# Liquid Level Indicator (LLI) Utilizing an Optical Time-of-Flight Sensor for Clean Agent Fire Suppression System Cylinders

by

Ian Stumpe

A Report Submitted to the Faculty of the

Milwaukee School of Engineering

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Engineering

Milwaukee, Wisconsin
May 2023

## Abstract

The purpose of this report is to describe, explain and discuss the results associated with the author's Milwaukee School of Engineering (MSOE) Master of Science in Engineering (MSE) capstone project. The purpose of the capstone project is to design a Time-of-Flight (TOF) sensor-based liquid level indicator/sensor (LLI or LLS) to measure fire suppression agents in pressurized storage cylinders associated with clean agent fire suppression systems**.** The conventional clean agents today are FM-200 and 3M Novec 1230, and in compliance with standard practice, the amount of agent in a storage cylinder must be regularly measured to ensure that the fire suppression system can provide adequate protection in the event of a fire. The current technology employed to measure fire suppression agents in storage cylinders features a Dip Tape design that is associated with inaccuracies because it is cumbersome to use and prone to human-operator errors. A review of literature was first undertaken for the following purposes: (i) to verify the relevant standards in clean agent fire suppression systems, (ii) to clarify the types of sensors available for the measurement of liquid levels, and (iii) to review relevant patent literature to determine the state-of-the-art in liquid level measurement, particularly with respect to fire suppression systems. A new design—which leverages the temperature and weight of the liquid agent to calculate the volume and mass of agent in a storage cylinder—was developed. The new design features a Time-of-Flight (TOF) sensor, a Texas Instruments LM35 temperature sensor, a Human Machine Interface (HMI), non-volatile system memory to store settings and data, and an ATMEGA328P microcontroller. System hardware and software programming was performed with Arduino. A POC system was successfully developed in five phases. In Phase One, components were selected, and a prototype unit was assembled. In Phase Two, the Arduino microcontroller firmware was developed, enabling the system to display the TOF sensor data and temperature data. Phase Three entailed the development of an algorithm for converting the TOF and temperature data into a volume and weight. Phase Four saw the development of additional firmware features (i.e., a pushbutton), and Phase Five was devoted to the challenging development of the user interface. The report features detailed explanations of each of these phases, including development and strategies associated with firmware and other source code. Following the development phases, accuracy and repeatability testing and verification of sensor data, and verification of the firmware were undertaken. In place of FM-200 and 3M Novec 1230, water—which has similar chemical characteristics—was employed. The project demonstrates that the new TOF system design can accurately determine the agent weight and is therefore a promising LLI alternative technology for clean agent fire suppression systems. However, testing and verification revealed an inconsistency in sensor data associated with temperature, along with the realization that the LM35 temperature sensor needs to be replaced with a more accurate sensor. It was additionally determined that the current hardware memory size is likely inadequate to support significant data recording. Firmware testing, moreover, revealed two software bugs, one associated with serial communications and one associated with data recording. Further testing and development associated with these issues—along with the need to verify the system with FM-200 and 3M Novec 1230—is recommended.

## Acknowledgments

I would like to express my deepest appreciation to my advisor Dr. Cory Prust and committee members Gary Shimek and Dr. Subha Kumpaty.  I also could not have undertaken this journey without the full support and understanding of my spouse Michelle and daughter Scarlett.

## Table of Contents

Appendix F:  ROI Center-Point Calibration Data ............................................ 146

Appendix G:  LLI Tube Assembly Drawing .................................................... 147

Appendix H:  LLI External Float Assembly ..................................................... 148

Appendix I:  LLI Internal Float Assembly ...................................................... 149

## List of Figures

## List of Tables

## Nomenclature

*Abbreviations*

**ATMEGA 328P** – Atmel Atmega328P 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash produced by the Atmel Corporation

**BUS** – A distinct set of conductors carrying data and control signals within a computer system, to which pieces of equipment may be connected in parallel.

**C –** Centigrade or Celsius

**EEPROM** – Electrically Erasable Programmable Read-Only Memory

**ESPEC** – ESPEC North America, Inc.

**FM** – Factory Mutual

**GUI** – Graphical User Interface

**HMI** – Human Machine Interface

**IDE** – Integrated Development Environment

**ISR** – Interrupt Service Routine

**LCD –** Liquid Crystal Display

**LLI –** Liquid Level Indicator

**LM35** – Family of Texas Instruments Analog Temperature Sensors

**LM35DZ** – Texas Instruments Analog Temperature Sensor

**mm** – Millimeters

**MSE –** Master of Science in Engineering

**MSOE --** Milwaukee School of Engineering

**NFPA** – National Fire Protection Association

**OSHA --** The Occupational Safety and Health Administration

**PC** – Personal Computer

**POC** – Proof of Concept

**RGB** – Red Green Blue

**ROI** – Region of Interest

**SI** – International System of Units

**TLX** – TLX Technologies

**TOF** – Time of Flight

**UART --** Universal Asynchronous Receiver / Transmitter

**UL** – Underwriters Laboratories Inc.

**USB --** Universal Serial BUS

**VL53L1X --** ST Microelectronics Flight Sense Series Time of Flight Sensor

## Introduction

The purpose of this report is to describe, explain and discuss the results associated with the author's Milwaukee School of Engineering (MSOE) Master of Science in Engineering (MSE) capstone project. The purpose of the capstone project was to design a Time-of-Flight (TOF) sensor-based liquid level indicator/sensor (LLI or LLS) to measure fire suppression agents in pressurized storage cylinders associated with clean agent fire suppression systems.

This project is a future development project for TLX Technologies [1]. The development team is made up of Patrick Schwobe (Mechanical Engineer), Brett Berger (Drafter), and Ian Stumpe (Electrical/Test Engineer and author). The author's area of responsibility includes the design of the sensor and control circuit, the development of the control and user interface software, and testing of the device. Other aspects of the project such as float, tube assembly and enclosure design are addressed as they relate to the author's area of responsibility.

TLX Technologies [1] is a custom solenoid and valve manufacturer, and one of the leading suppliers of fire suppression actuators to fire suppression system manufacturers [1]. Through TLX's reputation in the fire suppression actuator market, a fire suppression system manufacturer contacted TLX to develop a LLI to replace the existing Dip Tape LLI (see Figure 1) used in their systems. This project offers TLX Technologies an opportunity to not only diversify, but also to expand its business because of a significant market for LLIs in the fire protection industry. The market demand just for that single manufacturer was 10,000 units per year, representing growth in sales of over $500,000 per TLX Technologies [1].

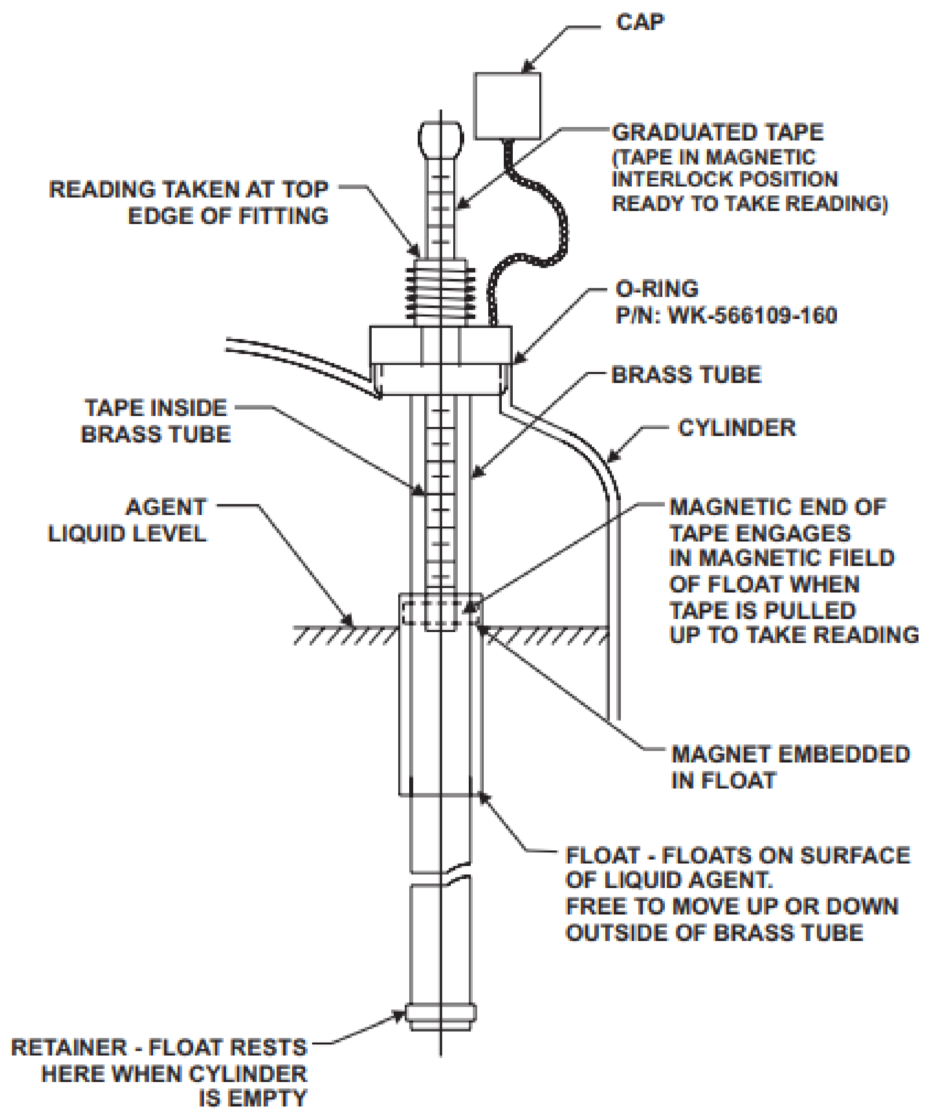

CAP
GRADUATED TAPE (TAPE IN MAGNETIC INTERLOCK POSITION READY TO TAKE READING)
READING TAKEN AT TOP EDGE OF FITTING
O-RING P/N: WK-566109-160
BRASS TUBE
CYLINDER
TAPE INSIDE BRASS TUBE
AGENT LIQUID LEVEL
MAGNETIC END OF TAPE ENGAGES IN MAGNETIC FIELD OF FLOAT WHEN TAPE IS PULLED UP TO TAKE READING
MAGNET EMBEDDED IN FLOAT
FLOAT - FLOATS ON SURFACE OF LIQUID AGENT. FREE TO MOVE UP OR DOWN OUTSIDE OF BRASS TUBE
RETAINER - FLOAT RESTS HERE WHEN CYLINDER IS EMPTY

**Figure 1: Dip Tape Liquid Level Indicator [2].**

Beyond the business case, there is a need to replace the current Dip Tape LLI due to multiple deficiencies in its design. Firstly, the device relies on a human operator to make the distance measurement, by reading the measurement from the graduated tape [3]. Thus, the accuracy of the device can be significantly affected by a Parallax Error introduced by the operator. Parallax Error is the apparent shift in an object’s position as it is viewed from different angles, so if the operator’s eye is not directly above the measuring tape, significant error could be introduced into the measurement [4]. Not only is the accuracy of the measurement affected by the operator, but also the repeatability of the measurement. The repeatability of the measurement is affected by the operator due to the need for the operator to manually pull the graduated tape out of the tube until the operator feels the magnetic pull between the magnet on the graduated tape and the magnet embedded float as shown in Figure 1 [3]. Because of these issues, it makes it almost impossible to detect a 5% loss of agent in the cylinder, which is a National Fire Protection Association (NFPA) requirement [5]. Additionally, the device forces the operator to determine what the ambient temperature of the fire suppression agent is because no onboard temperature sensor is configured on the device. This is an issue because the density of the agent is highly dependent on temperature. Therefore, depending on where the operator measures the ambient temperature, the result may not match the actual temperature of the agent. This will cause the operator to use the wrong temperature curve on the manufacturer-supplied look-up table (Figure 2), giving the operator the wrong weight measurement. This in turn could lead to the operator replacing an otherwise good tank. On the other hand, these issues could cause the operator to miss a faulty tank, leading to a fire suppression system that no longer has enough agent to extinguish a fire, which is a significant safety issue.

Another deficiency of the Dip Tape LLI is that taking a measurement is time consuming. The operator needs to remove the protective cover that covers the tape, take the measurement, find the lookup table (see Figure 2) that corresponds to that specific tank size for that specific system and use the correct curve (curves are based on agent temperature) to determine the weight of the agent [6]. In addition, there are usually only three different temperature curves on many of the system manufacturer-provided lookup tables, so if the current ambient temperature is not one of those curves, the operator must make an educated guess as to what the actual weight is.

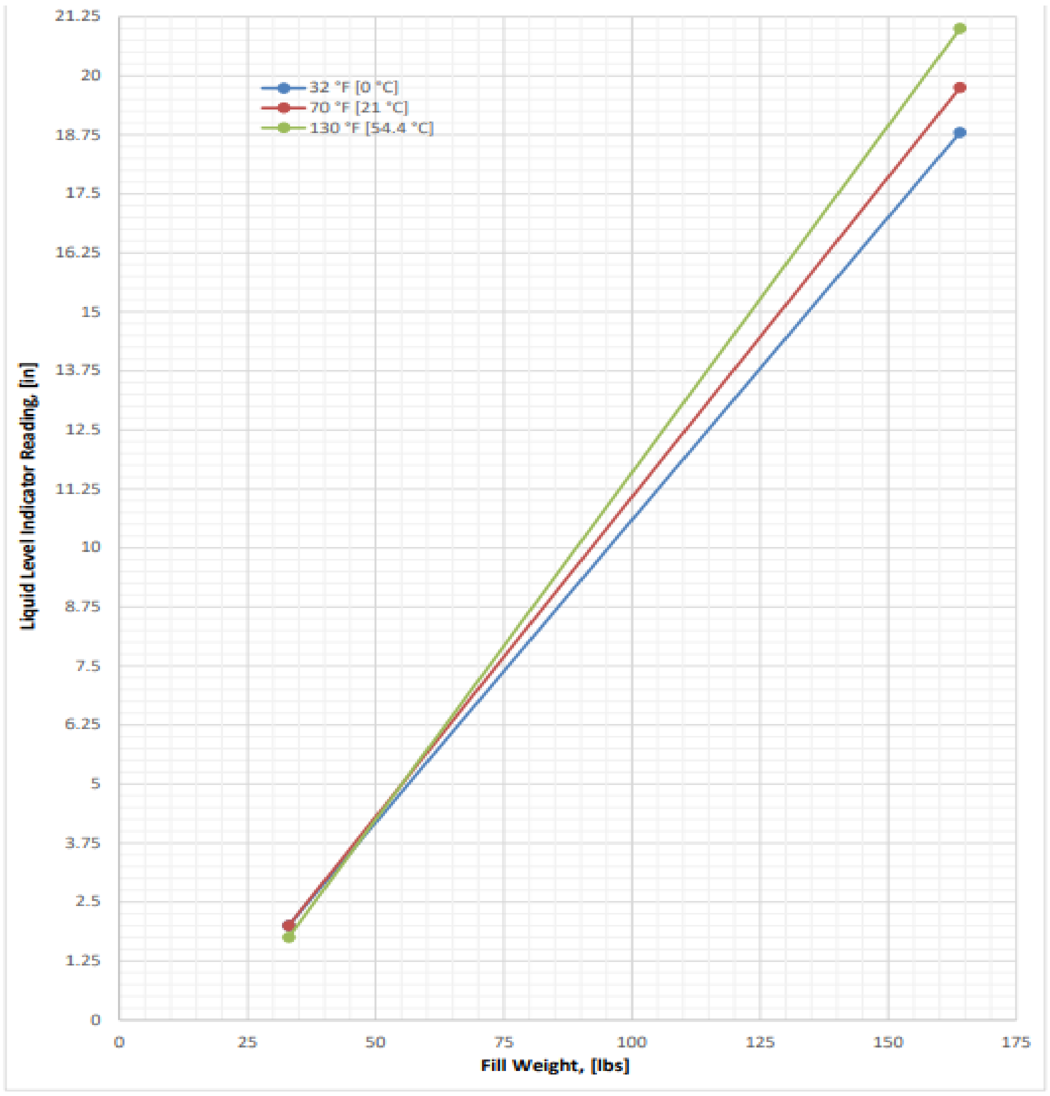


**Figure 2: Sample Lookup Table that Converts Liquid Level to Weight [7].**

Because of the deficiencies with the Dip Tape LLI and no market available alternative, a new device needed to be designed. This new design replaces the graduated tape with a Time-of-Flight (TOF) sensor and an internal magnet embedded float. The internal float interacts with the external magnet embedded float, allowing the TOF sensor to measure the height of the agent as shown in Figure 3. TOF sensors are an optical-based technology that transmit a cone-shaped beam of photons from an emitter and measure the amount of time taken for those photons to bounce back to the receiver [8, 9, 10].

