tmrRead.Enabled = true;
        }

        private void btnStop_Click(object sender, EventArgs e)
        {
            tmrRead.Enabled = false;
            btnStop.Enabled = false;
            btnStart.Enabled = true;
            btnStart.Focus();
        }

        private void btnClear_Click(object sender, EventArgs e)
        {
            txtAbsIRGA.Clear();
            txtDiffIRGA.Clear();
            txtCO2MFV.Clear();
```

```
            txtAirMFV.Clear();
            txtTempRH.Clear();
        }
        #endregion
        #region periodic data
        private void nudTimeDelay_ValueChanged(object sender, EventArgs
e)
        {
            //change timer interval
            tmrRead.Interval =
Convert.ToInt32(nudDisplayDataDelay.Value) * 1000;
        }

        private void nudSaveDataDelay_ValueChanged(object sender,
EventArgs e)
        {
            nextSaveTime =
lastSaveTime.AddMinutes((double)nudSaveDataDelay.Value);
        }

        #region periodic delegates, events, methods, and timer event
        //create delegate for UI screen updating
        public delegate void buttonMFC();
        public delegate void readTimer(object values);
        public delegate void displayTRH(object values);

        public event buttonMFC btnMFC;
        public event readTimer readDiff;
        public event readTimer readAbs;
        public event readTimer readCO2;
        public event readTimer readAir;
        public event readTimer readTRH;
        public event displayTRH disTRH;

        //method for thread event
        private void MFCbtn_readTick()
        {
            btnCO2MFCStatus.Enabled = true;
            btnAirMFCStatus.Enabled = true;
            btnAirSetFlow.Enabled = true;
            btnAirReadFlow.Enabled = true;
            trbServo0.Enabled = true;
            trbServo1.Enabled = true;
            tmrRead.Enabled = true;
        }

        //method for thread event
        private void DiffIRGA_readTick(object values)
        {
            txtDiffIRGA.Text = txtDiffIRGA.Text + values + "\n";
            txtDiffIRGA.SelectionStart = txtDiffIRGA.Text.Length;
            txtDiffIRGA.ScrollToCaret();
        }

        //method for thread event
```

```
private void AbsIRGA_readTick(object values)
{
    txtAbsIRGA.Text = txtAbsIRGA.Text + values + "\n";
    txtAbsIRGA.SelectionStart = txtAbsIRGA.Text.Length;
    txtAbsIRGA.ScrollToCaret();
}

//method for thread event
private void CO2MFV_readTick(object values)
{
    txtCO2MFV.Text = txtCO2MFV.Text + values + "\r\n";
    txtCO2MFV.SelectionStart = txtCO2MFV.Text.Length;
    txtCO2MFV.ScrollToCaret();
}

//method for thread event
private void AirMFV_readTick(object values)
{
    txtAirMFV.Text = txtAirMFV.Text + values + "\r\n";
    txtAirMFV.SelectionStart = txtAirMFV.Text.Length;
    txtAirMFV.ScrollToCaret();
}

//method for thread event
private void ReadTempRH_readTick(object values)
{
    txtTempRH.Text = txtTempRH.Text + values + "\r\n";
    txtTempRH.SelectionStart = txtTempRH.Text.Length;
    txtTempRH.ScrollToCaret();
}

private void DisplayTempRH_readTick(object values)
{
    double[,] dbValues = (double[,])values;
    for (int i = 0; i < 4; i++ )
    {
        dbValues[0, i] = Math.Round(dbValues[0, i], 2);
    }
    lblProbe1Temp.Text = dbValues[0, 0].ToString();
    lblProbe1RH.Text = dbValues[0, 1].ToString();
    lblProbe2Temp.Text = dbValues[0, 2].ToString();
    lblProbe2RH.Text = dbValues[0, 3].ToString();
}

private void tmrRead_Tick(object sender, EventArgs e)
{
    btnCO2MFCStatus.Enabled = false;
    btnAirMFCStatus.Enabled = false;
    btnAirSetFlow.Enabled = false;
    btnAirReadFlow.Enabled = false;
    trbServo0.Enabled = false;
    trbServo1.Enabled = false;

    tmrRead.Enabled = false;
    //create new thread so main GUI thread doesn't lag
```

```
            readThread = new Thread(new ThreadStart(multiRead));
            readThread.IsBackground = true;
            readThread.Start();
            if (firstThreadCreated == false)
            {
                btnStop.Enabled = true;
                btnStop.Focus();
                firstThreadCreated = true;
            }
        }
        #endregion

        private void multiRead()
        {
            try
            {
                //update blower time
                Cuvette.Properties.Settings.Default.BlowerTime +=
DateTime.Now - lastBlowerTimeUpdate;
                lastBlowerTimeUpdate = DateTime.Now;
                Cuvette.Properties.Settings.Default.Save();
                //lock serial port
                lock (sp1)
                {
                    //read from equipment
                    excelSheet =
(Excel.Worksheet)excelSheets.get_Item(1);
                    readLICOR();
                    updateDiffGraph();
                    excelSheet =
(Excel.Worksheet)excelSheets.get_Item(2);
                    readSBA4();
                    updateAbsGraph();
                    excelSheet =
(Excel.Worksheet)excelSheets.get_Item(3);
                    readCO2MFC();
                    excelSheet =
(Excel.Worksheet)excelSheets.get_Item(4);
                    readAirMFC();
                    excelSheet =
(Excel.Worksheet)excelSheets.get_Item(5);
                    readTempRH();
                    sp1.Close();
                }
                //write back to UI thread
                this.Invoke(this.btnMFC);
                //save to Excel if appropriate
                if (nextSaveTime <= DateTime.Now)
                {
                    saveExcel();
                    lastSaveTime = DateTime.Now;
                    nextSaveTime =
lastSaveTime.AddMinutes((double)nudSaveDataDelay.Value);
                    excelCurrentRow++;
                }
```

