Requirement Specifications: Astrophysical Materials Lab Data Logging and Instrument Control Tool

Quinton Jasper, Hui Wan, Elijah Anakalea-Buckley, Hao Liu

11/6/2020

Contents

1 Introduction

With breakthroughs in technology, the scientific pursuit of understanding the universe was always initiated from mankind's own perspective, here on Earth. And in that time, it has acquired a relatively intimate understanding within Earth's domain. Earth's geology and atmospheric environment come to us as second nature: a blue sky, liquid water, and a surface crust that is carved and molded by an energy-rich climate. Of course, this is a rare environment on the astronomical scale. In the Milky Way galaxy alone, there are an estimated 100 billion planets[]. Of that, it appears less than 3% can be deemed Earth-like, with similar proximity to the sun and similar environmental properties[].As for other planetary bodies in the universe, they experience a range of inhospitable environments, such as Uranus's average surface temperature of 49 Kelvin[], as planets become more unfathomably cold the farther they are from a star. These are the worlds that TITAN's clients are working to better understand. For this project, team TITAN is operating on the core data-collection system for Lowell Observatory's Astrophysical Ice Lab on NAU campus. Led by Dr. Will Grundy and Dr. Jennifer Hanley, researchers at the Ice Lab make use of a custom-engineered cryochamber which simulates the unbelievably cold temperatures found only in the harshest regions of space, while studying its effects on elements and chemical compounds. For the purposes of TITAN's involvement in the project, the clients' cryochamber rig consists of several engineered components: the plexiglass freezing chamber, upper and lower temperature sensors, a temperature controller, pressure regulator, lower heating system, and cooling system. To make use of this system, Dr. Grundy has established a Linux computing environment that executes custom software which primarily interfaces with the system's temperature and pressure sensors and stores collected temperature data into a standard CSV spreadsheet, which itself is stored within a structured folder hierarchy on the filesystem.

2 Problem Statement

The research done in the astrophysical materials lab on the NAU campus is conducted through various instruments that provide temperature control, pressure readings, multiple sensors as well as safety for the researchers. All these tools are used to simulate the conditions of extraterrestrial bodies. Currently all these tools are being controlled through a system built by Dr. Grundy and his team. The system in place handles the tools and extracts data from the sensors, dumps it into files to later be analyzed. The problem with the system in place is that it's operating based on outdated software technologies, as well as limited in its functionality. It's a homebuilt solution that has the core essentials for their research in mind but not much else. The software places limits on hardware controllability, and workflow efficiency. Team Titans looks to provide a more professional-grade solution. A full redesign with modern graphical interface, implemented on modern technologies and modular architecture.

3 Solution Vision

Team Titan will first provide a completely updated version to the current system, re-written in the most recent python syntax and corresponding libraries. This refresh of the old system will allow research to be continued while the team works to further develop new features. The goal is to provide a totally up-to-date base product, built with a clean, modular structure that will allow users, whether it be the Ice Lab team or TITAN, to add and remove features without sacrificing the integrity of the code's core functionality. The bulleted list below shows a list of broad solutions that team titan looks to provide to the client separated into major sections: User Interface Updated Professional look (determined through visual prototypes to client) Live Data Plotting (Reach Requirement)Front-end Access to the Database Lab Hardware Connectivity Secure stable connections between "data-production" and "data-collection" Determine route for connectivity (USB or Network) Modularity Recreate the system with modular components that are self-contained within the library/file but not dependent on other aspects of the system. Establish Framework for accepting new tools/devices Framework for New UI add-ons Data Handling Data safe storage & Data Recovery Alarm Systems based on Data Exporting to usable format (Reach Requirement)Database Traversal

The features above will be prioritized to ensure that the client is getting the most crucial aspects of the system completed in the time provided. "Reach" features will be implemented after the essentials. These features along with the capabilities that the system already has will be the complete solution that Team Titan provides for the clients and can serve as a base for modular research equipment in any field of study.