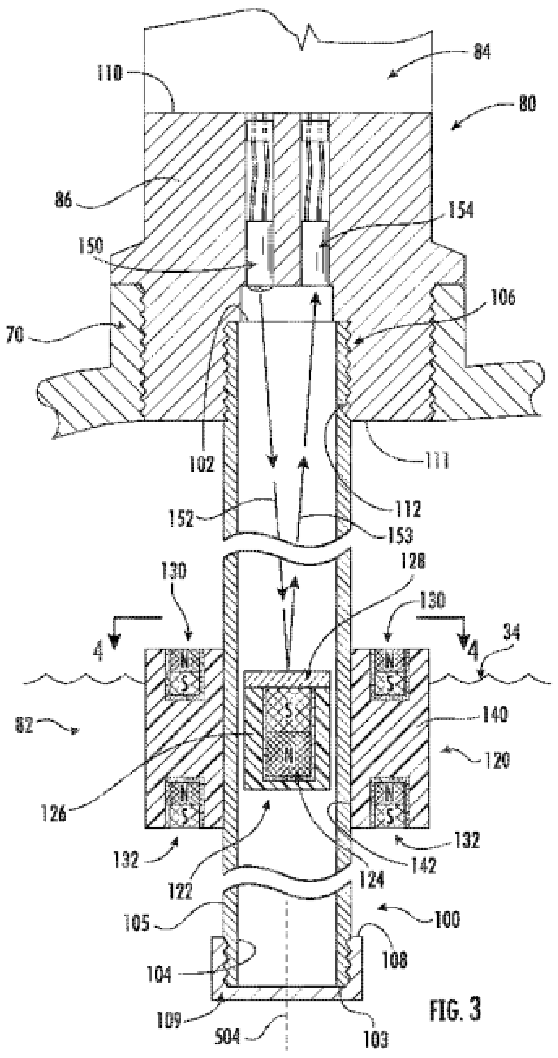


**Figure 3: Magnetic Trap Suppression Tank Level Sensor Patent WO2020/112241A1 [11].**

Additionally, the new design incorporates a temperature sensor so that the device can automatically calculate the weight of the agent. This design removes the need for the operator to use look-up tables (Figure 2). Furthermore, this new device incorporates an

LCD display, a human machine interface, time-of-day clock, push button and internal non-volatile memory. The components of the new LLI device are shown in a block diagram in Figure 15. These components will be discussed in greater detail in the Methods section of this report.

## Background

### Clean Agent Fire Suppression Systems

Clean Agent Fire Suppression Systems are fire suppression systems that use an electrically non-conductive, volatile, or gaseous agent that does not leave a residue upon evaporation [12]. There are multiple clean agents used in this class of fire suppression system. For this project, FM-200 and 3M Novec 1230 systems are the focus because both agents are compressed liquids [12]. Clean Agent Fire Suppression Systems have been in use since 1994 because of the phasing out of halogenated agents due to the detrimental effect of halons on the environment [13, 14]. Because Clean Agent Fire Suppression Systems are a direct replacement for Halon Clean Agent systems, they are typically used in high value buildings, such as telecommunication, data centers, process control rooms, medical facilities, museums, and libraries. The systems themselves are specifically tailored to whatever they are protecting so no system is the same. They are all made up of the same components, as shown in Figure 4, but the number and layout of these components are dependent on the application. This is especially true when it comes to the number of Agent Storage Containers, where the number of tanks can vary depending on the size of the protected area. This affects the number of LLIs in the system (each tank will have an LLI). As a matter of fact, there is a field of engineering devoted to fire protection and the design of these systems. A good reference source for designing these systems is the *SFPE Handbook of Fire Protection Engineering* [15].

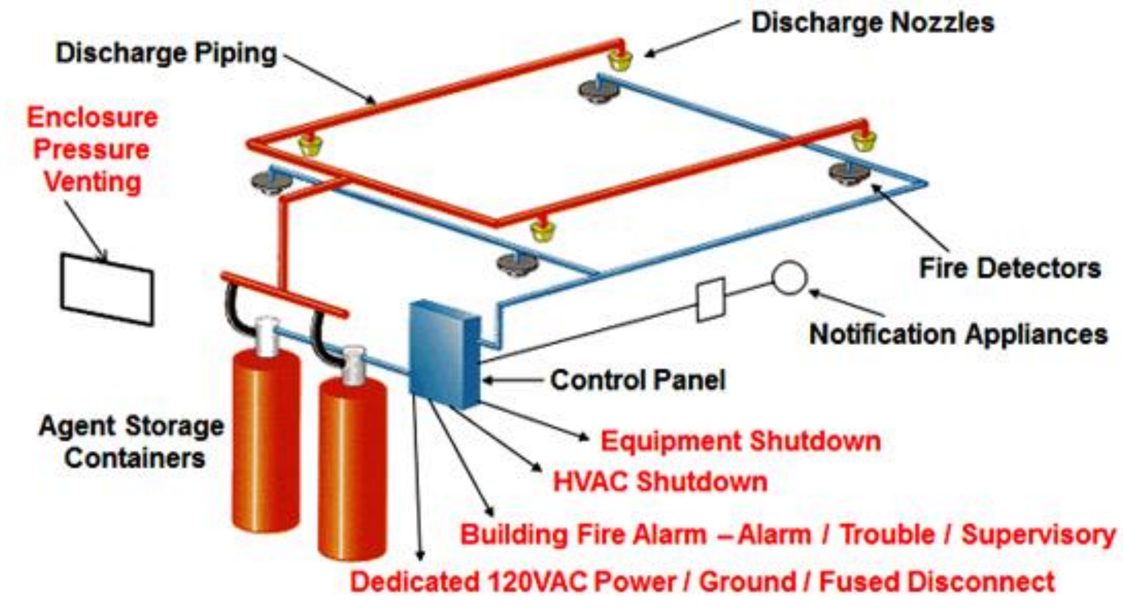


**Figure 4: Clean Agent Fire Suppression System Equipment [16].**

For Clean Agent Fire Suppression Systems, there are several industrial safety regulations and compliance agencies. These agencies are established by federal governments and by private organizations. An example of a government-based agency is the United States Department of Labor's Occupational Safety and Health Administration (OSHA). Examples of private organizations include UL Solutions (UL), the National Fire Protection Association (NFPA), and Factory Mutual (FM). So, what is the difference? Typically, the adherence to government-based agency codes, standards, and requirements is required by law, whereas the adherence to private organizations is voluntary. However, in some cases, those government agencies will adopt private organizations' codes, standards, and requirements instead of creating their own. For example, OSHA has adopted several codes from the NFPA. Another difference between the agencies is the agency’s scope. For instance, OSHA covers all the safety requirements that a company must follow to keep their workers safe, whereas the NFPA sets the standards for the fire protection related systems that those individual companies will install in their facilities [17]. In the case of UL and FM, these agencies cover the requirements for the individual components that make up those systems.

The standards that pertain to the TOF sensor based LLI are NPFA 2001, FM 5600, and UL 2166. NFPA standard 2001 requires that if an agent storage cylinder shows a loss in agent quantity (weight) of more than 5%, the cylinder must be refilled or replaced [5]. Standards FM 5600 and UL 2166 point to the NFPA 2001 standard and require that LLIs as they pertain to fire protection systems must accurately indicate the quantity of clean agent stored in an agent storage cylinder to within a tolerance of ±2.5 percent [18, 19]. In addition to those requirements, UL standard UL 864 Control Units and Accessories for Fire Alarm System also applies due to the device being an accessory for fire suppression systems [20]. Per the UL 864 standard, all accessories that are in outdoor wet locations are tested at -40°C to 66°C. This standard was chosen because it is the most stringent regarding operating temperature, and from an application point of view, it is unknown if the device will be kept in an outdoor wet location. Using these standards and operational requirements for FM-200 and 3M Novec 1230 Clean Agent Fire Suppression Systems, the device requirements shown in Table 1 were chosen. Table 1 summarizes the proof of concept – or evidence – demonstrating that the project device is feasible.

**Table 1: LLI Proof of Concept Requirements.**

| Liquid Level Indicator Requirements | | | | | |
|---|---|---|---|---|---|
| Parameter | Min Value | Nominal Value | Max Value | Units | Tolerance |
| Weight of Agent Accuracy | | | | % | ±2.5 |
| Operating Pressure | | 25 (363) or 35 (508) | 50 (725) | bar (psi) | +35 (508) |
| Burst Pressure | | 105 (1523) | | bar (psi) | N/A |
| Measurement Range (Distance) | 0 | | 2032 (80) | mm(in.) | |
| Operating Temperature | -40 (40) | | 66 (151) | °C(°F) | ±1 (±2) |
| External Float Density | < 1 (0.58) | | | g/ml (oz/in^3) | |

The existing LLI technology that is used in Clean Agent Fire Suppression systems is very limited. The only type that is currently in use is the Dip Tape LLI pictured in Figure 1. However, there are several different liquid level sensing products available on the market. These products utilize a range of different technologies, but they all fall into four different categories, including single point level, continuous level, multi-point level and visual level indicators [21]. The current Dip Tape LLI belongs to the visual level category, and it is the only type the author could find on the market to be certified for use in Clean Agent Fire Suppression system cylinders. The author did find others that are certified for use in water-based fire suppression sprinkler systems. Figure 5 shows the variety of LLIs that are currently available. All the different LLI technologies that are shown in Figure 5 are discussed in further detail by Hunt [22].

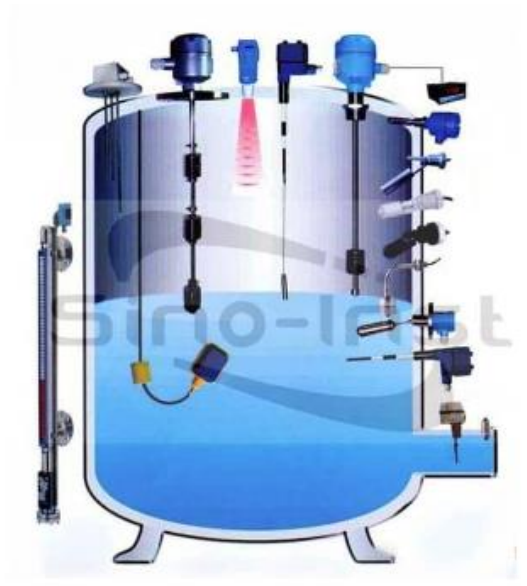


**Figure 5: Available Liquid Level Sensors [23].**

The following is a brief description of Capacitive, Ultrasonic and Optical based sensors, based on Hunt [22].

Capacitive sensors can measure point or continuous levels. The device acts like a capacitor with the liquid as the dielectric, where the output capacitive/resistive values are conditioned and converted to analog signals. Hunt [22] indicates that capacitive sensors are suitable for use with liquids, pastes, and some solids.

Ultrasonic sensors are used for measuring liquid level, and are split into two categories, contact and non-contact [22, 23]. Contact type sensors are typically used in pipes or vessels to operate pumps, solenoid valves, and high/low alarms, and to measure liquid levels at a certain point. The ultrasonic signal passes across a small gap filled by the target liquid. If this level falls below a set level, the signal attenuates and switches a relay. In the case of the non-contact type ultrasonic sensor, the echo from the ultrasonic beam reflects off the liquid surface and returns to the sensor where it measures the time it took for the sound reflections to return, converting it to distance [22, 23].

Optical liquid level sensors like the one in this capstone project device use visible, infrared, or laser light [22]. They measure the liquid level the same way the non-contact ultrasonic sensor does. The sensor transmits the light and measures the time it takes for photons to reflect off the surface of the liquid. These sensors can also be contact or non-contact point type sensors [22].

**Time-of-Flight Sensors**

Time-of-Flight (TOF) sensors are optical sensors that use the time-of-flight principle to determine the distance to an object. The time-of-flight principle is a method for measuring the distance between a sensor and an object, based on the time difference between the emission of a signal (in this case infrared light) and its return to the sensor, after being reflected by an object [10]. These sensors fall into two different categories, direct and indirect. The direct type sends out short pulses of light that last a few nanoseconds and then measure the time it takes for some of the emitted light to bounce back. The indirect type sends out a continuous, modulated light and measures the phase of the reflected light to calculate the distance to an object. Both direct and indirect TOF sensors are used in a range of applications, including robot navigation, vehicle monitoring, people counting, and object detection. In the case of object detection, these sensors not only detect solid objects, but they can also detect liquids such as water. Examples of these sensors being used to detect the level of liquids would be in coffee machines, soap dispensers, and smart toilets [8]. A specific example that utilizes the pulsed time-of-flight principle is the high-accuracy liquid level meter prototype created by Matta [24]. This liquid level gauge was developed for measuring liquid level accurately at distances up to 30 meters. The system consists of an optomechanical sensor head and electronics unit connected via two optical fibers [24]. Another example is the optical system that was developed to detect the volume of a liquid medical sample in labeled test tubes Liu [25].

## Review of Literature

There are several patents that use these different LLI technologies for different applications -- some for the fire protection market and others for applications outside of the fire protection market. The patents shown in Table 2 feature a mixture of applications, both fire protection and others. These patents and their applications are discussed over the next few paragraphs, starting with the non-fire protection applications, then moving on to cover the fire protection applications. It is additionally considered how they apply or do not apply to the new LLI design featured in this capstone project.

**Table 2: List of Applicable Patents.**

| Paten Number | Inventor |
|---|---|
| US5585786A | Clark Reece R *et al.* [26] |
| WO82/04316A1 | Fraser Gordon Bryce [27] |
| KR20150004237U | Lee, Kyung Mi [28] |
| CN105486382A | Chen Jianwen *et al.* [29] |
| EP3722757A1 | Chan, Eric Y et *al.* [30] |
| CN206772394U | Qu Fuping [31] |
| DE202016006955U1 | Liverani Maurizio [32] |
| WO2020112241A1 | Piech Marcin *et al.* [33] |
| WO2020/112218A1 | Piech Marcin *et al.* [11] |

The first patent to be discussed is patent number US5585786, “Optical Tank-Level Gauge”, invented by R.R. Clark and G.C. Barmore, Jr, shown in Figure 6 [26]. The application of this specific device is for use in large tanks such as barge tanks or railway car tanks.

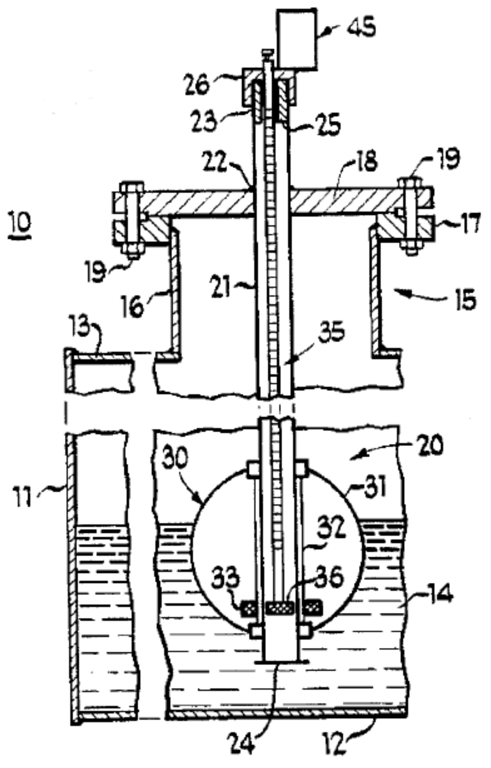


**Figure 6: Optical Tank-Level Gauge Patent US5585786 [26].**

The invention consists of an optical read head (45), a cylindrical gauging rod (35) that has alternating reflective and non-reflective stripes covering the length of the rod connected to a cylindrical magnet (36), and float magnet (33), which moves up and down with the level of the liquid. As the liquid level decreases or increases, the float magnet moves with the liquid level. Because the float magnet (33) is magnetically coupled with the cylindrical magnet (36), the gauging rod (35) moves up or down with the liquid level. As the gauging rod moves, the optical read head determines the liquid level using a quadrature encoder. The quadrature encoder provides the ability to count transitions on a pair of digital signals. The signals are positioned 90° out-of-phase, which allows for the detection of direction and relative position. In conjunction, with a third index signal, an absolute position can be established [34].