```
                //calculate PID
                co2AbsErrorValue = co2AbsSetValue - co2AbsNewValue;
                co2Integral = co2Integral + co2AbsErrorValue;
                if (co2Integral > 10)
                {
                    co2Integral = 10;
                }
                else if (co2Integral < -10)
                {
                    co2Integral = -10;
                }
                co2Sum = co2MFCNewValue + co2ProportionalConstant *
co2AbsErrorValue + co2IntegralConstant * co2Integral;
                if (co2Sum < 0)
                {
                    co2Sum = 0;
                }
                sendCommand(D, co2Sum);
                //check lights
                checkLights();
            }
            catch (Exception theException)
            {
                if (readThread.IsAlive)
                    readThread.Abort();
            }
            finally
            {
                if (sp1.IsOpen)
                    sp1.Close();
            }
        }

        private void readLICOR()
        {
            string LICORValues = null;

                if (sp1.IsOpen == false)
                    sp1.Open();
                //create byte array for writing
                /*{fixed preamble, user defined preamble,
                   switch address, port}*/
                byte[] switchToLICOR = { ESC, STX, B, D };
                //write to serial switch
                sp1.Write(switchToLICOR, 0, switchToLICOR.Length);
                //empty input buffer
                if (sp1.BytesToRead != 0)
                    sp1.DiscardInBuffer();
                //wait for bytes
                while (sp1.BytesToRead == 0) ;
                LICORValues = sp1.ReadLine();
                //first readline reads end of message fragment
                //second readline get full message
                while (sp1.BytesToRead == 0) ;
                LICORValues = sp1.ReadLine();
```

```
                sp1.Write(closePort, 0, closePort.Length);


            //parse string into array
            string[] LICORdelimiteArray = { "  ", "\r", "\n" };
            string[] LICORParsedValues =
LICORValues.Split(LICORdelimiteArray,
StringSplitOptions.RemoveEmptyEntries);
            //convert string[] to double[]
            double[,] LICORDoubleParsedValues = new double[1,
LICORParsedValues.Length];
            int i = 0;
            double LICORresult;
            //if not numeric don't write to excel
            //to remove auto header
            if (double.TryParse(LICORParsedValues[i], out LICORresult))
            {
                //write timestamp
                writeTimeStamp();

                //convert string to double
                foreach (string x in LICORParsedValues)
                {
                    LICORDoubleParsedValues[0, i] =
Convert.ToDouble(x);
                    i++;
                }

                if (LICORParsedValues.Length == 4)
                {
                    //write to excel
                    excelRange = excelSheet.get_Range("B" +
excelCurrentRow, Missing.Value);
                    excelRange = excelRange.get_Resize(1,
LICORDoubleParsedValues.Length);
                    excelRange.Value2 = LICORDoubleParsedValues;
                    diffGraphValue = LICORDoubleParsedValues[0, 0];
                }
            }

            //write back to UI thread
            this.Invoke(this.readDiff, new object[] { LICORValues });
        }

        private void readSBA4()
        {
            string SBA4Values = null;

                if (sp1.IsOpen == false)
                    sp1.Open();
                //create byte array for writing
                /*{fixed preamble, user defined preamble,
                   switch address, port}*/
                byte[] switchToSBA4 = { ESC, STX, B, E };
                //write to serial switch
```

```
                sp1.Write(switchToSBA4, 0, switchToSBA4.Length);
                //empty input buffer
                if (sp1.BytesToRead != 0)
                    sp1.DiscardInBuffer();
                //wait for bytes
                while (sp1.BytesToRead == 0) ;
                SBA4Values = sp1.ReadLine();
                //first readline reads end of message fragment
                //second readline get full message
                while (sp1.BytesToRead == 0) ;
                SBA4Values = sp1.ReadLine();
                sp1.Write(closePort, 0, closePort.Length);


            //parse string into array
            string[] SBA4delimiteArray = { " ", "  ", "   ", "    ", "
", "\r", "\n" };
            string[] SBA4LongParsedValues =
SBA4Values.Split(SBA4delimiteArray,
StringSplitOptions.RemoveEmptyEntries);

            //write timestamp
            writeTimeStamp();
            //remove first element
            ArrayList SBA4ParsedValues = new
ArrayList(SBA4LongParsedValues.Length);
            SBA4ParsedValues.AddRange(SBA4LongParsedValues);
            if ((string)SBA4ParsedValues[0] == "M")
            {
                SBA4ParsedValues.Remove("M");
                //convert string to double
                double[,] SBA4DoubleParsedValues = new double[1,
SBA4ParsedValues.Count];
                int i = 0;
                foreach (string x in SBA4ParsedValues)
                {
                    SBA4DoubleParsedValues[0, i] = Convert.ToDouble(x);
                    i++;
                }

                if (SBA4LongParsedValues.Length == 13)
                {
                    co2AbsNewValue = (decimal)SBA4DoubleParsedValues[0,
2];
                    //write to excel
                    excelRange = excelSheet.get_Range("B" +
excelCurrentRow, Missing.Value);
                    excelRange = excelRange.get_Resize(1,
SBA4DoubleParsedValues.Length);
                    excelRange.Value2 = SBA4DoubleParsedValues;
                    absGraphValue = SBA4DoubleParsedValues[0, 2];
                }
            }

            //write back to UI thread
```

```
            this.Invoke(this.readAbs, new object[] { SBA4Values });
        }

        private void readCO2MFC()
        {
            string CO2MFCValues = null;

                if (sp1.IsOpen == false)
                    sp1.Open();
                //create byte array for writing
                /*{fixed preamble, user defined preamble,
                   switch address, port}*/
                byte[] switchToCO2MFC = { ESC, STX, A, D };
                //write to serial switch
                sp1.Write(switchToCO2MFC, 0, switchToCO2MFC.Length);
                //empty input buffer
                if (sp1.BytesToRead != 0)
                    sp1.DiscardInBuffer();
                //get flow

/*********************************************************
                 * Backslash characters are needed, because when
changing
                 * to a different port two characters are sent through
the
                 * new port.  Not backslashing results in error from
MFVs.
                 *
*********************************************************/
                sp1.Write("\b\b\bf\r");
                //wait for bytes
                while (sp1.BytesToRead == 0) ;
                CO2MFCValues = sp1.ReadTo("\r");
                sp1.Write(closePort, 0, closePort.Length);


            //write timestamp
            writeTimeStamp();
            //convert string to double
            string[] CO2MFCdelimiteArray = { "*" };
            string[] CO2MFCParsedValues =
CO2MFCValues.Split(CO2MFCdelimiteArray,
StringSplitOptions.RemoveEmptyEntries);

            double[,] CO2MFCDoubleParsedValues = new double[1,
CO2MFCParsedValues.Length];
            CO2MFCDoubleParsedValues[0, 0] =
Convert.ToDouble(CO2MFCParsedValues[0]);
            co2MFCNewValue = (decimal)CO2MFCDoubleParsedValues[0, 0];

            if (CO2MFCParsedValues.Length == 1)
            {
                //write to excel
                excelRange = excelSheet.get_Range("B" +
excelCurrentRow, Missing.Value);
```