4 Project Requirements

4.1 Functional Requirements

The numbered sections below are the domain/high level requirements followed by the "in-depth" technical requirements that follow. The domain level requirements have no priority to them, but the sub-requirements below will be implemented from high priority to low or from top to bottom. Each domain level requirement will be its own file module to create an easy to manage modular system. Hardware → System Connectivity Hardware connectivity module will allow for creation of custom "driver" scripts, where users can connect different lab hardware with the main computer The UI will feature a notification area, which will provide human-readable status updates for hardware connectivity If hardware (USB, Ethernet port) is disconnected, a message in the upper-right corner with red background will alert that there is no connection If hardware is successfully connected, a message in the upper-right corner with green background will alert connection success. Hardware connectivity module will allow hardware reconnection with system breaking down or power failure Hardware connectivity module will allow data resuming/recovering from the data log files Network connectivity module will provide shared-private network It is shared for authorized users or staffs and workers in the lab It is private for any other person not relevant to this project

4.2 User Interface (Temperature Control Module)

Abstractions for Common commands Construction of a dictionary with hardware-level commands stored for abstraction (making aliases for temperature controller commands) Sample from DAQ commands = ["?AI", "AI:CHMODE=DIFF", "AI{0}:RANGE=BIP1V", "AI{1}:RANGE=BIP1V", "AI{2}:RANGE=BIP10V", "AI{3}:RANGE=BIP10V", #"?AI{0}:VALUE", "?AI{0}:VALUE", "?AI{1}:VALUE", "?AI{2}:VALUE", "?AI{3}:VALUE"]

Pop-out window which will provide users can enter commonly-used hardware API commands, and save them so they can be accessed again later. Text box will be front and center, where users can enter commands Save button will store those commands and place them in a list widget to the side, where they can be selected and executed using an ‘Execute’ button just below

User Interface (Data Plotting) Graphical data-plotting module will provide X, Y axis scaling controls. In a similar manner to the previous implementation, it will be in the form of text boxes placed in sensible locations near the plot widget

Graphical data-plotting module will allow the user to toggle different data input streams via checkbox group which will be placed just to the side of the plot. Graph data will be updated via a “refresh” function that will be called whenever UI modifications are made, or a certain refresh time has passed UI Modifications are considered: Window resize, graph axis-scale changes when new sensor is toggled on/off,

User Interface (Alarm System) Alarm UI will allow users to enter temperature and pressure bounds as alarm parameters within lower and upper-bound text boxes. Temperature bounds will encompass the cases of: if temperature is above T°(Celsius/Fahrenheit/Kelvin) and/or if temperature is below T°(C/F/K) Pressure bounds will encompass the cases of: if pressure is above P(ppi/atm/etc.) and/or if pressure is below P(ppi/atm/etc.) Alarm UI will provide pre-made Linux command abstractions via drop-down menu for different alarm methods

The Linux system can send an email using the built-in ‘mail’ command. It would explain why the email was sent, what prompted the email (error or manual alarm set), and time sent. Users can choose to have an audio alert, which will have the computer make an audible sound. In addition to the sound, the alarm button itself will also change to the color red to provide visual feedback Event system will need to be put in place, such that when the “event” of an alarm is triggered, subsequent, relevant functions will be called There will be an option that allows the user to enter a custom Linux command, in case a custom solution is needed

Data Handling Accurate interface with database/data logging system to connect to the UI live data plotting Safe Data Storage Secure Data Organization Data Recovery “Save Experiment” notification to avoid data loss on the user end Connect with UI software to display alarms based on the data received from the sensors (Reach Requirement) Create a query interface to traverse the database/repository based on the needs of the user.

4.3 Performance Requirements

The following paragraphs will explain the performance requirements as well as verifiability methods for the corresponding domain level functional requirements:

Hardware Connectivity: The expected setup for connecting lab equipment to the central computer will be via wired Ethernet and USB. It is expected that the user interface will be able to render data results to the screen while the data-handling component organizes and saves that data into a database structure. In this situation, a “latency budget” will be developed through testing and the data transferring latency between hardware in the lab and main computers will be calculated. It’s expected to remain below the calculated upper bounds and above calculated lower bounds to avoid overloads or crashes while ensuring good performance. Also, the connectivity will protect the system from power failure, system crash, network outage or external damage. Then, data can be recovered from the log files stored in the spare equipment where their capability should be more than N MB/GB/TB in case of loss of data. The data stored in the main computers or hardware in the lab should be recorded to a memorizer which is used to store data specially and then covered every Y days/weeks to ensure the efficiency of connectivity and transmission.