The “Fluid Level Indicator” patent number WO82/04316A1, invented by G.B. Fraser, uses laser beam interferometry to determine the fluid level [27]. Laser beam interferometry is a distance measurement technique done by splitting a laser beam into two, sending each of the two beams along different directions in space and then recombining the beams. If the beams travelled the same distance, the detector would see a recombined beam of the same brightness as the original beam, or nothing at all, depending on how the detector is set up [35]. The invention is shown in Figure 7.

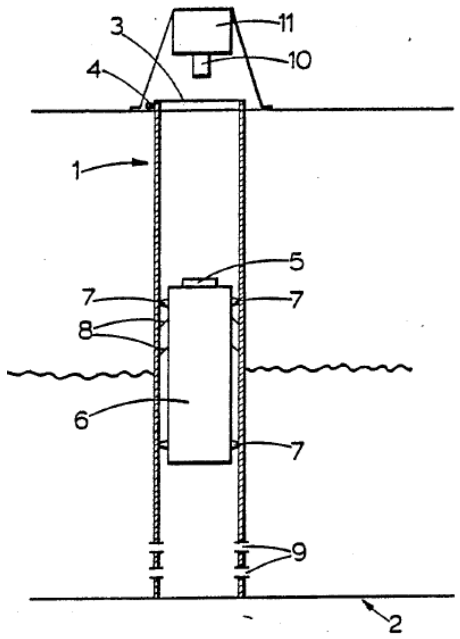


**Figure 7: Fluid Level Indicator Patent WO82/04316A1 [27].**

The invention consists of a tube (1), a reflector (5), float (6), laser (10), and an interferometric detector (11). The transmitted light from the laser (10) is reflected by the reflector (5), which is affixed to a float (6) that is at the same height as the liquid level. The reflected light is then captured by the interferometric detector (11) where electronic circuitry determines the position of the reflector (5). The application for this LLI is for use in large tanks such as those used on tanker trucks [27].

The next invention falls into the single point level type of LLI. It is for a "Water Level Sensing Apparatus," patent number KR20150004237U, invented by K.M. Lee [28]. This invention is used to control the flow of water into a water tank. As shown in Figure 8, the invention consists of a vertical pipe (110), linear hall sensor (120), a metal rod (130), a magnet embedded float (150, 140), and tank (200). As water flows in or out of the tank (200), the magnet embedded float (150, 140) rises or lowers with the water level. The magnetic field created by the embedded magnet of the float interacts with the metal rod (130), causing the magnetic flux to increase or decrease. This increase or decrease in the magnetic flux is measured by the linear hall sensor (120), which is then interpreted via a microcontroller to determine the height of the water. The controller will turn the water supply on or off depending on the water level.

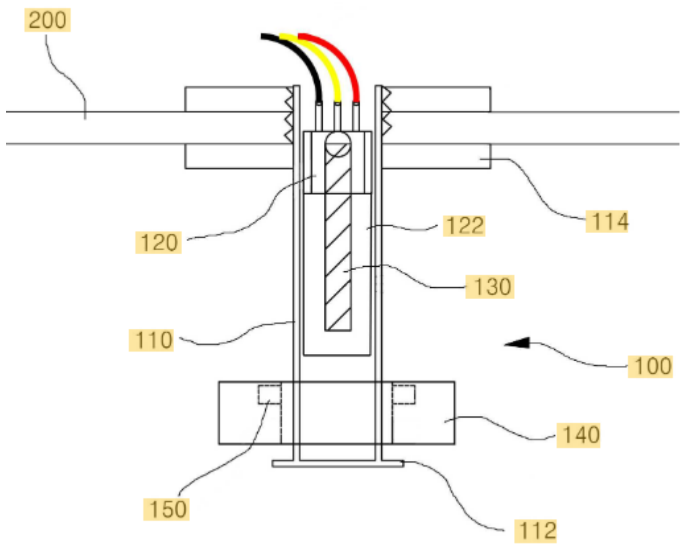


**Figure 8: Water Level Sensing Apparatus Patent KR20150004237U [28].**

Patent CN105486382A shown in Figure 9 was invented by Chen Jianwen [29]. The invention is a continuous level type LLI that uses a magneto-strictive sensor (2) to sense the position of the magnetic float (1). A magneto-strictive sensor is a wire or bar referred to as a waveguide that is typically made of an iron alloy. Short pulses of current are applied to the waveguide causing a magnetic field to be formed. This field and the magnetic field of the magnetic float interacts causing a torsional strain the Wiedemann Effect. This strain is then detected by a signal converter that can determine the exact position of the float and thus the liquid level [36].

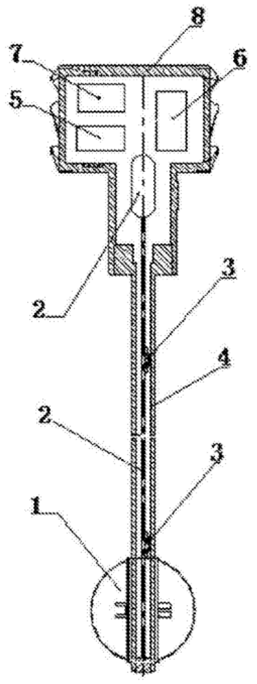


**Figure 9: Magneto-strictive Liquid Level Patent CN105486382A [29].**

As shown in Figure 10, patent EP3722757A1 invented by Eric Y Chan and Denis G Koshinz uses a type of TOF sensor to sense the level of liquid fuel in a fuel tank [30]. The Non-Contact Time-of-Flight Fuel Level Sensor Using Plastic Optical Fiber utilizes a non-contact plastic optical fiber (POF) (14) to optically sense the level of fuel (2) in a fuel tank (10). It further consists of a high-speed and high-power red laser diode (28), and an ultra-high-sensitivity photon-counting avalanche photodiode (32). The fuel level is sensed when the avalanche photodiode (32) first detects the light reflected by the POF end face (1) and then detects the light reflected by the fuel surface (3) in response to the emission of a laser pulse (18) by the red laser diode (28). A time delay detection circuit (30) calculates the time interval separating the respective times of arrival. This time interval is used by the fuel level calculator (34) to calculate the fuel level.

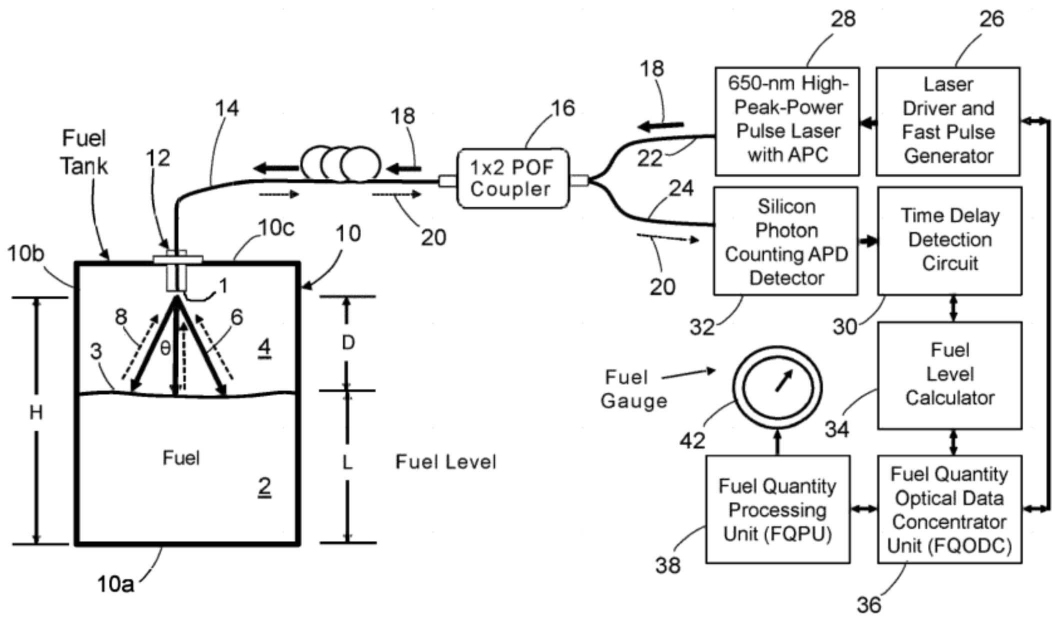

28
26
18
650-nm High-Peak-Power Pulse Laser with APC
Laser Driver and Fast Pulse Generator
14
18
16
Fuel Tank
12
1x2 POF Coupler
22
24
Silicon Photon Counting APD Detector
Time Delay Detection Circuit
10c
10
20
10b
20
1
8
6
3
θ
4
D
32
30
Fuel
Level
Calculator
Fuel Gauge
H
34
42
Fuel
2
L
Fuel Level
Fuel Quantity Processing Unit (FQPU)
Fuel Quantity Optical Data Concentrator Unit (FQODC)
38
10a
36

**Figure 10: Non-Contact Time-of-Flight Fuel Level Sensor using Plastic Optical Fiber Patent Number US5585786 [30].**

The next set of patents deal with inventions that are meant to be used in fire suppression systems. This first patent, shown in Figure 11, the Liquid Extinguisher Steel Cylinder LLI, was invented by Qu Fuping [31]. It is a single point level type of LLI using reed switch technology. As the magnet embedded float (6, 5) drops due to the liquid level decreasing caused by the loss of fire suppression agent, the magnetic field of the float interacts with the reed switch (4) causing, it to close. This action completes the alarm circuit (7), initiating an alarm, denoting that there is an issue with the agent storage tank.

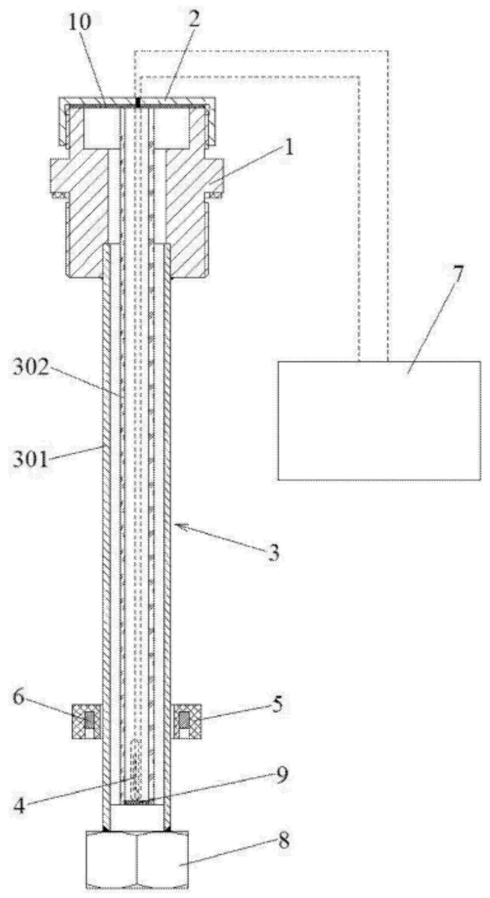


**Figure 11: Liquid Extinguisher Steel Cylinder LLI Patent CN206772394U [31].**

Patent DE202016006955U1, invented by Liverani Maurizio, was designed to be used in flammable liquid storage tanks [32]. This device, which is shown in Figure 12, is an example of a continuous level ultrasonic type LLI. In this invention, sound waves generated by the receiver/generator (10) spread along the conductor (11). Once the sound waves reach the float (18) and magnet (19), which are at the height of the liquid, they are reflected to the receiver/generator (10). Data from the receiver are then interpreted by electronic circuit (17) to determine the height of the liquid. Additionally, a temperature sensor that is at the bottom of tube (2) transmits fluid temperature data to the electronic circuit (17). Using the height and temperature data, the invention will set off an alarm based on pre-determined limits.

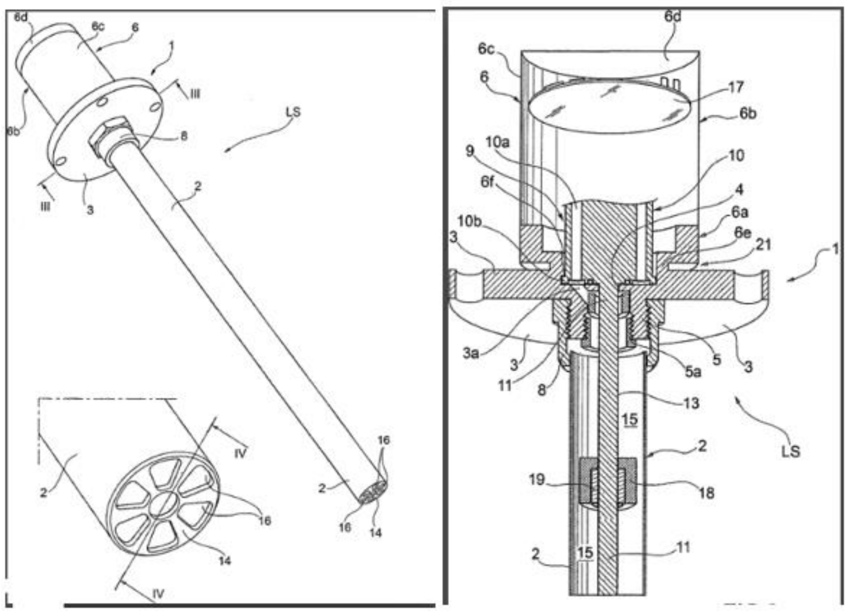


**Figure 12: Flammable Liquid LLI Patent DE202016006955U1 [32].**

The last two patents listed in Table 2, WO2020112241A1 and WO2020/112218A1, were invented by Piech *et al*. [11, 33]. The application for both inventions is for use in fire suppressant storage tanks and both are continuous level type LLIs. Additionally, both inventions are alternatives for the Dip Tape type LLI that is currently used. Unfortunately, it appears that neither are currently available on the open market. Even the patent owner, Carrier Corporation, does not advertise these LLIs.

Patent WO2020112241A1 shown in Figure 13 consists of multiple magnetic field sensors (152) arranged on a carrier (150) that measure the magnetic field of the magnet embedded float (120). An electronics module (166) interprets the magnetic field data from the magnetic field sensors (152) to determine the height of the liquid level. This design idea was a contender for the author's capstone project. However, after some further research, it was learned that the number of magnetic field sensors required to make the device accurate enough was cost prohibitive. Additionally, the coding required to correctly interpret the magnetic field data provided by the sensor would have been far too complicated.  So, the idea was scrapped.

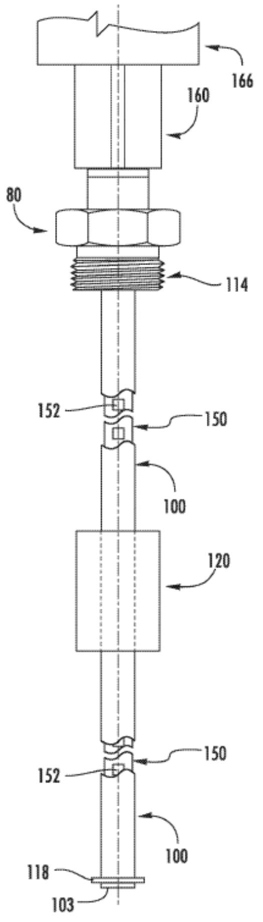


**Figure 13: Adaptable Suppression Tank Level Sensor Patent WO2020/112218A1 [33].**

That leads to the second patent by Piech *et al*. [11], WO2020/112241A1, the "Magnetic Trap Suppression Tank Level Sensor", shown in Figure 14.  The key design element in this invention is that it incorporates a two-float design, including an inner float (128) embedded with a magnet (124) that is coupled with an external float (120) that features embedded magnets on the top (130) and bottom (132) of the float.  With this trapped inner float design, several different linear displacement sensor technologies can be used to determine the height of the liquid.  This is pointed out in the claim section of the patent where it calls out several technologies. These technologies include magnetic field sensors (Hall Effect sensors) or magnetic switches (reed-switches), radar-based sensors, as well as resistance, ultrasonic, and optical sensors, such as the TOF sensor featured in this author's device [11]. Several searches using Google, Google Scholar, and Milwaukee School of Engineering library's Summon Discovery Service found no scholarly articles nor a market example of the patent's proposed device, which is similar to the author's design.