```
                excelRange = excelRange.get_Resize(1,
CO2MFCParsedValues.Length);
                excelRange.Value2 = CO2MFCDoubleParsedValues;
            }

            //write back to UI thread
            this.Invoke(this.readCO2, new object[] { CO2MFCValues });
        }

        private void readAirMFC()
        {
            string AirMFCValues = null;

                if (sp1.IsOpen == false)
                    sp1.Open();
                //create byte array for writing
                /*{fixed preamble, user defined preamble,
                   switch address, port}*/
                byte[] switchToAirMFC = { ESC, STX, A, E };
                //write to serial switch
                sp1.Write(switchToAirMFC, 0, switchToAirMFC.Length);
                //empty input buffer
                if (sp1.BytesToRead != 0)
                    sp1.DiscardInBuffer();
                //get flow

/*********************************************************
                 * Backslash characters are needed, because when
changing
                 * to a different port two characters are sent through
the
                 * new port.  Not backslashing results in error from
MFVs.
                 *
*********************************************************/
                sp1.Write("\b\b\bf\r");
                //wait for bytes
                while (sp1.BytesToRead == 0) ;
                AirMFCValues = sp1.ReadTo("\r");
                sp1.Write(closePort, 0, closePort.Length);


            //write timestamp
            writeTimeStamp();
            //convert string to double
            string[] AirMFCdelimiteArray = { "*" };
            string[] AirMFCParsedValues =
AirMFCValues.Split(AirMFCdelimiteArray,
StringSplitOptions.RemoveEmptyEntries);

            double[,] AirMFCDoubleParsedValues = new double[1,
AirMFCParsedValues.Length];
            AirMFCDoubleParsedValues[0, 0] =
Convert.ToDouble(AirMFCParsedValues[0]);
```

```
            if (AirMFCParsedValues.Length == 1)
            {
                //write to excel
                excelRange = excelSheet.get_Range("B" +
excelCurrentRow, Missing.Value);
                excelRange = excelRange.get_Resize(1, 1);
                excelRange.Value2 = AirMFCDoubleParsedValues;
            }

            //write back to UI thread
            this.Invoke(this.readAir, new object[] { AirMFCValues });
        }

        private void readTempRH()
        {
            string TempRHValues = null;

                if (sp1.IsOpen == false)
                    sp1.Open();
                //create byte array for writing
                /*{fixed preamble, user defined preamble,
                   switch address, port}*/
                byte[] switchToTempRH = { ESC, STX, A, A };
                //write to serial switch
                sp1.Write(switchToTempRH, 0, switchToTempRH.Length);
                //extra write/read to prepare A->D converter
                sp1.Write("\r");
                TempRHValues = sp1.ReadTo("\r");
                //empty input buffer
                if (sp1.BytesToRead != 0)
                    sp1.DiscardInBuffer();
                //get flow
                sp1.Write("U0\r");
                //wait for bytes
                while (sp1.BytesToRead == 0) ;
                TempRHValues = sp1.ReadTo("\r");
                for (int i = 1; i < 4; i++)
                {
                    sp1.Write("U" + i.ToString() + "\r");
                    //wait for bytes
                    while (sp1.BytesToRead == 0) ;
                    TempRHValues = TempRHValues + " " +
sp1.ReadTo("\r");
                }
                sp1.Write(closePort, 0, closePort.Length);


            //write timestamp
            writeTimeStamp();
            //convert string to double
            string[] TempRHdelimiteArray = { " " };
            string[] TempRHParsedValues =
TempRHValues.Split(TempRHdelimiteArray,
StringSplitOptions.RemoveEmptyEntries);
```

```
            double[,] TempRHDoubleParsedValues = new double[1,
TempRHParsedValues.Length];
            for (int i = 0; i < 4; i++)
            {
                TempRHParsedValues[i] = TempRHParsedValues[i].Remove(0,
2);
                TempRHDoubleParsedValues[0, i] =
Convert.ToDouble(Convert.ToInt32(TempRHParsedValues[i], 16));
            }
            //convert Temp & RH from ADC values to correct values
            /*equations are simplified, for full explanation see
             * analog conversion-1(modified).xls*/
            TempRHDoubleParsedValues[0, 0] =
TempRHDoubleParsedValues[0, 0] * 200 / 1024 - 40;
            TempRHDoubleParsedValues[0, 1] =
TempRHDoubleParsedValues[0, 1] * 200 / 1024;
            TempRHDoubleParsedValues[0, 2] =
TempRHDoubleParsedValues[0, 2] * 200 / 1024 - 40;
            TempRHDoubleParsedValues[0, 3] =
TempRHDoubleParsedValues[0, 3] * 200 / 1024;

            if (TempRHParsedValues.Length == 4)
            {
                //write to excel
                excelRange = excelSheet.get_Range("B" +
excelCurrentRow, Missing.Value);
                excelRange = excelRange.get_Resize(1,
TempRHDoubleParsedValues.Length);
                excelRange.Value2 = TempRHDoubleParsedValues;
            }

            //write back to UI thread
            this.Invoke(this.readTRH, new object[] { TempRHValues });
            this.Invoke(this.disTRH, new object[] {
TempRHDoubleParsedValues });
        }

        private void tabControl1_SelectedIndexChanged(object sender,
EventArgs e)
        {
            switch (tabControl1.SelectedIndex)
            {
                case 0:
                    {
                        txtDiffIRGA.SelectionStart =
txtDiffIRGA.Text.Length;
                        txtDiffIRGA.ScrollToCaret();
                        break;
                    }
                case 1:
                    {
                        txtAbsIRGA.SelectionStart =
txtAbsIRGA.Text.Length;
                        txtAbsIRGA.ScrollToCaret();
                        break;
```

```
                    }
                case 2:
                    {
                        txtCO2MFV.SelectionStart =
txtCO2MFV.Text.Length;
                        txtCO2MFV.ScrollToCaret();
                        break;
                    }
                case 3:
                    {
                        txtAirMFV.SelectionStart =
txtAirMFV.Text.Length;
                        txtAirMFV.ScrollToCaret();
                        break;
                    }
            }
        }

        private void writeTimeStamp()
        {
            string CurrentTime = DateTime.Now.ToString();
            excelRange = excelSheet.get_Range("A" + excelCurrentRow,
Missing.Value);
            excelRange = excelRange.get_Resize(1, 1);
            excelRange.Value2 = CurrentTime;
        }

        private void updateDiffGraph()
        {
            // Make sure that the curvelist has at least one curve
            if (zedGraphControl1.GraphPane.CurveList.Count <= 0)
                return;