Data Visualization Module: The data visualization will operate off a timed “clock” method, this will force the UI to refresh its contents and render new UI visuals within the confines of n seconds, as is designated by the user, and is no less than 2 seconds. Taking into consideration the overhead of UI data rendering, it will be expected that it will take no more than n seconds to execute the process of: extracting equipment data, and re-rendering the plot to the screen, while handling >1000 data points rendered to the screen.

Alarm UI Module: The main aspects of performance for the Alarm module are to ensure processing of the current time, processing comparisons between the alarm’s set time and real-world time, and processing comparisons between user-set temperature/pressure bounds and each respective input data-stream. The goal is to make sure latency of these calculations is under 0.5 seconds. To phrase it another way, ≤ 1.5 seconds per UI data refresh User Interface As A Whole: The main concern with user interfaces as a whole is the concept of multithreading (or forgetting to implement multithreading) which can result in frozen/unresponsive windows. The UI window itself will need to be light-weight enough to maintain ≥ 60 fps screen refresh-rate. Multithreading comes into place when the UI itself is operating child-tasks—such as refreshing/rendering the data visualization—and ensures the child task does not use the same system resources designated for the UI. The UI designer needs to implement multithreading such that no UI freezes/crashes occur when child processes are called.

Data Handling: The data reflected in the live plotting will be accurate and a mirror of the data being gathered and logged from the sensors. This will be measured when testing the system as a whole and monitoring both aspects to ensure consistency. Safe Data Storage will ensure that all data is being recorded and organized. This will be verified through recovering data from the storage. There will also be a "Save Experiment" button if the user has not saved to avoid user error. The alarm system will work cohesively with data handling and will be tested based on accurate triggers and conditions will be manually changed to trigger alarms. If the rest of the features are achieved, traversing the database will come next. It will be based around a configurable interface that allows the user to create queries for the database. It will cut down the database to make a browsable page so that the user may find desired data.

Environmental Requirements

The restrictions on the technologies and libraries being used is based on the capabilities of the lab and the technologies it holds:

The application will operate on the latest version of the Linux environment Fedora (v33) and testing will ensure that all functions can properly operate under the environment conditions Local ethernet and USB serial connections will be the two modes of lab equipment connection and data flow. The database syntax will follow what is required for MariaDB to ensure Dr. Grundy's maintainability The latest version of Python installed by default on Fedora 33 (Python 3.9.0) will be used to develop the application The application will operate on top of the Qt5 UI library (Qt version 5.15.1 LTS) PyQt5 will act as the bindings between Python and the Qt Framework Hooks and external Linux commands will be written using BASH scripting syntax

5 Potential Risks

In the Feasibility Document, many technological challenges were put forward in terms of the user interface, data logging, hardware connectivity and network compatibility. Accordingly, some solutions were come up with each section. The likelihood of each case will be assessed using the following terminology: 0%-30% chance of occurrence: Low Probability 34%-70% chance of occurrence: Medium Probability >71% chance of occurrence: High Probability

The top choice is to use the Python 3 MySQL library. Though MySQL is effective and works well with general Python programs, there exists the incompatibility between floating-point values and standard integers. If an unanticipated floating-point value or int value is entered into the MySQL database, provided the database is expecting the opposite, the database will store inaccurate, or completely incorrect data. There is a low probability of this occurring, as the input data stream is predictable and can be better understood through documentation. To reduce chances to zero, the team will integrate functionality that will allow users to verify the changes to the database before final submission.

The method of saving data, and extracting data to render to the screen follows the process: (Data input from hardware → append data to the log file (CSV) → Read data from the CSV → Render CSV data to the graph.) This is to ensure the data is saved first before it is manipulated. However, if the team were to implement the exact same system with a database, it is likely that the computer system, a desktop PC, will not be able to handle the operation of opening an entire database table. There is a high probability that the desktop hardware (RAM and CPU power) will not be sufficient, leading to desktop crashes and data loss. To counter this, the database logic written in python will need to be optimized for limited database entries and queries (only affecting the most recent/relevant entry in the table). This will significantly reduce RAM usage and speed up the write → read → render pipeline. The previous version of the system uses Python 2 and GTK 2, which have since been deprecated. Provided Fedora's updates continue to roll-out, there is a medium probability that software dependencies will become deprecated and be removed from the system entirely. To counter this, there are two options. From discussions with the client, the solution agreed upon was to pin the system version, in this case Fedora v33, such that major Linux repository changes will not have any effect unless the client chooses to update to the next major Fedora release. In this case, that would be Fedora v34.