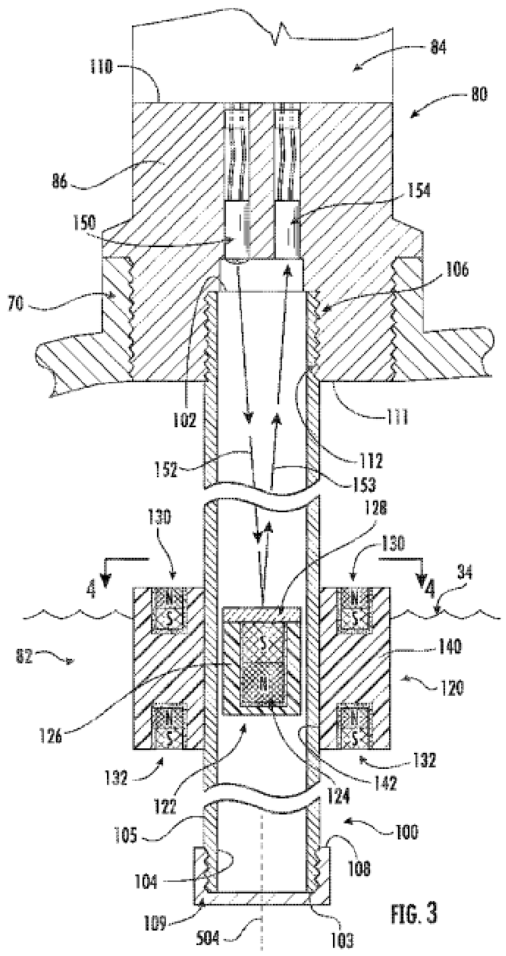


**Figure 14: Magnetic Trap Suppression Tank Level Sensor Patent WO2020/112241A1 [11].**

As discussed, there are several different LLI technologies that are available on the market. However, these commercially available technologies are not suitable for use in

Clean Agent Fire Suppression storage cylinders due to the system design requirements required by regulatory agencies such as OSHA, NFPA, FM and UL. Additionally, the cost of these technologies such, as the proposed design in patent WO2020/112218A1, prevents these technologies from being adopted for use in Clean Agent Fire Suppression storage cylinders. This review of literature further shows that even though there is a patent that outlines the general idea of the author's design, that design has not been made commercially available [11]. Additionally, that patent pertains to the configuration of the external float (uses two magnets to capture the inner float) and not to the technology used to detect the inner float [11].

## Methods

The purpose of this MSOE MSE capstone project is to design a new LLI to replace the Dip Tape LLIs used in Clean Agent Fire Suppression Systems. This new design replaces the graduated tape and magnet (see Figure 1) with a TOF sensor and magnet embedded inner float, as seen in Figure 14. To accomplish this goal, a microcontroller was added to interpret the distance data from the TOF sensor and to perform the calculations required to convert the distance measurement into a weight. To make these calculations, the microcontroller also needs to know the temperature of the agent because the densities of FM-200 and 3M-Novec 1230 are dependent on temperature, so a temperature sensor was added to the design. Additionally, the device needs a display to show the calculated weight measurement and a Human Machine Interface (HMI) so that important information could be entered into the device, such as the measurements of the storage tank, type of agent, and data capture settings. Because of the need for an HMI, additional non-volatile memory is necessary to store device settings. Adding the non-volatile memory would also allow the device to become a data recorder, allowing for all measurement data to be stored and analyzed if a failure in the tank occurs. However, to accommodate the data recording feature, a time-of-day clock is needed so that each data point will feature an accurate timestamp. A block diagram of this Proof-of-Concept (POC) design is shown in Figure 15.

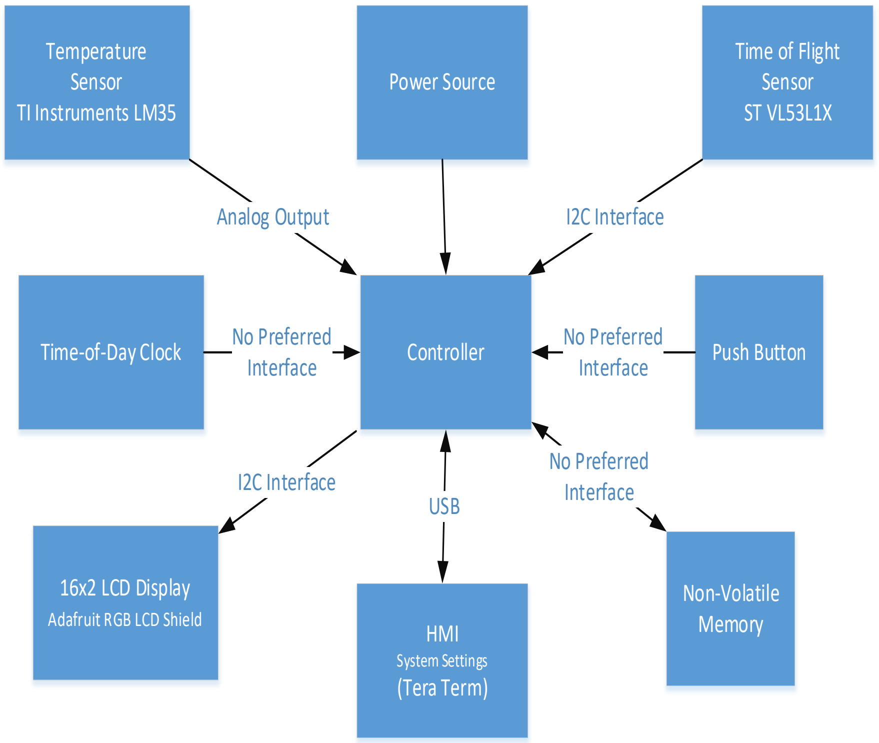


**Figure 15: Proof-of-Concept Sensor Design.**

## Phase One Development

Phase One of this project entailed the building of a proof-of-concept unit that could be used in verifying whether the selected TOF sensor would work in this application. The selected TOF sensor was the ST Microelectronics Flight Sense series VL53L1X. This sensor was selected for several reasons, including its resolution of ±1mm, its repeatability of ±0.15% to ±1%, its range of four meters, its programable range of interest, and its pre-developed software library [37]. Because of the sensor's pre-defined software library, several sources are available that offer VL53L1X sensor development boards. A sensor development board is a printed circuit board used for embedded system development, including a series of hardware components, such as central processing unit, memory, input device, output device, data path/bus, and external resource interface [38]. The VL53L1X development board that was selected for this project is the SparkFun Qwiic Distance Sensor (VL53L1X) Breakout. This device was selected because an Arduino Software Library is available [39] that would render coding much easier because of the author's prior experience with Arduino development boards. The development board's compatibility with Arduino also led to the selection of the Arduino Uno development board for the microcontroller (ATMEGA328P microcontroller) [40, 41]. For the temperature sensor, the Texas Instruments LM35 was selected because of its accuracy of ±1°C, its linearity, and its temperature range of -55°C to 150°C [42]. The last component that was required for this phase was a display device. For the display, the Adafruit Learning System RGB LCD Shield was selected. Like the sensor development board, this development board was selected because it features a developed Arduino library (Adafruit RGB LCD Shield) [43].

With the components selected, parts could be ordered and the first POC units could be built. Figures 16 through 18 show the assembled POC in its enclosure that was designed by Brett Berger. Figure 19 shows the assembled POC on a mock agent storage cylinder. The tube assembly that the POC sits on was designed by Patrick Schwobe, along with the external and internal float. Drawings of these are shown in Appendix G through I.

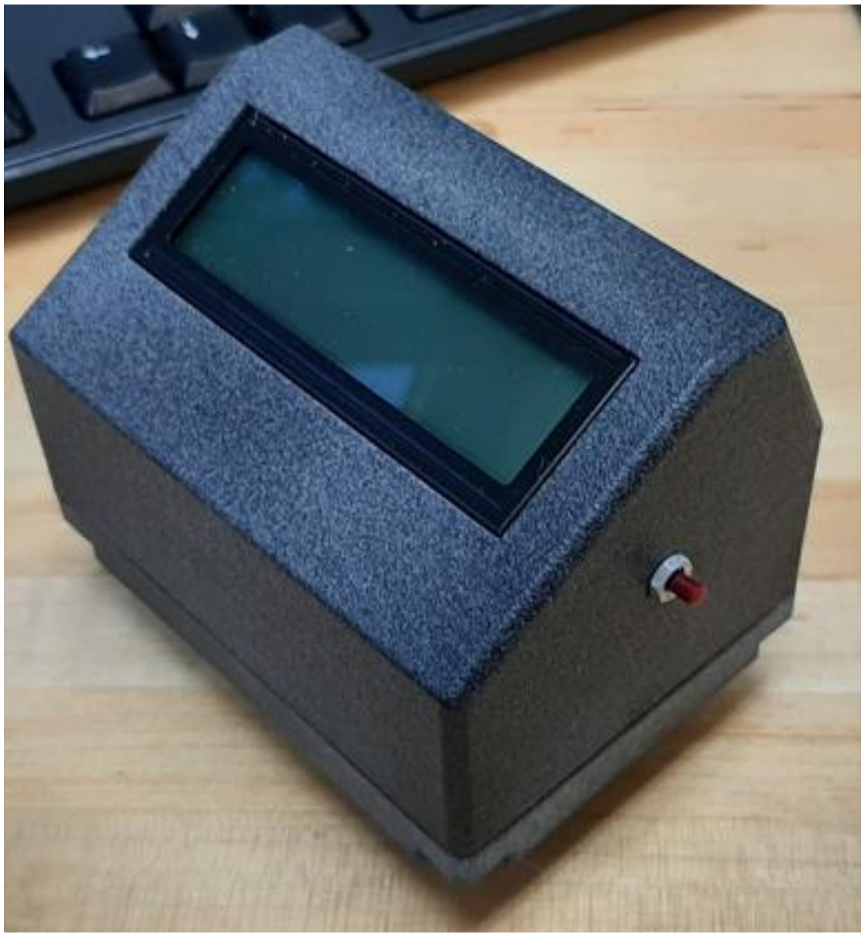


**Figure 16: TOF LLI Sensor Unit Side View.**

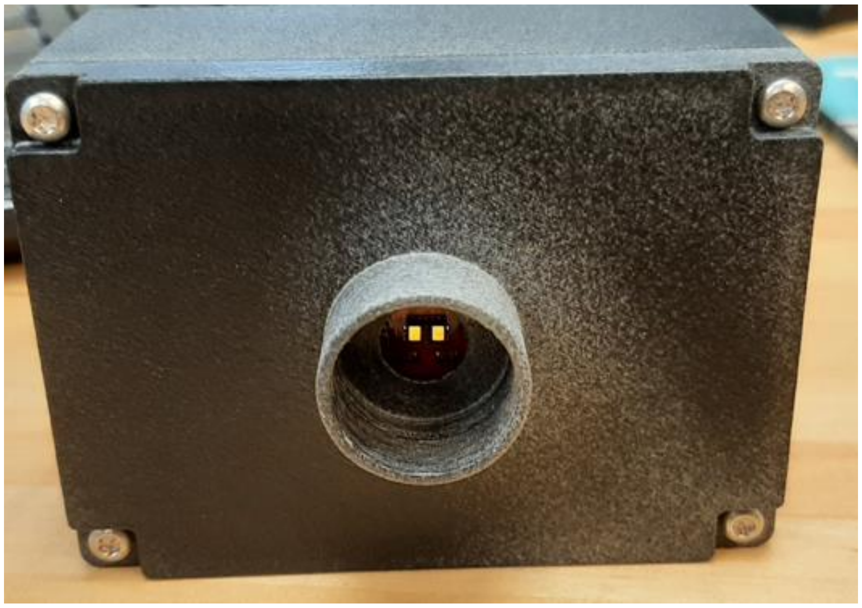


**Figure 17: TOF LLI Sensor Unit Bottom View Showing VL53L1X Sensor.**

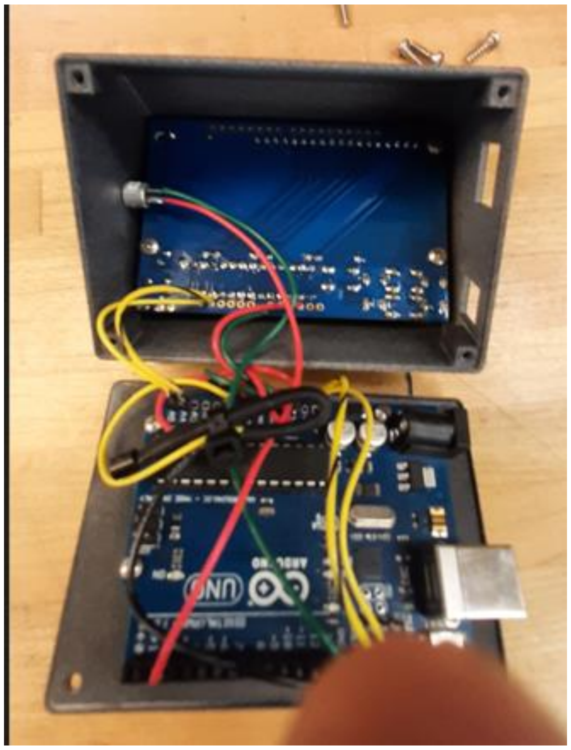


**Figure 18: TOF LLI Sensor Unit Internal View Showing Arduino UNO and LM35.**

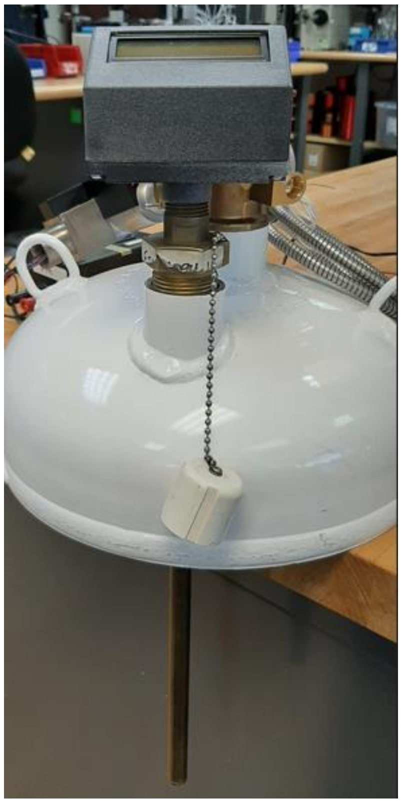
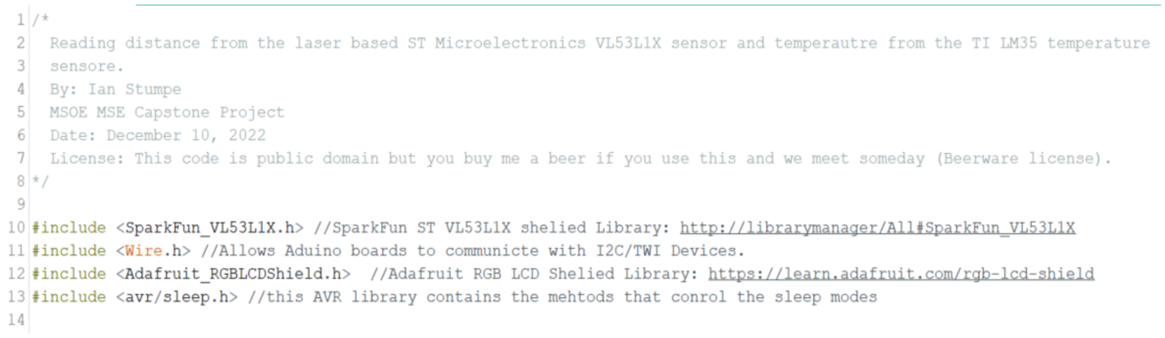


**Figure 19: TOF LLI Full Assembly.**

## Phase Two Development

With the POC assembled, the next phase entailed writing the software the Arduino UNO microcontroller needs to communicate with the TOF sensor and temperature sensor and display the distance and temperature data on the LCD. This revision of the code would serve three purposes: one, to demonstrate that the TOF sensor, temperature sensor and LCD function correctly; two, to verify the accuracy of the TOF; and three, to verify the accuracy of the temperature sensor. This revision of the code is shown in its entirety in Appendix A and is discussed in further detail in the following paragraphs.

There is one observation that needs to be noted about the code before it is discussed in further detail. That is that as the reader reviews the code in Appendix A, it will be seen that some lines of the code fall in between these /* */ and follow behind two forward slashes (//) as shown in Figure 20.