            // Get the first CurveItem in the graph
            LineItem curve = zedGraphControl1.GraphPane.CurveList[0] as
LineItem;
            if (curve == null)
                return;

            // Get the PointPairList
            IPointListEdit list = curve.Points as IPointListEdit;
            // If this is null, it means the reference at curve.Points
does not
            // support IPointListEdit, so we won't be able to modify it
            if (list == null)
                return;

            // Time is measured in seconds
            double time = (Environment.TickCount - diffTickStart) /
1000.0;

            // 3 seconds per cycle
            list.Add(time, Convert.ToDouble(diffGraphValue));
```

```
            // Keep the X scale at a rolling 300 second interval, with
one
            // major step between the max X value and the end of the
axis
            Scale xScale = zedGraphControl1.GraphPane.XAxis.Scale;
            if (time > xScale.Max - xScale.MajorStep)
            {
                xScale.Max = time + xScale.MajorStep;
                xScale.Min = xScale.Max - 300.0;
            }

            // Make sure the Y axis is rescaled to accommodate actual
data
            zedGraphControl1.AxisChange();
            // Force a redraw
            zedGraphControl1.Invalidate();
        }

        private void updateAbsGraph()
        {
            // Make sure that the curvelist has at least one curve
            if (zedGraphControl2.GraphPane.CurveList.Count <= 0)
                return;

            // Get the first CurveItem in the graph
            LineItem curve = zedGraphControl2.GraphPane.CurveList[0] as
LineItem;
            if (curve == null)
                return;

            // Get the PointPairList
            IPointListEdit list = curve.Points as IPointListEdit;
            // If this is null, it means the reference at curve.Points
does not
            // support IPointListEdit, so we won't be able to modify it
            if (list == null)
                return;

            // Time is measured in seconds
            double time = (Environment.TickCount - absTickStart) /
1000.0;

            // 3 seconds per cycle
            list.Add(time, Convert.ToDouble(absGraphValue));

            // Keep the X scale at a rolling 300 second interval, with
one
            // major step between the max X value and the end of the
axis
            Scale xScale = zedGraphControl2.GraphPane.XAxis.Scale;
            if (time > xScale.Max - xScale.MajorStep)
            {
                xScale.Max = time + xScale.MajorStep;
                xScale.Min = xScale.Max - 300.0;
            }
```

```
            // Make sure the Y axis is rescaled to accommodate actual
data
            zedGraphControl2.AxisChange();
            // Force a redraw
            zedGraphControl2.Invalidate();
        }

        #endregion
        #region help/about
        private void helpToolStripMenuItem1_Click(object sender,
EventArgs e)
        {
            try
            {
                Directory.SetCurrentDirectory(excelCurrentDirectory);
                System.Diagnostics.Process.Start("Cuvette Help.pdf");
            }
            catch
            {
                MessageBox.Show("Help file not found.\r\nPlease place
help file in the same folder as the program.\r\n",
                    "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }

        private void aboutToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            About About = new About();
            About.ShowDialog();
        }
        private void exitToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            this.Close();
        }
        #endregion
        #region servos
        private void trbServo0_Scroll(object sender, EventArgs e)
        {
            sendCommand(0, trbServo0.Value);
        }

        private void trbServo1_Scroll(object sender, EventArgs e)
        {
            sendCommand(1, trbServo1.Value);
        }
        #endregion
        #region icon
        private void cmsAbout_Click(object sender, EventArgs e)
        {
            About About = new About();
            About.ShowDialog();
```

```
        }

        private void cmsHelp_Click(object sender, EventArgs e)
        {
            try
            {
                Directory.SetCurrentDirectory(excelCurrentDirectory);
                System.Diagnostics.Process.Start("Cuvette Help.pdf");
            }
            catch
            {
                MessageBox.Show("Help file not found.\r\nPlease place
help file in the same folder as the program.\r\n",
                    "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }

        private void cmsExit_Click(object sender, EventArgs e)
        {
                this.Close();
        }
        #endregion
        #region load/closing
        private void Form1_Load(object sender, EventArgs e)
        {
            Process[] pProcess = Process.GetProcessesByName("Excel");
            if (pProcess.Length != 0)
            {
                DialogResult result = MessageBox.Show("This program
uses Microsoft Excel to store data," +
                    " and an Excel process is currently open in the
Task Manager.  " +
                    "\n\r\n\rClick Cancel to close Excel yourself, or
click OK to close it now.",
                    "Warning!", MessageBoxButtons.OKCancel,
MessageBoxIcon.Warning,
                    MessageBoxDefaultButton.Button1);
                if (result == DialogResult.OK)
                {
                    foreach (Process p in pProcess)
                    {
                        p.Kill();
                    }
                }
                else
                {
                    extraProcessAtStartup = true;
                    this.Close();
                }

            }
            excelApp = new Excel.Application();
            excelApp.DisplayAlerts = false;
            excelApp.Interactive = false;
```

```
            excelBooks = excelApp.Workbooks;

            lastBlowerTimeUpdate = DateTime.Now;

            /*********************************
             * Setup differential IRGA Graph
             * ******************************/
            GraphPane diffPane = zedGraphControl1.GraphPane;
            diffPane.Title.Text = "Differential IRGA\n";
            diffPane.XAxis.Title.Text = "Time, Seconds";
            diffPane.YAxis.Title.Text = "CO2 Consumption, um/m";
            diffPane.Title.FontSpec.Size = 24;
            diffPane.XAxis.Title.FontSpec.Size = 24;
            diffPane.YAxis.Title.FontSpec.Size = 24;

            // The RollingPointPairList is an efficient storage class
that always
            // keeps a rolling set of point data without needing to
shift any data values
            RollingPointPairList diffList = new
RollingPointPairList(100);

            // Initially, a curve is added with no data points (list is
empty)
            // Color is blue, and there will be no symbols
            LineItem diffCurve = diffPane.AddCurve("CO2 Consumption",
diffList, Color.Blue, SymbolType.None);

            // Just manually control the X axis range so it scrolls
continuously
            // instead of discrete step-sized jumps
            diffPane.XAxis.Scale.Min = 0;
            diffPane.XAxis.Scale.Max = 300;
            diffPane.XAxis.Scale.MinorStep = 5;
            diffPane.XAxis.Scale.MajorStep = 15;

            // Scale the axes
            zedGraphControl1.AxisChange();