A major feature of the desktop software is control of the temperature regulator. As this involves physical temperature and pressure manipulation, there is a risk of lab equipment damage. The system could find itself overly pressurized if the machines are warmed or cooled too quickly. Dr. Grundy and Dr. Hanley have implemented pressure fail-safes that will prevent the cryo-chamber from shattering. However, in the case that these fail safes are activated, all facets of data accuracy and system reliability are lost until the system is manually reset. This occurrence would result in wasted time and effort from the entire team. There is a Low Probability of this occurring. To bring the chance all the way to zero, the team will follow the temperature-change rate logic Dr. Grundy has already templated in the previous software implementation. The team will also work closely with the clients who better understand the physical constraints of equipment materials.

6 Project Plan

As of the writing of this document, team Titan has established 11 major milestones for the team to achieve as this project progresses. Among the first, which the team is already working to complete, is to obtain a deep understanding of the project itself and how it operates before the team lays any hands on it. After acquiring project requirements from the client, the team will begin the development process, encompassing the following high-level steps in semi-progressive order:

Communication with the client to establish requirements

Establishing a full development environment Finalizing all project dependencies Decide on an appropriate Linux Environment Ensure this development environment is reproducible

Establish Testing Schedule in Lab Communicate with clients a schedule to test the software developed using the tools that will be controlled with software

Successfully establish a data-capturing backend. A rough application should be able to capture data from the appropriate data channels, decode the raw data input stream from the test data output, and print the data to a command line interface

Design the data visualization module The only functionality of this module will be to take an input of the captured data stream and render it into a predetermined graph format.

Design the Data Logging System Create a python module/API which will provide callable functions for data-logging. Design an organized SQL table structure Integrate this logic with the user interface

Design work for Alarm UI system Establish Alarm methods (System beep, Email messages) Create Alarm module in python Interface python front-end with "hooks" for abstracting Linux system commands

Final unification of modular components with the user interface The user interface, for the scope of this project, will act as the main entry-point for program execution. The team will ensure that each modular component interfaces with the entry-point.

Test System Run mock experiment with developed software

Refactoring Implement needed improvements that are discovered after testing

Create Project API Documentation

As displayed in Figure 1, the team has created a Gantt chart to help visualize the estimated timeframe for each step in the process. At this time, external obligations from the Capstone course itself are not known, though the team anticipates completion of the core project code before the end of March. This will allow ample time for the team to parse, correct, and document the code before May, when the final project should be prepared for demonstration and submission to the client.

November 16th December 8th January 15th January 30th February 13th February 28th March 10th March 26th April 9th April 16th April 29th Requirements Engineering

Establish Development Environment

Establish Testing Schedule in Lab

Create Data-capturing Back-end

Design Data Visualization Module

Design Data Logging System

Design Alarm UI System

Final Module Unification with UI Entry-Point

Testing System in Real Environment

Refactoring and Bug Fixes/Reach Requirements

Final Documentation Stage

Figure 1: Estimated time allocation for each task over the course of the fall/spring semester

7 Conclusion

For this project, team TITAN is working on the core data-collection system for Lowell Observatory's Astrophysical Ice Lab on NAU campus. The system collects temperature data and controls the several engineered components (temperature controller and pressure sensor). However, the system has several problems that need fixing and improving. As the system is working based on outdated software technologies, its functionalities and support lifecycle are severely limited. All code is written in one file, making it difficult to make revisions when bugs are found. Because of the current software's hand-made systems, it's difficult to maintain stable performance and system efficiency, as no low-level computing optimizations are in place. These problems will likely cause trouble for laboratory staff, preventing jobs from being completed on time, or worse, result in a total system shutdown. It's crucial that the team pays close attention to these problems and commit to fixing them before they present themselves while in production. As such, Python 3 along with new external libraries will be used to rewrite code and implement new logic that will better handle the test of time as the project scales. The end goal is to modularize the code. All code is divided into different parts according to their function and types. The team has been made aware of the problems the old system has and has worked diligently to establish some useful solutions. Time has been spent analyzing potential system-wide risks so that we can easily avoid them in the development and testing stages. The most important thing is that the team provides a specific project plan, establishing a schedule that will aid in making progress. The team expects to make a difference to the system within the timeframe as expected.