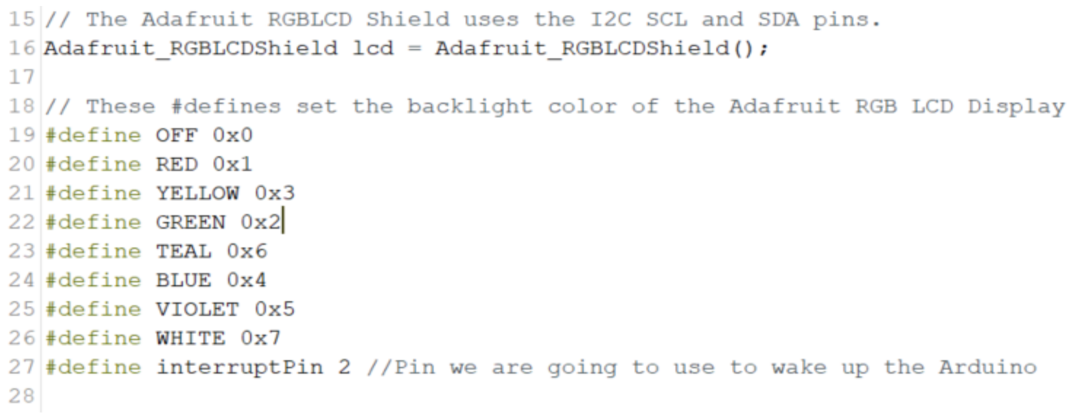


```
 1 /*
 2   Reading distance from the laser based ST Microelectronics VL53L1X sensor and temperautre from the TI LM35 temperature
 3   sensore.
 4   By: Ian Stumpe
 5   MSOE MSE Capstone Project
 6   Date: December 10, 2022
 7   License: This code is public domain but you buy me a beer if you use this and we meet someday (Beerware license).
 8 */
 9
10 #include <SparkFun_VL53L1X.h> //SparkFun ST VL53L1X shelied Library: http://librarymanager/All#SparkFun_VL53L1X
11 #include <Wire.h> //Allows Aduino boards to communicte with I2C/TWI Devices.
12 #include <Adafruit_RGBLCDShield.h>  //Adafruit RGB LCD Shelied Library: https://learn.adafruit.com/rgb-lcd-shield
13 #include <avr/sleep.h> //this AVR library contains the mehtods that conrol the sleep modes
14
```

**Figure 20: Source Code Section One.**

All words that follow the forward slashes or sit between /* */ are comments and are not part of the actual code. Their purpose is to explain what that line or section of code does. For example, the snippet of code shown in Figure 20, which is the first 13 lines of code that is shown in Appendix A, shows the header of the code lines 1 through 8, which lay between /**/. The point of the header is to give general information about the code. In this case, it gives a brief description about what the code does, who authored it, when it was created, and if there is any licensing or copyright information. Also shown in that figure, on lines 10 through 13, are the included software libraries. These libraries provide the background code and subroutines that allow the code to function properly. Without them, the code that follows would not function and would not be recognized by the Arduino's programming language compiler. The compiler is a special program that translates the programming language's source code (the text shown in Appendix A) into code that the microcontroller can understand. The included libraries are the "SparkFun_VL53L1X.h", which provides the functionality to interface with the SparkFun VL53L1X development board; "Wire.H", which provides the functionality required to communicate with $I^2C$ devices; "Adafruit_RGBLCDShield.h", which provides the functionality to interface/control the LCD display; and "avr/sleep.h", which provides the functionality to control the microcontroller's sleep modes.

Section Two of the code, which is shown in Figure 21, shows lines 15 through 28 of the code. The code shown on line 16 enables the microcontroller to communicate with the LCD display via its $I^2C$ interface.

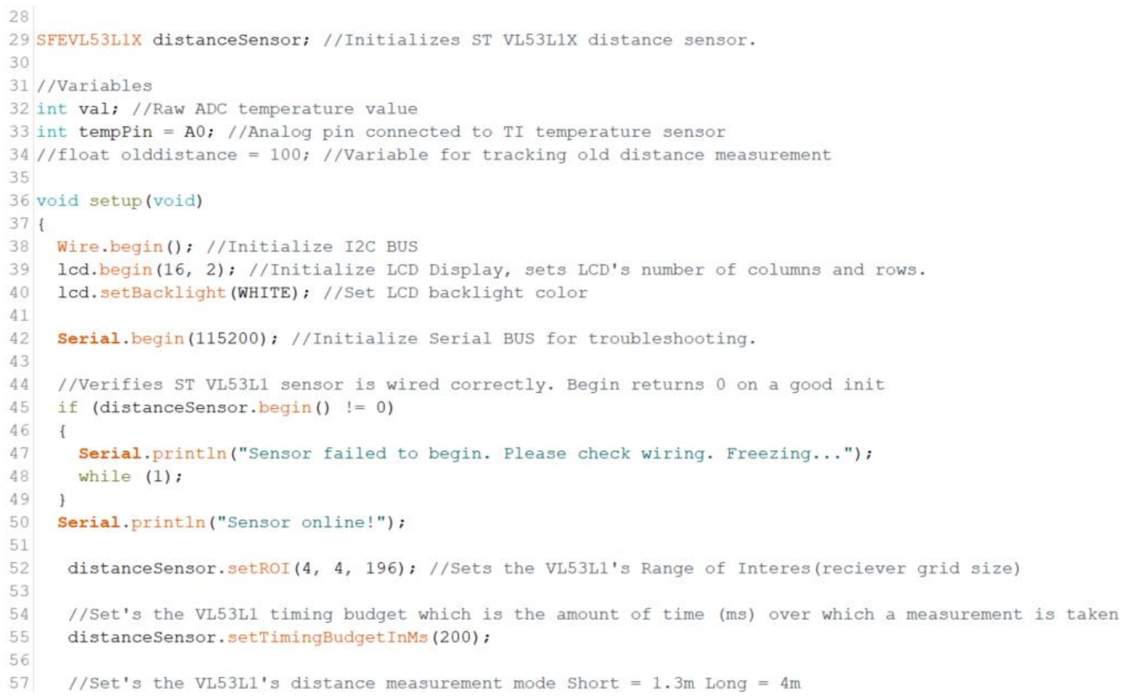


```
15 // The Adafruit RGBLCD Shield uses the I2C SCL and SDA pins.
16 Adafruit_RGBLCDShield lcd = Adafruit_RGBLCDShield();
17
18 // These #defines set the backlight color of the Adafruit RGB LCD Display
19 #define OFF 0x0
20 #define RED 0x1
21 #define YELLOW 0x3
22 #define GREEN 0x2
23 #define TEAL 0x6
24 #define BLUE 0x4
25 #define VIOLET 0x5
26 #define WHITE 0x7
27 #define interruptPin 2 //Pin we are going to use to wake up the Arduino
28
```

**Figure 21: Source Code Section Two.**

Lines 19 through 27 denote constant values and the names associated with that value. These are used later in the code to determine the color of the LCD backlight.

The next section of code shown in Figure 22 covers the initialization of the VL53L1X TOF sensor, non-constant variables, and the setup function. Like line 16 in the previous section, line 29 enables the microcontroller to communicate with the VL53L1X TOF sensor. Lines 32 and 33 establish the global variables “val” and “tempPin”; global variables are used by the microcontroller to store information used to by the program. The variable “val” is used to store the raw temperature value from the microcontroller’s analog to digital converter (ADC). The ADC converts the analog voltage that is measured on the “tempPin” (microcontroller’s analog input A0 pin) to a digital value between 0 and 1023. This analog voltage is the output from the LM35 temperature sensor. Next in this section of code is the “setup” function. In this function

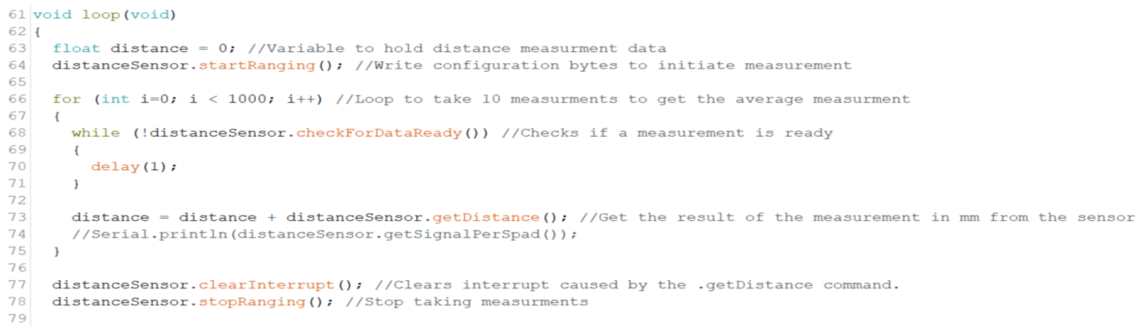


```
28
29 SFEVL53L1X distanceSensor; //Initializes ST VL53L1X distance sensor.
30
31 //Variables
32 int val; //Raw ADC temperature value
33 int tempPin = A0; //Analog pin connected to TI temperature sensor
34 //float olddistance = 100; //Variable for tracking old distance measurement
35
36 void setup(void)
37 {
38   Wire.begin(); //Initialize I2C BUS
39   lcd.begin(16, 2); //Initialize LCD Display, sets LCD's number of columns and rows.
40   lcd.setBacklight(WHITE); //Set LCD backlight color
41
42   Serial.begin(115200); //Initialize Serial BUS for troubleshooting.
43
44   //Verifies ST VL53L1 sensor is wired correctly. Begin returns 0 on a good init
45   if (distanceSensor.begin() != 0)
46   {
47     Serial.println("Sensor failed to begin. Please check wiring. Freezing...");
48     while (1);
49   }
50   Serial.println("Sensor online!");
51
52    distanceSensor.setROI(4, 4, 196); //Sets the VL53L1's Range of Interes(reciever grid size)
53
54    //Set's the VL53L1 timing budget which is the amount of time (ms) over which a measurement is taken
55    distanceSensor.setTimingBudgetInMs(200);
56
57    //Set's the VL53L1's distance measurement mode Short = 1.3m Long = 4m
```

**Figure 22: Source Code Section Three.**

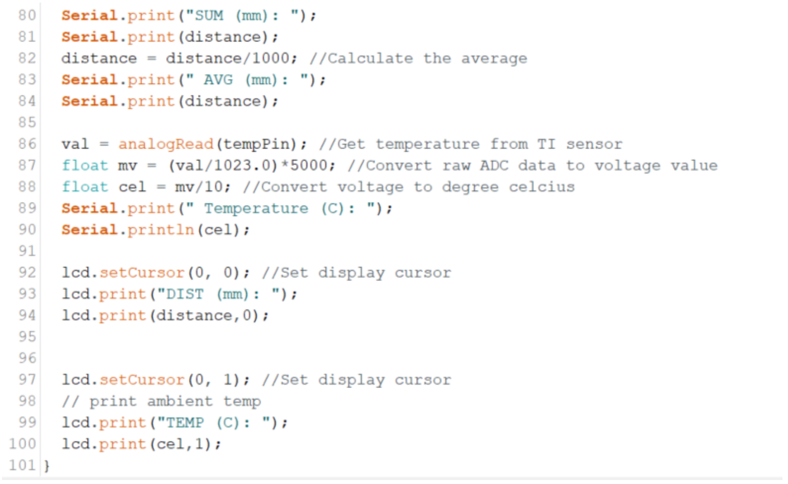
are the commands that set the basic settings for the hardware connected to the microcontroller. Line 38 initializes the microcontroller's $I^2C$ communication BUS so it can transmit commands and receive information from the TOF sensor and LCD display. Lines 39 and 40 turns the LCD display on, tells the microcontroller that it is a 16x2 (16 columns, 2 rows) display and sets the backlight of the display to white. This variable name should look familiar as it was defined in source code Section Two and represents the value for the color white, which is hexadecimal value of 7. The next line of code, line 42 initializes the microcontroller's serial BUS, which is another communication BUS like $I^2C$, but this communication BUS is used to communicate with the PC. It is mainly used by the programmer to see if the program is running correctly. For example, the line of code on line 47 and 50 tell the microcontroller to transmit information telling the programmer that it was either able to or not able to communicate with the TOF sensor. This leads to the last few lines of this section of code, which deal with verifying that the microcontroller can communicate with the TOF sensor and setting the sensor up. Line 45 of the code tells the microcontroller to verify that it can communicate with the TOF sensor. The lines of code shown on lines 52, 55 and 58 set the Region-of-Interest, Timing Budget, and the Distance Mode of the TOF sensor. These settings are discussed in greater detail in the following sections of this report.

The last two sections of this program, shown in Figures 23 and 24, show the "loop" function, which is where all the action takes place. The code in this function is continuously run by the microcontroller. It contains the code that tells the TOF when to take distance measurements, when to stop taking distance measurements, calculates the average of the distance measurement, takes ambient temperature readings, and transmits the distance and temperature data to the display.



```
void loop(void)
{
  float distance = 0; //Variable to hold distance measurment data
  distanceSensor.startRanging(); //Write configuration bytes to initiate measurement

  for (int i=0; i < 1000; i++) //Loop to take 10 measurments to get the average measurment
  {
    while (!distanceSensor.checkForDataReady()) //Checks if a measurement is ready
    {
      delay(1);
    }

    distance = distance + distanceSensor.getDistance(); //Get the result of the measurement in mm from the sensor
    //Serial.println(distanceSensor.getSignalPerSpad());
  }

  distanceSensor.clearInterrupt(); //Clears interrupt caused by the .getDistance command.
  distanceSensor.stopRanging(); //Stop taking measurments

```

**Figure 23: Source Code Section Four A.**



```
  Serial.print("SUM (mm): ");
  Serial.print(distance);
  distance = distance/1000; //Calculate the average
  Serial.print(" AVG (mm): ");
  Serial.print(distance);

  val = analogRead(tempPin); //Get temperature from TI sensor
  float mv = (val/1023.0)*5000; //Convert raw ADC data to voltage value
  float cel = mv/10; //Convert voltage to degree celcius
  Serial.print(" Temperature (C): ");
  Serial.println(cel);

  lcd.setCursor(0, 0); //Set display cursor
  lcd.print("DIST (mm): ");
  lcd.print(distance,0);


  lcd.setCursor(0, 1); //Set display cursor
  // print ambient temp
  lcd.print("TEMP (C): ");
  lcd.print(cel,1);
}
```

**Figure 24:  Source Code Section Four B.**

Before the accuracy of the TOF could be tested, the author needed to verify the optimum settings for the VL53L1X TOF sensor that would allow for the most accurate measurements.  These settings are associated with the sensor’s Region-of-Interest (ROI), Timing Budget, Inter-Measurement Period, and Distance Mode.  Of these settings, the most influential is the ROI.  The ROI determines the size and position of the receiving array, thus allowing the Field-of-View to be reduced from 27° to 15°, as shown in Figure 25.

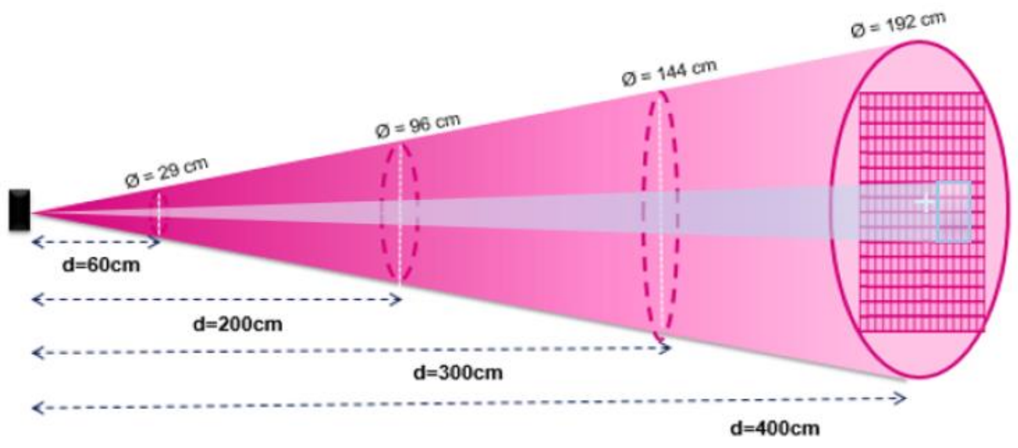


**Figure 25: VL53L1X Field-of-View at 27° (Pink) and 15° (Blue) [44].**

This receiving array is made up of 256 Single Photon Avalanche Diodes (SPAD) that form a 16 x 16 grid, which is shown in Figure 26. However, the size of the array is programable all the way down to a 4 x 4 grid. Because of this feature, the location of this ROI is also programable. Because the sensor needs to detect a float located in a 0.436-inch diameter tube, the size of the ROI was set to a 4 x 4 grid. However, the optimal location of the grid was an unknown.