            /*********************************
             * Setup absolute IRGA Graph
             * ******************************/
            GraphPane absPane = zedGraphControl2.GraphPane;
            absPane.Title.Text = "Absolute IRGA\n";
            absPane.XAxis.Title.Text = "Time, Seconds";
            absPane.YAxis.Title.Text = "CO2 Consumption, mm/m";
            absPane.Title.FontSpec.Size = 24;
            absPane.XAxis.Title.FontSpec.Size = 24;
            absPane.YAxis.Title.FontSpec.Size = 24;

            // The RollingPointPairList is an efficient storage class
that always
            // keeps a rolling set of point data without needing to
shift any data values
```

```
            RollingPointPairList absList = new
RollingPointPairList(100);

            // Initially, a curve is added with no data points (list is
empty)
            // Color is blue, and there will be no symbols
            LineItem absCurve = absPane.AddCurve("CO2 Consumption",
absList, Color.Blue, SymbolType.None);

            // Just manually control the X axis range so it scrolls
continuously
            // instead of discrete step-sized jumps
            absPane.XAxis.Scale.Min = 0;
            absPane.XAxis.Scale.Max = 300;
            absPane.XAxis.Scale.MinorStep = 5;
            absPane.XAxis.Scale.MajorStep = 15;

            // Scale the axes
            zedGraphControl2.AxisChange();

            //restore user values
            nudCO2ProportionalGain.Value =
Cuvette.Properties.Settings.Default.ProportionalGain;
            nudCO2IntegralGain.Value =
Cuvette.Properties.Settings.Default.IntegralGain;
            nudCO2SetValue.Value =
Cuvette.Properties.Settings.Default.CO2SetValue;
            lblCO2CurrentSetpoint.Text =
Cuvette.Properties.Settings.Default.CO2SetValue.ToString();
            nudAirFlowMagnitude.Value =
Cuvette.Properties.Settings.Default.AirFlowMagnitude;
            nudAirFlowTime.Value =
Cuvette.Properties.Settings.Default.AirFlowTime;
            nudDisplayDataDelay.Value =
Cuvette.Properties.Settings.Default.DisplayDataDelay;
            nudSaveDataDelay.Value =
Cuvette.Properties.Settings.Default.SaveDataDelay;

            co2AbsSetValue = nudCO2SetValue.Value;
            co2ProportionalConstant = nudCO2ProportionalGain.Value;
            co2IntegralConstant = nudCO2IntegralGain.Value;
        }

        private void Form1_FormClosing(object sender,
FormClosingEventArgs e)
        {
            try
            {
                if (extraProcessAtStartup == false)
                {
                    if (sp1.IsOpen == true)
                        sp1.Close();

                    if (excelFilename != null)
```

```
                        File.SetAttributes(excelFilename,
FileAttributes.Normal);

                    GC.Collect();
                    GC.WaitForPendingFinalizers();

                    excelApp.Interactive = true;
                    if (excelBook != null)
                        excelBook.Close(null, null, null);
                    excelApp.Workbooks.Close();
                    excelBooks.Close();
                    excelApp.Application.Quit();
                    excelApp.Quit();

                    if (excelBook != null)
                    {
                        Marshal.FinalReleaseComObject(excelRange);
                        Marshal.FinalReleaseComObject(excelSheet);
                        Marshal.FinalReleaseComObject(excelSheets);
                        Marshal.FinalReleaseComObject(excelBook);
                    }
                    Marshal.FinalReleaseComObject(excelBooks);
                    Marshal.FinalReleaseComObject(excelApp.Workbooks);
                    Marshal.FinalReleaseComObject(excelApp);

                    excelFilename = null;
                    excelRange = null;
                    excelSheet = null;
                    excelSheets = null;
                    excelBook = null;
                    excelBooks = null;
                    excelApp = null;

                    GC.Collect();
                    GC.WaitForPendingFinalizers();

                    Process[] pProcess =
Process.GetProcessesByName("Excel");
                    if (pProcess.Length != 0)
                    {
                        foreach (Process p in pProcess)
                        {
                            p.Kill();
                        }
                    }

                    //save user values

Cuvette.Properties.Settings.Default.ProportionalGain =
nudCO2ProportionalGain.Value;
                    Cuvette.Properties.Settings.Default.IntegralGain =
nudCO2IntegralGain.Value;
                    Cuvette.Properties.Settings.Default.CO2SetValue =
nudCO2SetValue.Value;
```

```
Cuvette.Properties.Settings.Default.AirFlowMagnitude =
nudAirFlowMagnitude.Value;
                    Cuvette.Properties.Settings.Default.AirFlowTime =
nudAirFlowTime.Value;

Cuvette.Properties.Settings.Default.DisplayDataDelay =
nudDisplayDataDelay.Value;
                    Cuvette.Properties.Settings.Default.SaveDataDelay =
nudSaveDataDelay.Value;
                    Cuvette.Properties.Settings.Default.BlowerTime +=
DateTime.Now - lastBlowerTimeUpdate;
                    Cuvette.Properties.Settings.Default.Save();
                }
            }
            catch (Exception theException)
            {
                String errorMessage;
                errorMessage = "Error: ";
                errorMessage = String.Concat(errorMessage,
theException.Message);
                errorMessage = String.Concat(errorMessage, " Line: ");
                errorMessage = String.Concat(errorMessage,
theException.Source);

                MessageBox.Show(errorMessage, "Error");
            }
        }
        #endregion
        #region MFCStatus
        private void nudCO2ProportionalGain_ValueChanged(object sender,
EventArgs e)
        {
            co2ProportionalConstant = nudCO2ProportionalGain.Value;
        }

        private void nudCO2IntegralGain_ValueChanged(object sender,
EventArgs e)
        {
            co2IntegralConstant = nudCO2IntegralGain.Value;
        }

        private void btnCO2SetFlow_Click(object sender, EventArgs e)
        {
            co2AbsSetValue = nudCO2SetValue.Value;
            lblCO2CurrentSetpoint.Text = co2AbsSetValue.ToString();
        }

        private void btnAirSetFlow_Click(object sender, EventArgs e)
        {
            if (nudAirFlowTime.Value == 0)
            {
                sendCommand(E, nudAirFlowMagnitude.Value);
            }
            else
```