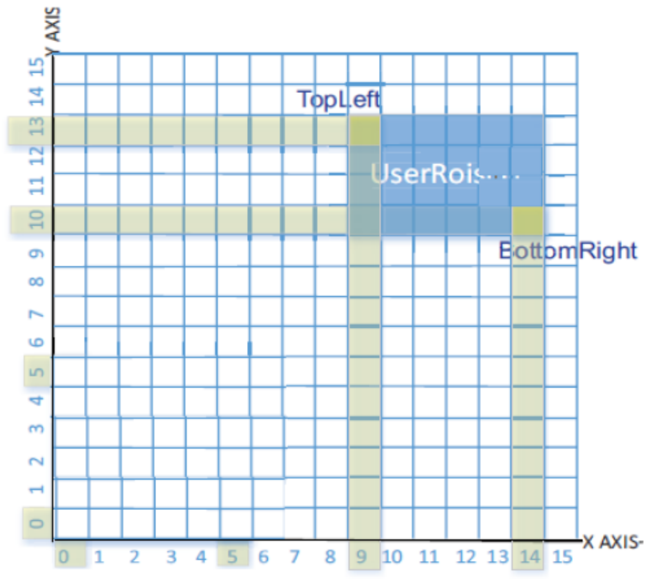
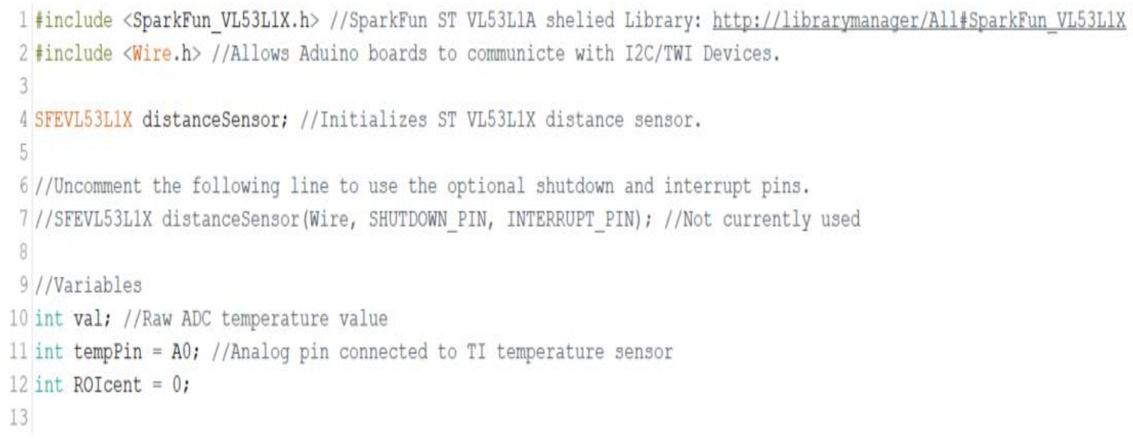


**Figure 26: VL53L1X 16x16 SPAD Receiving Array [44].**

The Timing Budget setting, like the location of the ROI, was another unknown. The Timing Budget sets the time required by the sensor to perform one range measurement, with the minimum being 20ms and the maximum being 1000ms [45]. However, this setting was limited to 15, 20, 33, 50, 100, 200, or 500ms in the SparkFun Arduino software library [39]. The Inter-Measurement Period determines the period between each range measurement. This setting was left at its default setting of 100ms [36]. Unlike the ROI and Timing Budget settings, the final setting -- the Distance Mode -- was easy to determine. There are three options for this setting, including Short, Medium (not implemented in the Arduino Library [39]), and Long. Each setting has its own maximum measurement range, with Short being 1.3m and Long being 4m. Since the tube assembly was less than 0.4m long, the Short Mode was selected.

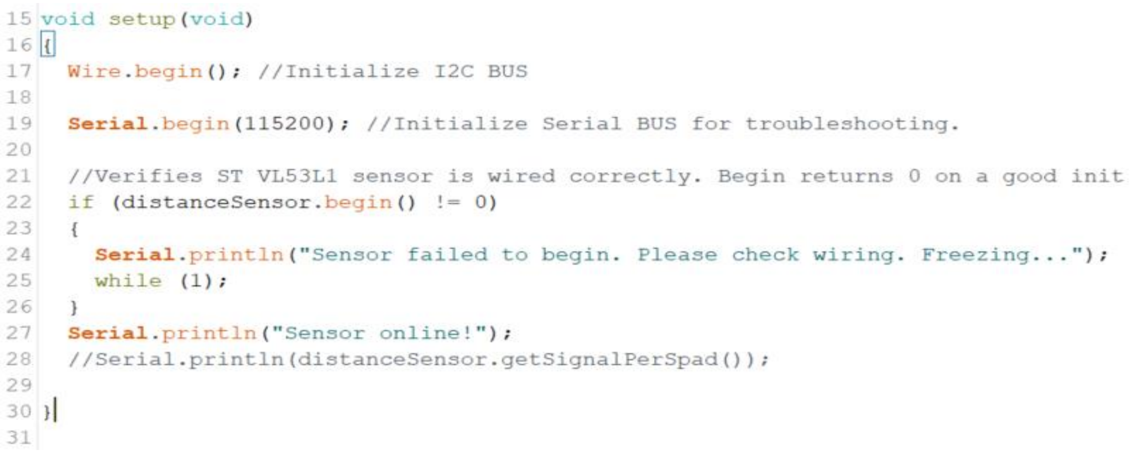
To determine the two unknown settings -- the location of the ROI and the Timing Budget -- the ROI Calibration code was written. This code, which is in shown in Appendix B, automates the process of determining the optimal ROI location by scanning through each of the 256 center points to determine if the measurement is within ±1mm of a set distance. The first section of this code is shown in Figure 27, and it includes the required software libraries needed to operate the VL53L1X TOF sensor and the microcontroller's $I^2C$ communication BUS (lines 1 and 2). These two libraries were used in the previous code so they should look familiar. Following the libraries, line 4 of the code initializes the TOF sensor and lines 10 through 12 are the variables used in the program to store information. Two of these variables should look familiar -- those are "val" and "tempPin", which again are used for the same purpose. However, it is the third variable on line 12, “ROIcent”, which is very important. This variable holds the ROI center point that is used in the “setROI” command used later in this code.



```
 1 #include <SparkFun_VL53L1X.h> //SparkFun ST VL53L1A shelied Library: http://librarymanager/All#SparkFun_VL53L1X
 2 #include <Wire.h> //Allows Aduino boards to communicte with I2C/TWI Devices.
 3
 4 SFEVL53L1X distanceSensor; //Initializes ST VL53L1X distance sensor.
 5
 6 //Uncomment the following line to use the optional shutdown and interrupt pins.
 7 //SFEVL53L1X distanceSensor(Wire, SHUTDOWN_PIN, INTERRUPT_PIN); //Not currently used
 8
 9 //Variables
10 int val; //Raw ADC temperature value
11 int tempPin = A0; //Analog pin connected to TI temperature sensor
12 int ROIcent = 0;
13
```

**Figure 27: ROI Calibration Source Code Section One.**

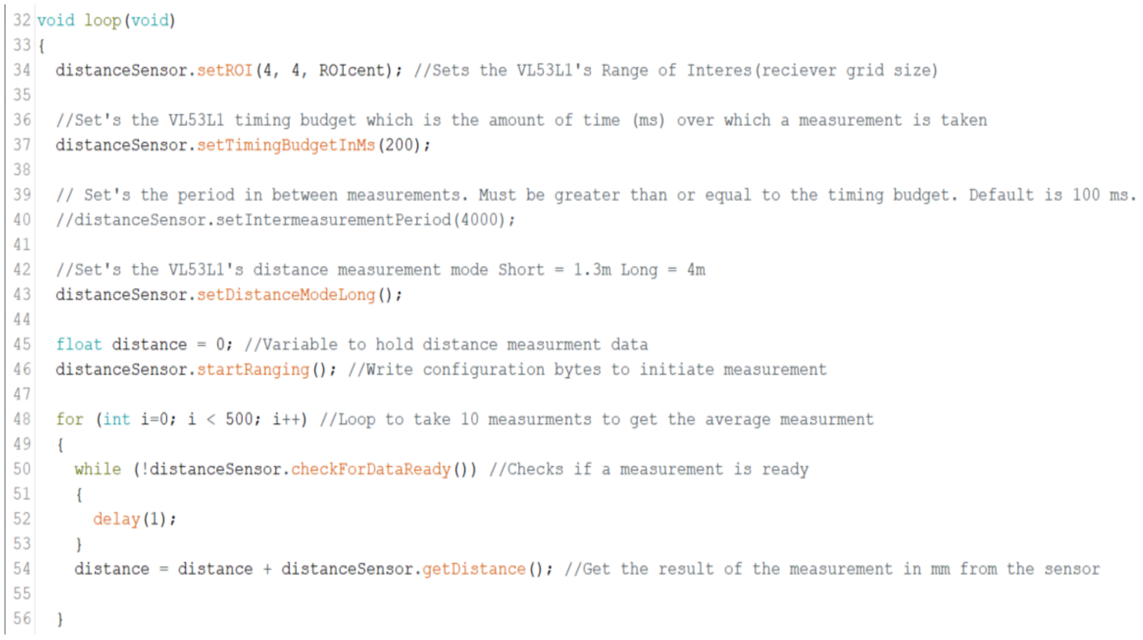
Section Two of the ROI Calibration Code contains the “setup” function, which is similar to the “setup” function used in the previous code. However, this time around, the function does not contain the commands that set the ROI, Timing Budget, and Distance Mode. The rest of the function is the same as the one used in the previous code and is shown in Figure 28, so it still initializes the $I^2C$ and Serial communication BUS(s) and initializes the TOF sensor.



```
15 void setup(void)
16 {
17   Wire.begin(); //Initialize I2C BUS
18
19   Serial.begin(115200); //Initialize Serial BUS for troubleshooting.
20
21   //Verifies ST VL53L1 sensor is wired correctly. Begin returns 0 on a good init
22   if (distanceSensor.begin() != 0)
23   {
24     Serial.println("Sensor failed to begin. Please check wiring. Freezing...");
25     while (1);
26   }
27   Serial.println("Sensor online!");
28   //Serial.println(distanceSensor.getSignalPerSpad());
29
30 }
31
```

**Figure 28: ROI Calibration Source Code Section Two.**

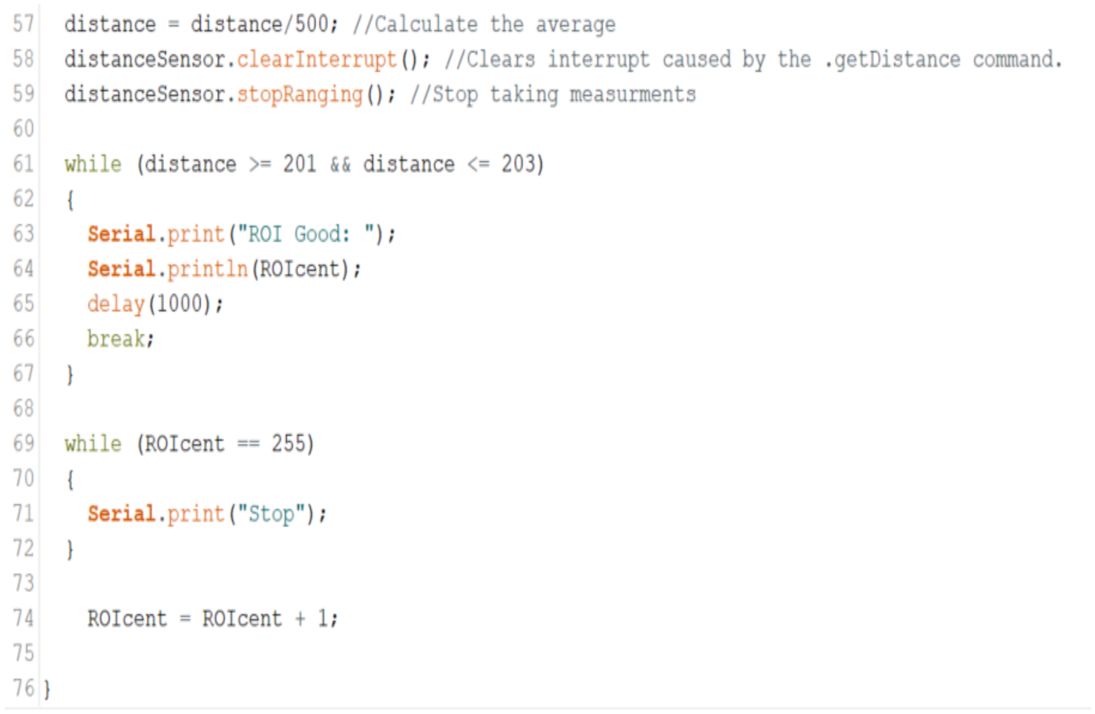
Like the previous code, the next section shows the “loop” function. As stated before, the “loop” function is run continuously by the microcontroller. The commands that set the ROI, Timing Budget and Distance mode were moved from the “setup” function to this function, as shown in Figure 29. This was done so that each time the microcontroller ran this code, it would reset the ROI, Timing Budget, and Distance Mode settings of the TOF. This allows the program to scan through all 256 ROI center points. This portion of the “loop” function also contains the “for” loop that allows for multiple distance measurements to be taken so that an average distance could be calculated.



```
void loop(void)
{
  distanceSensor.setROI(4, 4, ROIcent); //Sets the VL53L1's Range of Interes(reciever grid size)

  //Set's the VL53L1 timing budget which is the amount of time (ms) over which a measurement is taken
  distanceSensor.setTimingBudgetInMs(200);

  // Set's the period in between measurements. Must be greater than or equal to the timing budget. Default is 100 ms.
  //distanceSensor.setIntermeasurementPeriod(4000);

  //Set's the VL53L1's distance measurement mode Short = 1.3m Long = 4m
  distanceSensor.setDistanceModeLong();

  float distance = 0; //Variable to hold distance measurment data
  distanceSensor.startRanging(); //Write configuration bytes to initiate measurement

  for (int i=0; i < 500; i++) //Loop to take 10 measurments to get the average measurment
  {
    while (!distanceSensor.checkForDataReady()) //Checks if a measurement is ready
    {
      delay(1);
    }
    distance = distance + distanceSensor.getDistance(); //Get the result of the measurement in mm from the sensor

  }
```

**Figure 29: ROI Calibration Source Code Section Three A.**

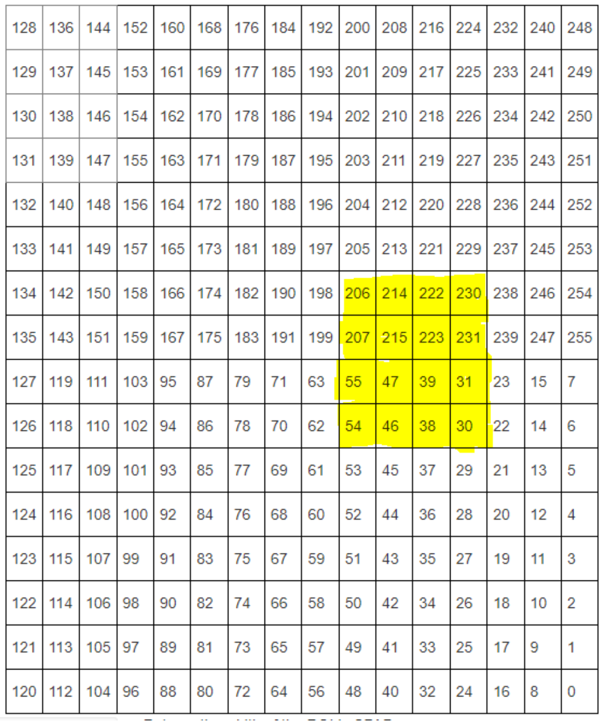
The rest of the “loop” function is shown in Figure 30. Like the previous code, Section Three B shows the commands used to calculate the average distance line 57, and the command used to stop and reset the TOF sensor. What is different are the two “while” loops. The first “while” loop shown on lines 61 through 67 tell the microcontroller that if the distance measurement is within 1mm of the set distance (in this case, the set distance is 202mm) to transmit via the Serial BUS that the ROI center point is good and what that center point is. That ROI center point is then incremented on line 74 of the code, which adds one to the "ROIcent" variable. This continues to happen until the "ROIcent" variable reaches 255, which is when the second "while" loop on lines 69 through 72 stops the program and has the microcontroller transmit stop via the Serial BUS so that the author would know that all ROI center points have been scanned.



```
57  distance = distance/500; //Calculate the average
58  distanceSensor.clearInterrupt(); //Clears interrupt caused by the .getDistance command.
59  distanceSensor.stopRanging(); //Stop taking measurments
60
61  while (distance >= 201 && distance <= 203)
62  {
63    Serial.print("ROI Good: ");
64    Serial.println(ROIcent);
65    delay(1000);
66    break;
67  }
68
69  while (ROIcent == 255)
70  {
71    Serial.print("Stop");
72  }
73
74    ROIcent = ROIcent + 1;
75
76 }
```

**Figure 30: ROI Calibration Source Code Section Three B.**

This calibration program was run at six different positions -- 102, 152, 202, 252, 301, and 352mm -- with the Timing Budget set to 50, 100 and 200ms. The calibration process determined that the optimal ROI location was at grid position 230, shown in Figure 31, and the optimal Timing Budget was 200ms. The data captured through this calibration process are presented in the Results section.



| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 128 | 136 | 144 | 152 | 160 | 168 | 176 | 184 | 192 | 200 | 208 | 216 | 224 | 232 | 240 | 248 |
| 129 | 137 | 145 | 153 | 161 | 169 | 177 | 185 | 193 | 201 | 209 | 217 | 225 | 233 | 241 | 249 |
| 130 | 138 | 146 | 154 | 162 | 170 | 178 | 186 | 194 | 202 | 210 | 218 | 226 | 234 | 242 | 250 |
| 131 | 139 | 147 | 155 | 163 | 171 | 179 | 187 | 195 | 203 | 211 | 219 | 227 | 235 | 243 | 251 |
| 132 | 140 | 148 | 156 | 164 | 172 | 180 | 188 | 196 | 204 | 212 | 220 | 228 | 236 | 244 | 252 |
| 133 | 141 | 149 | 157 | 165 | 173 | 181 | 189 | 197 | 205 | 213 | 221 | 229 | 237 | 245 | 253 |
| 134 | 142 | 150 | 158 | 166 | 174 | 182 | 190 | 198 | 206 | 214 | 222 | 230 | 238 | 246 | 254 |
| 135 | 143 | 151 | 159 | 167 | 175 | 183 | 191 | 199 | 207 | 215 | 223 | 231 | 239 | 247 | 255 |
| 127 | 119 | 111 | 103 | 95 | 87 | 79 | 71 | 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |
| 126 | 118 | 110 | 102 | 94 | 86 | 78 | 70 | 62 | 54 | 46 | 38 | 30 | 22 | 14 | 6 |
| 125 | 117 | 109 | 101 | 93 | 85 | 77 | 69 | 61 | 53 | 45 | 37 | 29 | 21 | 13 | 5 |
| 124 | 116 | 108 | 100 | 92 | 84 | 76 | 68 | 60 | 52 | 44 | 36 | 28 | 20 | 12 | 4 |
| 123 | 115 | 107 | 99 | 91 | 83 | 75 | 67 | 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 |
| 122 | 114 | 106 | 98 | 90 | 82 | 74 | 66 | 58 | 50 | 42 | 34 | 26 | 18 | 10 | 2 |
| 121 | 113 | 105 | 97 | 89 | 81 | 73 | 65 | 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 |
| 120 | 112 | 104 | 96 | 88 | 80 | 72 | 64 | 56 | 48 | 40 | 32 | 24 | 16 | 8 | 0 |

**Figure 31: VL53L1X Receiving Array ROI Setting.**

With the optimal TOF settings determined, the accuracy of the TOF and temperature sensor could be tested. This test was conducted at three different temperatures -- 0°C, 23°C and 49°C -- using an ESPEC environmental chamber located at TLX Technologies. At each temperature, distance measurement readings were recorded at three different positions and the average of the readings was compared to the actual distance measurement. This same technique was used to test the temperature sensor. The results of this testing are covered in the Results section. For further testing details, see the MSE Capstone LLI Test Plan in Appendix C.

## Phase Three Development

With Phase Two complete, the next phase of the project was to further develop the LLI's firmware by adding an algorithm that converts the distance measurement from the TOF sensor and the temperature sensor into a weight. This algorithm uses two equations to convert the distance measurement into a weight. Equation (1) is used to calculate the volume of the liquid in the storage cylinder:

$$V = \pi * (r - t)^2 * (h - d), \tag{1}$$

where V, the volume of the liquid, equals the squared quantity r, the radius of the cylinder, minus t, the thickness of the cylinder walls, times the quantity, h, the height of the cylinder, minus d, the distance to the top of the agent. Equation (2) is employed to calculate the mass of the liquid agent:

$$m = \rho * V, \tag{2}$$

where m, the mass of the liquid, equals ρ, the density of the liquid, times V, the volume of the liquid. Density was determined by using linear interpolation that pulled temperature and density data from two 24-bit arrays. A linear interpolation algorithm developed by Llamas was employed from the Arduino InterpolationLib library [46]. Linear Interpolation is a method for estimating the value of a function between any two known values [47]. Equation (3) is employed to perform linear interpolation:

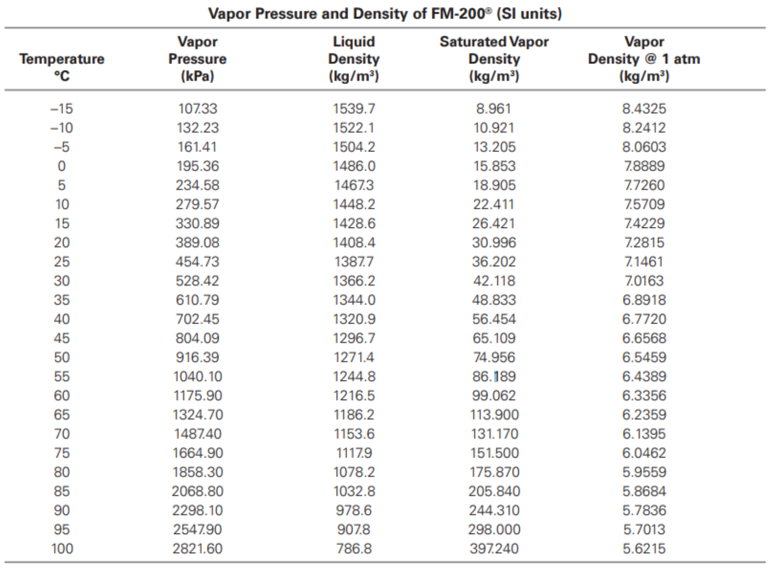
$$y = y_1 + \frac{(x - x_1)(y_2 - y_1)}{x_2 - x_1}, \tag{3}$$

where $x_1$ and $y_1$ are the coordinates of the first known value, $x_2$ and $y_2$ are the coordinates of the second known value, x is the value that falls between $x_1$ and $x_2$ where the interpolated value of y needs to be determined. For example, to determine the density of FM-200 at 23°C, x is the measured temperature 23, and y is the unknown density value. To determine y via Equation (3), it is necessary to know what two points that the x and y values fall between. This can be determined by using the temperature and density data from Table 3. Table 3 indicates the density of FM-200 at 20°C is 1408.4kg/m$^3$ and the density at 25°C is 1387.7, which serve as the ($x_1$, $y_1$) and ($x_2$, $y_2$) coordinates. This results in Equation (4):

$$y = 1408.4 + \frac{(23-20)(1387.7-1408.4)}{25-20}. \tag{4}$$

Solving Equation (4) gives an interpolated value of 1395.98kg/m$^3$. This interpolated value can then be entered into Equation (2) to determine the weight of the agent. The temperature and density data entered in the two 24-bit arrays for FM-200 were pulled from Table 3.

**Table 3: FM-200 Density versus Temperature [48].**

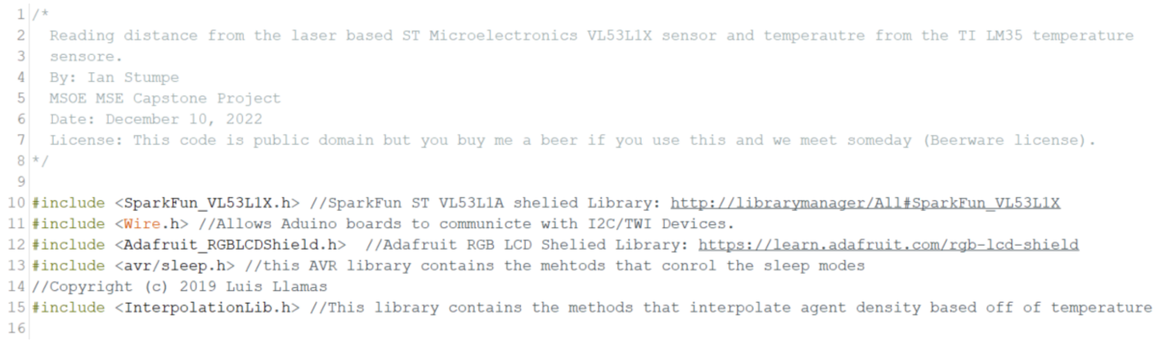


Vapor Pressure and Density of FM-200® (SI units)

| Temperature °C | Vapor Pressure (kPa) | Liquid Density (kg/m³) | Saturated Vapor Density (kg/m³) | Vapor Density @ 1 atm (kg/m³) |
|---|---|---|---|---|
| –15 | 107.33 | 1539.7 | 8.961 | 8.4325 |
| –10 | 132.23 | 1522.1 | 10.921 | 8.2412 |
| –5 | 161.41 | 1504.2 | 13.205 | 8.0603 |
| 0 | 195.36 | 1486.0 | 15.853 | 7.8889 |
| 5 | 234.58 | 1467.3 | 18.905 | 7.7260 |
| 10 | 279.57 | 1448.2 | 22.411 | 7.5709 |
| 15 | 330.89 | 1428.6 | 26.421 | 7.4229 |
| 20 | 389.08 | 1408.4 | 30.996 | 7.2815 |
| 25 | 454.73 | 1387.7 | 36.202 | 7.1461 |
| 30 | 528.42 | 1366.2 | 42.118 | 7.0163 |
| 35 | 610.79 | 1344.0 | 48.833 | 6.8918 |
| 40 | 702.45 | 1320.9 | 56.454 | 6.7720 |
| 45 | 804.09 | 1296.7 | 65.109 | 6.6568 |
| 50 | 916.39 | 1271.4 | 74.956 | 6.5459 |
| 55 | 1040.10 | 1244.8 | 86.189 | 6.4389 |
| 60 | 1175.90 | 1216.5 | 99.062 | 6.3356 |
| 65 | 1324.70 | 1186.2 | 113.900 | 6.2359 |
| 70 | 1487.40 | 1153.6 | 131.170 | 6.1395 |
| 75 | 1664.90 | 1117.9 | 151.500 | 6.0462 |
| 80 | 1858.30 | 1078.2 | 175.870 | 5.9559 |
| 85 | 2068.80 | 1032.8 | 205.840 | 5.8684 |
| 90 | 2298.10 | 978.6 | 244.310 | 5.7836 |
| 95 | 2547.90 | 907.8 | 298.000 | 5.7013 |
| 100 | 2821.60 | 786.8 | 397.240 | 5.6215 |

For this phase of the project, only one density/temperature look-up table was used. This was done because the main purpose of this phase was to create an algorithm that accurately calculates the weight of an agent. If this goal could be achieved with one agent, it could be achieved with multiple agents. Using this information, the weight conversion algorithm was created and added to the source code, creating Revision 2.0. This firmware code is shown in Appendix D. The testing methods used to verify Revision 2.0 of the firmware are presented in Appendix C and results are discussed in the

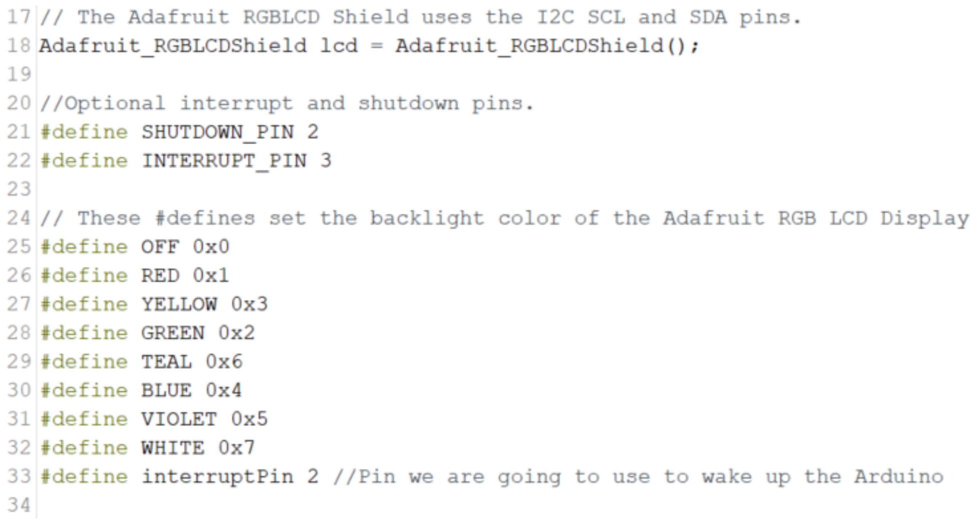
Results section.  There were significant changes made to the original source code that is shown in Appendix A.  The first change on line 15 shown in Figure 32 is the addition of the “InterpolationLib.h”, which was discussed earlier.



```
 1 /*
 2   Reading distance from the laser based ST Microelectronics VL53L1X sensor and temperautre from the TI LM35 temperature
 3   sensore.
 4   By: Ian Stumpe
 5   MSOE MSE Capstone Project
 6   Date: December 10, 2022
 7   License: This code is public domain but you buy me a beer if you use this and we meet someday (Beerware license).
 8 */
 9
10 #include <SparkFun_VL53L1X.h> //SparkFun ST VL53L1A shelied Library: http://librarymanager/All#SparkFun_VL53L1X
11 #include <Wire.h> //Allows Aduino boards to communicte with I2C/TWI Devices.
12 #include <Adafruit_RGBLCDShield.h>  //Adafruit RGB LCD Shelied Library: https://learn.adafruit.com/rgb-lcd-shield
13 #include <avr/sleep.h> //this AVR library contains the mehtods that conrol the sleep modes
14 //Copyright (c) 2019 Luis Llamas
15 #include <InterpolationLib.h> //This library contains the methods that interpolate agent density based off of temperature
16
```

**Figure 32: Source Code Revision 2.0 Section One.**

Without this library, determining the density of the agent would not be possible without significant changes to this code.  There were no significant changes made to the next section of code, shown in Figure 33, which is the same as the Appendix A code.

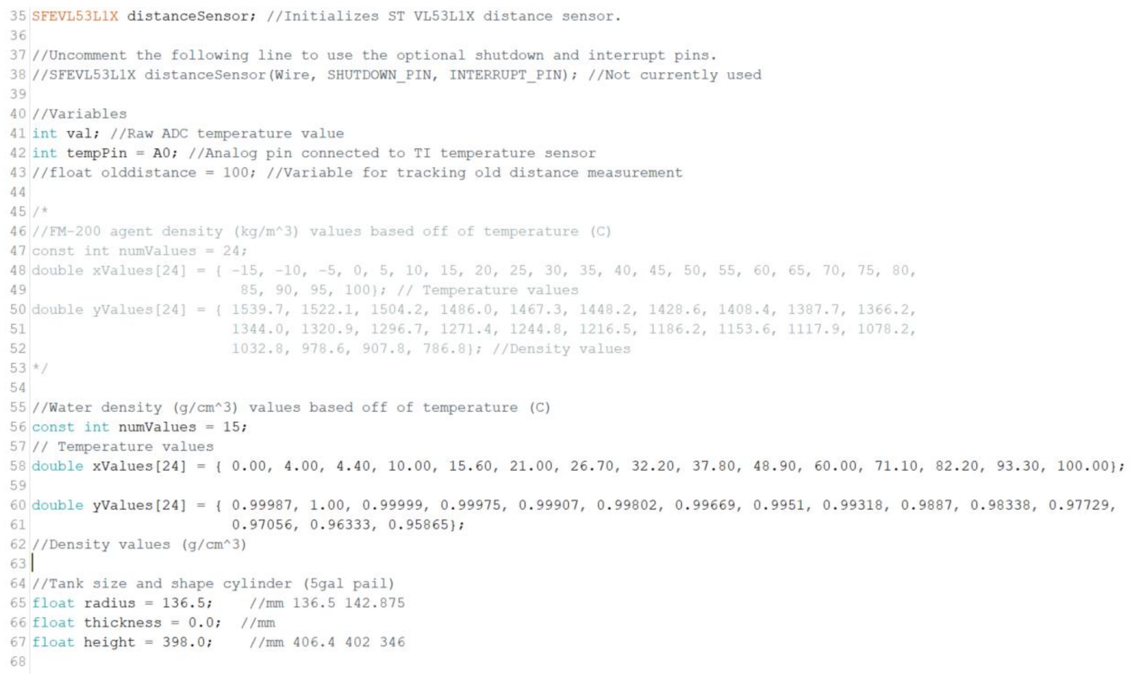


```
17 // The Adafruit RGBLCD Shield uses the I2C SCL and SDA pins.
18 Adafruit_RGBLCDShield lcd = Adafruit_RGBLCDShield();
19
20 //Optional interrupt and shutdown pins.
21 #define SHUTDOWN_PIN 2
22 #define INTERRUPT_PIN 3
23
24 // These #defines set the backlight color of the Adafruit RGB LCD Display
25 #define OFF 0x0
26 #define RED 0x1
27 #define YELLOW 0x3
28 #define GREEN 0x2
29 #define TEAL 0x6
30 #define BLUE 0x4
31 #define VIOLET 0x5
32 #define WHITE 0x7
33 #define interruptPin 2 //Pin we are going to use to wake up the Arduino
34
```

**Figure 33: Source Code Revision 2.0 Section Two.**

There were several changes made to the next section of code, shown in Figure 34. The first of these changes is shown in lines 46 through 62, which is the addition of those 24-bit arrays that hold the temperature and density data for the FM-200 agent and water.