```
            {
                rampAirMFC();
                tmrAirMFC.Enabled = true;
            }
        }

        private void btnAirReadFlow_Click(object sender, EventArgs e)
        {
            nudAirFlowMagnitude.Value = (decimal)sendCommand(E,
(decimal)-1);
        }

        private void btnCO2MFVStatus_Click(object sender, EventArgs e)
        {
            MFCStatus MFVStatus = new
MFCStatus(D,sp1,tmrRead,readThread);
            MFVStatus.ShowDialog();
        }

        private void btnAirMFVStatus_Click(object sender, EventArgs e)
        {
            MFCStatus MFVStatus = new MFCStatus(E, sp1, tmrRead,
readThread);
            MFVStatus.ShowDialog();
        }

        private void rampAirMFC()
        {
            decimal futureValue = nudAirFlowMagnitude.Value;
            decimal currentValue = sendCommand(E, (decimal)-2);
            airSpacing = (futureValue - currentValue) /
nudAirFlowTime.Value / 4;
            airSum = currentValue + airSpacing;
            airNumIterations = (int)nudAirFlowTime.Value * 4;
        }

        private void tmrAirMFC_Tick(object sender, EventArgs e)
        {
            if (airIterations <= airNumIterations)
            {
                sendCommand(E, airSum);
                airSum += airSpacing;
                airIterations++;
            }
            else
            {
                tmrAirMFC.Enabled = false;
                airIterations = 1;
            }
        }
        #endregion
        #region sendCommand
        private void sendCommand(int servoNumber, int servoValue)
        {
            bool isTimerEnabled;
```

```
            if (tmrRead.Enabled == true)
                isTimerEnabled = true;
            else
                isTimerEnabled = false;

            //wait for readThread to release serial port
            if (readThread != null)
            {
                tmrRead.Enabled = false;
                //wait for thread to finish
                while (readThread.IsAlive)
                {
                    Application.DoEvents();
                }
            }

            try
            {
                //lock serial port
                lock (sp1)
                {
                    if (sp1.IsOpen == false)
                        sp1.Open();

                    //create byte array for writing
                    /*{fixed preamble, user defined preamble,
                       switch address, port}*/
                    byte[] switchToStage = { ESC, STX, B, B };
                    //write to serial switch
                    sp1.Write(switchToStage, 0, switchToStage.Length);

                    //equation to convert to number for controller
                    int value1 = 0;
                    if (servoNumber == 0)
                    {
                        value1 = 38 * servoValue + 900;
                    }
                    else if (servoNumber == 1)
                    {
                        value1 = 30 * servoValue + 600;
                    }
                    else
                    {
                    }
                    //number one servo, position...
                    sp1.WriteLine("#" + servoNumber + "P" +
value1.ToString() + "\r");
                    sp1.Write(closePort, 0, closePort.Length);
                    sp1.Close();
                }
            }
            catch (Exception theException)
            {
                String errorMessage;
                errorMessage = "Error: ";
```

```
                errorMessage = String.Concat(errorMessage,
theException.Message);
                errorMessage = String.Concat(errorMessage, " Line: ");
                errorMessage = String.Concat(errorMessage,
theException.Source);

                MessageBox.Show(errorMessage, "Error");
            }
            if (isTimerEnabled == true)
                tmrRead.Enabled = true;
        }
        private decimal sendCommand(byte MFVport, decimal flowValue)
        {
            decimal decReadValue = 0;
            /*bool isTimerEnabled;
            if (tmrRead.Enabled == true)
                isTimerEnabled = true;
            else
                isTimerEnabled = false;

            //wait for readThread to release serial port
            if (readThread != null)
            {
                tmrRead.Enabled = false;
                //wait for thread to finish
                while (readThread.IsAlive)
                {
                    Application.DoEvents();
                }
            }*/

            try
            {
                //lock serial port
                lock (sp1)
                {
                    if (sp1.IsOpen == false)
                        sp1.Open();

                    /*{fixed preamble, user defined preamble,
                                        switch address, port}*/
                    byte[] sendCommand = { ESC, STX, A, MFVport };
                    //write to serial switch
                    sp1.Write(sendCommand, 0, sendCommand.Length);
                    //empty input buffer
                    if (sp1.BytesToRead != 0)
                        sp1.DiscardInBuffer();
                    //set flow

/*******************************************************
                     * Backslash characters are needed, because when
changing
                     * to a different port two characters are sent
through the
```

```
                     * new port.  Not backslashing results in error
from MFVs.
                     *
**************************************************/
                    if (flowValue == -1)
                    {
                        //get setpoint
                        sp1.Write("\b\b\bv 4\r");
                        //wait for bytes
                        while (sp1.BytesToRead == 0) ;
                        decReadValue =
Convert.ToDecimal(sp1.ReadTo("\r"));
                        if (MFVport == 'D')
                            nudCO2SetValue.Value = decReadValue;
                        else
                            nudAirFlowMagnitude.Value = decReadValue;
                        sp1.Write(closePort, 0, closePort.Length);
                        sp1.Close();
                    }
                    else if (flowValue == -2)
                    {
                        //get setpoint
                        sp1.Write("\b\b\bv 4\r");
                        //wait for bytes
                        while (sp1.BytesToRead == 0) ;
                        decReadValue =
Convert.ToDecimal(sp1.ReadTo("\r"));
                        sp1.Write(closePort, 0, closePort.Length);
                        sp1.Close();
                    }
                    else
                    {
                        sp1.Write("\b\b\bv 4 =" + flowValue + "\r");
                        sp1.Write(closePort, 0, closePort.Length);
                        sp1.Close();
                    }
                }
            }
            catch (Exception theException)
            {
                String errorMessage;
                errorMessage = "Error: ";
                errorMessage = String.Concat(errorMessage,
theException.Message);
                errorMessage = String.Concat(errorMessage, " Line: ");
                errorMessage = String.Concat(errorMessage,
theException.Source);

                MessageBox.Show(errorMessage, "Error");
            }
            //if (isTimerEnabled == true)
            //    tmrRead.Enabled = true;

            return decReadValue;
        }
```

```
        #endregion
        #region blower
        private void getBlowerTimeToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            Cuvette.Properties.Settings.Default.BlowerTime +=
DateTime.Now - lastBlowerTimeUpdate;
            lastBlowerTimeUpdate = DateTime.Now;
            MessageBox.Show("Cumulative blower time is: " +
Cuvette.Properties.Settings.Default.BlowerTime.ToString()
                + "\nDays.Hours:Minutes:Seconds.Ticks","Blower
Time",MessageBoxButtons.OK,MessageBoxIcon.Information);
        }

        private void reToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            DialogResult result = MessageBox.Show("Are you sure you
want to reset the cumulative blower time?",
                "Blower Time", MessageBoxButtons.YesNo,
MessageBoxIcon.Warning,MessageBoxDefaultButton.Button2);
            if (result == DialogResult.No)
                return;
            Cuvette.Properties.Settings.Default.Reset();
            lastBlowerTimeUpdate = DateTime.Now;
        }
        #endregion
        #region lights
        private void btnSP1_Click(object sender, EventArgs e)
        {
            toggleLightsButtons(btnSP1);
        }

        private void btnSP2_Click(object sender, EventArgs e)
        {
            toggleLightsButtons(btnSP2);
        }

        private void btnSP3_Click(object sender, EventArgs e)
        {
            toggleLightsButtons(btnSP3);
        }

        private void btnSP4_Click(object sender, EventArgs e)
        {
            toggleLightsButtons(btnSP4);
        }

        private void btnSP5_Click(object sender, EventArgs e)
        {
            toggleLightsButtons(btnSP5);
        }

        private void btnSP6_Click(object sender, EventArgs e)
        {
```

```
            toggleLightsButtons(btnSP6);
        }

        private void btnSP7_Click(object sender, EventArgs e)
        {
            toggleLightsButtons(btnSP7);
        }

        private void btnSP8_Click(object sender, EventArgs e)
        {
            toggleLightsButtons(btnSP8);
        }

        private void btnSP9_Click(object sender, EventArgs e)
        {
            toggleLightsButtons(btnSP9);
        }

        private void btnSP10_Click(object sender, EventArgs e)
        {
            toggleLightsButtons(btnSP10);
        }

        private void toggleLightsButtons(System.Windows.Forms.Button
button)
        {
            if (button.Text == "OFF")
            {
                button.Text = "ON";
                button.BackColor = Color.Green;
            }
            else
            {
                button.Text = "OFF";
                button.BackColor = Color.Transparent;
            }
        }
        private void checkLights()
        {
            Button[] arrayLightButtons =
                new Button[10] { btnSP1, btnSP2, btnSP3, btnSP4,
btnSP5, btnSP6, btnSP7, btnSP8,
                    btnSP9, btnSP10 };
            NumericUpDown[] arrayLightHours =
                new NumericUpDown[10] {nudHourSP1, nudHourSP2,
nudHourSP3, nudHourSP4, nudHourSP5,
                    nudHourSP6, nudHourSP7, nudHourSP8, nudHourSP9,
nudHourSP10};
            NumericUpDown[] arrayLightMins =
                new NumericUpDown[10] {nudMinSP1, nudMinSP2, nudMinSP3,
nudMinSP4, nudMinSP5,
                    nudMinSP6, nudMinSP7, nudMinSP8, nudMinSP9,
nudMinSP10};
```

```
            //find the latest time that is less than or equal to
current time
            int comparetime = 0;
            for (int i = 1; i < arrayLightHours.Length; i++)
            {
                if (arrayLightHours[i].Value <= DateTime.Now.Hour)
                {
                    if (arrayLightMins[i].Value <= DateTime.Now.Minute)
                    {
                        if (arrayLightHours[i].Value >=
arrayLightHours[comparetime].Value)
                        {
                            if (arrayLightMins[i].Value >=
arrayLightMins[comparetime].Value)
                            {
                                comparetime = i;
                            }
                        }
                    }
                }
            }
            if (arrayLightButtons[comparetime].Text == "ON")
            {
                lblLights.Text = "ON";
                lblLights.BackColor = Color.Green;
            }
            else
            {
                lblLights.Text = "OFF";
                lblLights.BackColor = Color.Transparent;
            }
        }
        private void btnCheckLights_Click(object sender, EventArgs e)
        {
            checkLights();
        }
        #endregion
    }
}
```

MFCStatus.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.IO.Ports;
using System.Text;
using System.Threading;
```

```
using System.Windows.Forms;

namespace Cuvette
{
    public partial class MFCStatus : Form
    {
        #region declarations
        byte port;
        SerialPort sp1;
        System.Windows.Forms.Timer tmrRead;
        Thread readThread;
        string currentDirectory = Application.StartupPath;

        bool isTimerEnabled;
        bool isPortOpen;

        const byte STX = 0x02;
        const byte EOT = 0x04;
        const byte ESC = 0x1b;
        const byte A = 0x41;
        const byte B = 0x42;
        const byte C = 0x43;
        const byte D = 0x44;
        const byte E = 0x45;

        byte[] closePort = { ESC, STX, EOT };

        string returnWord = null;
        #endregion
        public MFCStatus(byte passedport, SerialPort passedsp,
System.Windows.Forms.Timer passedtmr, Thread passedthread)
        {
            InitializeComponent();
            port = passedport;
            sp1 = passedsp;
            tmrRead = passedtmr;
            readThread = passedthread;
        }

        private void MFVStatus_Load(object sender, EventArgs e)
        {
            if (port == Convert.ToByte('D'))
            {
                this.Text = "CO2 MFC Status";
                lblAlarmHighUnit.Text = "slm";
                lblAlarmLowUnit.Text = "slm";
                lblWarningHighUnit.Text = "slm";
                lblWarningLowUnit.Text = "slm";
                nudAlarmHighPoint.Maximum = 0.25M;
                nudAlarmLowPoint.Maximum = 0.25M;
                nudWarningHighPoint.Maximum = 0.25M;
                nudWarningLowPoint.Maximum = 0.25M;
            }
            else if (port == Convert.ToByte('E'))
            {
```

```
                this.Text = "Air MFC Status";
                lblAlarmHighUnit.Text = "slm";
                lblAlarmLowUnit.Text = "slm";
                lblWarningHighUnit.Text = "slm";
                lblWarningLowUnit.Text = "slm";
                nudAlarmHighPoint.Maximum = 30;
                nudAlarmLowPoint.Maximum = 30;
                nudWarningHighPoint.Maximum = 30;
                nudWarningLowPoint.Maximum = 30;
            }

            if (tmrRead.Enabled == true)
            {
                isTimerEnabled = true;
                tmrRead.Enabled = false;
            }
            else
                isTimerEnabled = false;
            if (sp1.IsOpen == true)
                isPortOpen = true;
            else
                isPortOpen = false;
        }

        private void MFCStatus_FormClosing(object sender,
FormClosingEventArgs e)
        {
            try
            {
                if (isTimerEnabled == true)
                    tmrRead.Enabled = true;
                if (isPortOpen == true)
                    if (sp1.IsOpen == false)
                        sp1.Open();
            }
            catch
            {
            }
        }

        #region help
        private void linkLabel1_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
        {
            helpAlarmsWarnings();
        }

        private void linkLabel2_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
        {
            helpAlarmsWarnings();
        }

        private void linkLabel3_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
```

```
        {
            helpAlarmsWarnings();
        }

        private void helpAlarmsWarnings()
        {
            try
            {
                Directory.SetCurrentDirectory(currentDirectory);
                System.Diagnostics.Process.Start("Cuvette Help.pdf");
            }
            catch
            {
                MessageBox.Show("Help file not found.\r\nPlease place
help file in the same folder as the program.\r\n",
                    "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }
        #endregion

        private void sendCommand(string command, string value)
        {
            try
            {
                //lock serial port
                lock (sp1)
                {
                    if (sp1.IsOpen == false)
                        sp1.Open();
                    //create byte array for writing
                    /*{fixed preamble, user defined preamble,
                       switch address, port}*/
                    byte[] switchToMFV = { ESC, STX, A, port };
                    //write to serial switch
                    sp1.Write(switchToMFV, 0, switchToMFV.Length);
                    //empty input buffer
                    if (sp1.BytesToRead != 0)
                        sp1.DiscardInBuffer();

/*********************************************************
                     * Backslash characters are needed, because when
changing
                     * to a different port two characters are sent
through the
                     * new port.  Not backslashing results in error
from MFVs.
                     *
*********************************************************/
                    if (value == "return value")
                    {
                        sp1.Write("\b\b\b" + command + "\r");
                        //wait for bytes
                        while (sp1.BytesToRead == 0) ;
                        returnWord = sp1.ReadTo("\r");
```

```
                    }
                    else
                    {
                        sp1.Write("\b\b\b" + command + value + "\r");
                    }
                    sp1.Write(closePort, 0, closePort.Length);
                    sp1.Close();
                }
            }
            catch (Exception theException)
            {
                String errorMessage;
                errorMessage = "Error: ";
                errorMessage = String.Concat(errorMessage,
theException.Message);
                errorMessage = String.Concat(errorMessage, " Line: ");
                errorMessage = String.Concat(errorMessage,
theException.Source);

                MessageBox.Show(errorMessage, "Error");
            }

        }

        #region Alarm/Warning Status
        private void btnCurrentAlarms_Click(object sender, EventArgs e)
        {
            sendCommand("ma", "return value");
            lblAlarmStatus.Text = returnWord;
        }

        private void btnLatchedAlarms_Click(object sender, EventArgs e)
        {
            sendCommand("maa", "return value");
            lblAlarmStatus.Text = returnWord;
            btnLatchedAlarms.Enabled = false;
            btnClearLatchedAlarms.Enabled = true;
        }

        private void btnClearLatchedAlarms_Click(object sender,
EventArgs e)
        {
            sendCommand("maa =", lblAlarmStatus.Text);
            lblAlarmStatus.Text = "";
            btnClearLatchedAlarms.Enabled = false;
            btnLatchedAlarms.Enabled = true;
        }

        private void btnCurrentWarnings_Click(object sender, EventArgs
e)
        {
            sendCommand("mw", "return value");
            lblWarningStatus.Text = returnWord;
        }
```

```
        private void btnLatchedWarnings_Click(object sender, EventArgs
e)
        {
            sendCommand("mwa", "return value");
            lblWarningStatus.Text = returnWord;
            btnLatchedWarnings.Enabled = false;
            btnClearLatchedWarnings.Enabled = true;
        }

        private void btnClearLatchedWarnings_Click(object sender,
EventArgs e)
        {
            sendCommand("mwa =", lblWarningStatus.Text);
            lblWarningStatus.Text = "";
            btnClearLatchedWarnings.Enabled = false;
            btnLatchedWarnings.Enabled = true;
        }

        private void btnCurrentFlow_Click(object sender, EventArgs e)
        {
            sendCommand("mf", "return value");
            lblFlowStatus.Text = returnWord;
        }

        private void btnLatchedFlow_Click(object sender, EventArgs e)
        {
            sendCommand("mfa", "return value");
            lblFlowStatus.Text = returnWord;
            btnLatchedFlow.Enabled = false;
            btnClearLatchedFlow.Enabled = true;
        }

        private void btnClearLatchedFlow_Click(object sender, EventArgs
e)
        {
            sendCommand("mfa =", lblFlowStatus.Text);
            lblFlowStatus.Text = "";
            btnClearLatchedFlow.Enabled = false;
            btnLatchedFlow.Enabled = true;
        }
        #endregion
        #region Set/Read Alarm/Warning Points
        private void btnSetAlarmHigh_Click(object sender, EventArgs e)
        {
            sendCommand("g 9 =", nudAlarmHighPoint.Value.ToString());
        }

        private void btnSetAlarmLow_Click(object sender, EventArgs e)
        {
            sendCommand("g 11 =", nudAlarmLowPoint.Value.ToString());
        }

        private void btnSetWarningHigh_Click(object sender, EventArgs
e)
        {
```

```
            sendCommand("g 13 =",
nudWarningHighPoint.Value.ToString());
        }

        private void btnSetWarningLow_Click(object sender, EventArgs e)
        {
            sendCommand("g 15 =", nudWarningLowPoint.Value.ToString());
        }

        private void btnReadAlarmHigh_Click(object sender, EventArgs e)
        {
            sendCommand("g 9", "return value");
            nudAlarmHighPoint.Value = Convert.ToDecimal(returnWord);
        }

        private void btnReadAlarmLow_Click(object sender, EventArgs e)
        {
            sendCommand("g 11", "return value");
            nudAlarmLowPoint.Value = Convert.ToDecimal(returnWord);
        }

        private void btnReadWarningHigh_Click(object sender, EventArgs
e)
        {
            sendCommand("g 13", "return value");
            nudWarningHighPoint.Value = Convert.ToDecimal(returnWord);
        }

        private void btnReadWarningLow_Click(object sender, EventArgs
e)
        {
            sendCommand("g 15", "return value");
            nudWarningLowPoint.Value = Convert.ToDecimal(returnWord);
        }
        #endregion
        #region pause
        //pause function
        public static DateTime Pause(int ms)
        {
            //look at current time
            System.DateTime ThisMoment = System.DateTime.Now;
            //set delay duration
            System.TimeSpan duration = new System.TimeSpan(0, 0, 0, 0,
ms);
            //duration end
            System.DateTime AfterWards = ThisMoment.Add(duration);

            //inside time frame start and end
            while (AfterWards >= ThisMoment)
            {
                //work during pause
                System.Windows.Forms.Application.DoEvents();
                //monitor current system time
                ThisMoment = System.DateTime.Now;
            }
```

```
            return System.DateTime.Now;
        }
        #endregion
    }
}
```