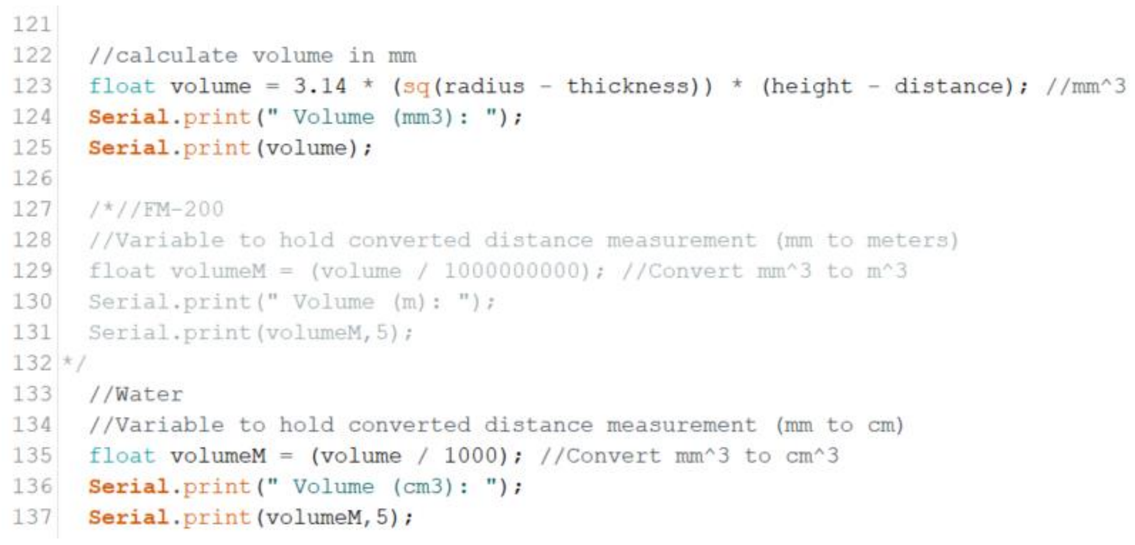
The temperature and density data for water were added because it was not possible to test the LLI with FM-200 nor Novec 1230, which this device is intended to be used with. The reasoning for this is covered in more detail in the Results section. The other major change to this section is the addition of three new variables, shown in lines 65 through 67, where they are used to store the dimensions of the agent storage cylinder. These variables are used later in the program to calculate the volume of agent.



```
35 SFEVL53L1X distanceSensor; //Initializes ST VL53L1X distance sensor.
36
37 //Uncomment the following line to use the optional shutdown and interrupt pins.
38 //SFEVL53L1X distanceSensor(Wire, SHUTDOWN_PIN, INTERRUPT_PIN); //Not currently used
39
40 //Variables
41 int val; //Raw ADC temperature value
42 int tempPin = A0; //Analog pin connected to TI temperature sensor
43 //float olddistance = 100; //Variable for tracking old distance measurement
44
45 /*
46 //FM-200 agent density (kg/m^3) values based off of temperature (C)
47 const int numValues = 24;
48 double xValues[24] = { -15, -10, -5, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80,
49                         85, 90, 95, 100}; // Temperature values
50 double yValues[24] = { 1539.7, 1522.1, 1504.2, 1486.0, 1467.3, 1448.2, 1428.6, 1408.4, 1387.7, 1366.2,
51                        1344.0, 1320.9, 1296.7, 1271.4, 1244.8, 1216.5, 1186.2, 1153.6, 1117.9, 1078.2,
52                        1032.8, 978.6, 907.8, 786.8}; //Density values
53 */
54
55 //Water density (g/cm^3) values based off of temperature (C)
56 const int numValues = 15;
57 // Temperature values
58 double xValues[24] = { 0.00, 4.00, 4.40, 10.00, 15.60, 21.00, 26.70, 32.20, 37.80, 48.90, 60.00, 71.10, 82.20, 93.30, 100.00};
59
60 double yValues[24] = { 0.99987, 1.00, 0.99999, 0.99975, 0.99907, 0.99802, 0.99669, 0.9951, 0.99318, 0.9887, 0.98338, 0.97729,
61                        0.97056, 0.96333, 0.95865};
62 //Density values (g/cm^3)
63
64 //Tank size and shape cylinder (5gal pail)
65 float radius = 136.5;     //mm 136.5 142.875
66 float thickness = 0.0;  //mm
67 float height = 398.0;     //mm 406.4 402 346
68
```

**Figure 34: Source Code Revision 2.0 Section Three.**

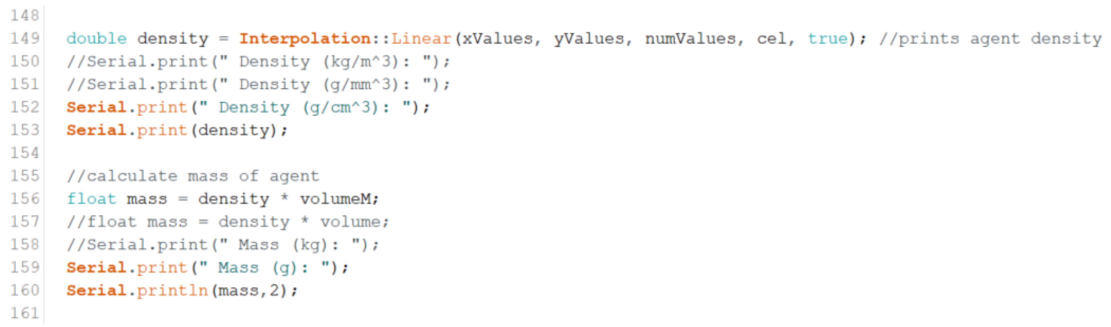
The next section of this code, which covers the “setup” function, saw no changes from what it was in the original code that is shown in Figure 22. However, the “loop” function had several lines of code added to it beyond what is shown in Figures 23 and 24. The first addition to this section on lines 122 to 125 in Figure 35 is the code needed to calculate the volume of the agent using those new variables described earlier and the calculated distance to the agent. Additionally, lines 127 through 137 were added to convert the volume, which is initially calculated in millimeters cubed to either meters cubed, or centimeters cubed, depending on if the agent is FM-200 or water. These conversions were needed for the chosen microcontroller to correctly calculate the weight of the agent.



```

  //calculate volume in mm
  float volume = 3.14 * (sq(radius - thickness)) * (height - distance); //mm^3
  Serial.print(" Volume (mm3): ");
  Serial.print(volume);

  /*//FM-200
  //Variable to hold converted distance measurement (mm to meters)
  float volumeM = (volume / 1000000000); //Convert mm^3 to m^3
  Serial.print(" Volume (m): ");
  Serial.print(volumeM,5);
*/
  //Water
  //Variable to hold converted distance measurement (mm to cm)
  float volumeM = (volume / 1000); //Convert mm^3 to cm^3
  Serial.print(" Volume (cm3): ");
  Serial.print(volumeM,5);
```

**Figure 35: Source Code Revision 2.0 Section Four.**

The final addition to the "loop" function is the code shown in Figure 36. This code determines the density of the agent and the total mass of the agent. Using Equation (3), the code on line 149 calculates the density for the calculated temperature stored in the variable "cel", which is then stored in the variable "density". This variable is then used on line 156 to calculate the mass of the agent.



```
148
149  double density = Interpolation::Linear(xValues, yValues, numValues, cel, true); //prints agent density
150  //Serial.print(" Density (kg/m^3): ");
151  //Serial.print(" Density (g/mm^3): ");
152  Serial.print(" Density (g/cm^3): ");
153  Serial.print(density);
154
155  //calculate mass of agent
156  float mass = density * volumeM;
157  //float mass = density * volume;
158  //Serial.print(" Mass (kg): ");
159  Serial.print(" Mass (g): ");
160  Serial.println(mass,2);
161
```

**Figure 36: Source Code Revision 2.0 Section Five.**

## Phase Four Development

Phase Four of the project entailed the addition of a pushbutton to the POC and new firmware that only takes a single measurement when the pushbutton is pressed. Once power is applied to the unit, the firmware takes one measurement of distance and temperature, and then displays the information on the LCD for five seconds. Once the five seconds are up, the LLI goes into a sleep mode. In sleep mode, the LCD is powered down so much less power is required. Once the button is pressed, the unit wakes up, initiating a distance and temperature measurement, displaying it on the LCD again for five seconds and then goes back into sleep mode. This firmware, "LLI Firmware with Interrupt", is shown in Appendix E. Compared to the "LLI Firmware Rev 1.0" shown in Appendix A, the major changes are in the "loop" function, which saw the code to take distance measurements and temperature measurements move to a new function called "Going_To_Sleep()". The first section of this function is shown in Figure 37.

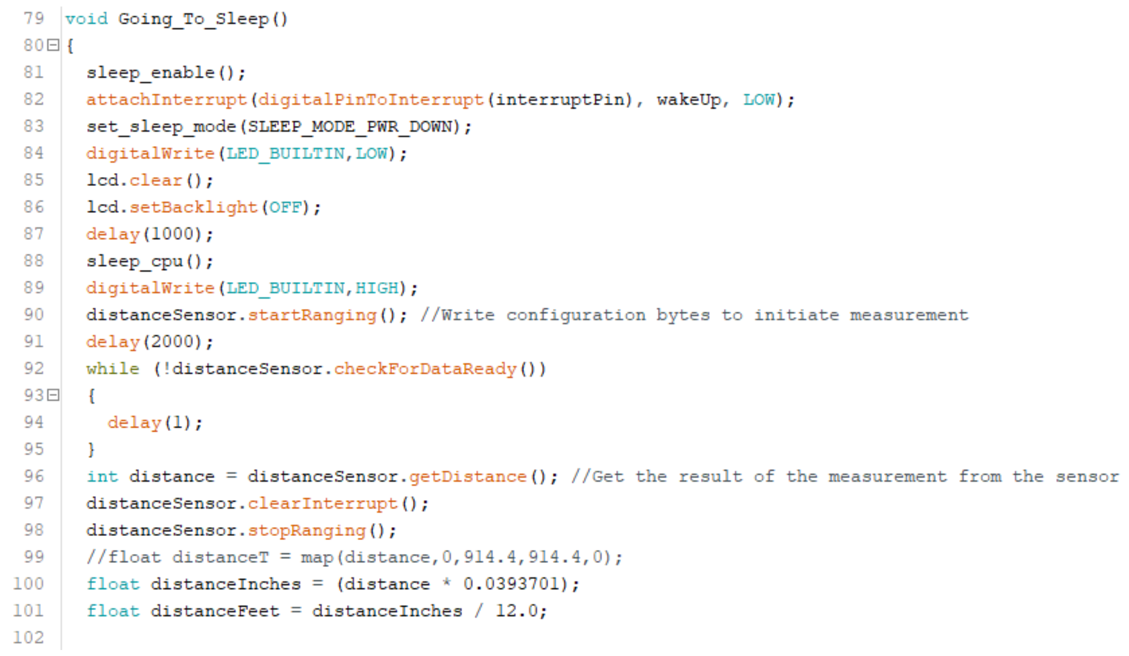
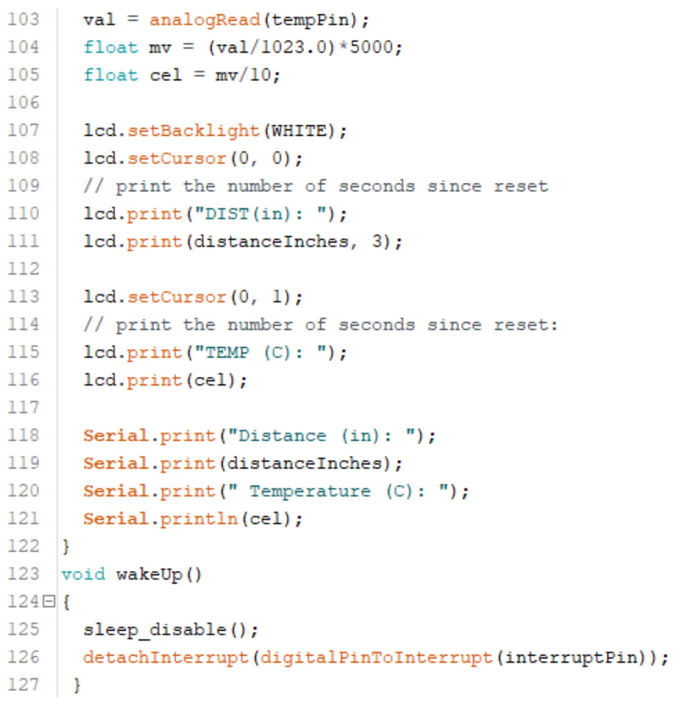

```
void Going_To_Sleep()
{
  sleep_enable();
  attachInterrupt(digitalPinToInterrupt(interruptPin), wakeUp, LOW);
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);
  digitalWrite(LED_BUILTIN,LOW);
  lcd.clear();
  lcd.setBacklight(OFF);
  delay(1000);
  sleep_cpu();
  digitalWrite(LED_BUILTIN,HIGH);
  distanceSensor.startRanging(); //Write configuration bytes to initiate measurement
  delay(2000);
  while (!distanceSensor.checkForDataReady())
  {
    delay(1);
  }
  int distance = distanceSensor.getDistance(); //Get the result of the measurement from the sensor
  distanceSensor.clearInterrupt();
  distanceSensor.stopRanging();
  //float distanceT = map(distance,0,914.4,914.4,0);
  float distanceInches = (distance * 0.0393701);
  float distanceFeet = distanceInches / 12.0;

```

**Figure 37: Going to Sleep Function Section A.**

With this new function comes some new commands -- the first is on line 81, the "sleep_enable()", which enables the microcontroller to enter into a sleep mode, which is a low power mode where the microcontroller disables certain circuitry or clocks that drive some of its peripheral functions. Depending on the microcontroller, it may have several different sleep modes that can be deployed. For example, the ATmega328P microcontroller that is used in this application has three modes all with varying degrees of power savings; they include Idle, Standby and Power-Down. In this application, the Power-Down mode is used, which shuts down all peripherals and offers the greatest amount of power savings. Following line 83, which tells the microcontroller which sleep mode to enter, the display is cleared, and the backlight is turned off. After a one second delay (line 87), the microcontroller enters the Power-Down sleep mode. The rest of the function starting at line 92 through 101, shown in Figure 38, is the same code used to determine the distance and temperature and to display it on the LCD.



```
  val = analogRead(tempPin);
  float mv = (val/1023.0)*5000;
  float cel = mv/10;

  lcd.setBacklight(WHITE);
  lcd.setCursor(0, 0);
  // print the number of seconds since reset
  lcd.print("DIST(in): ");
  lcd.print(distanceInches, 3);

  lcd.setCursor(0, 1);
  // print the number of seconds since reset:
  lcd.print("TEMP (C): ");
  lcd.print(cel);

  Serial.print("Distance (in): ");
  Serial.print(distanceInches);
  Serial.print(" Temperature (C): ");
  Serial.println(cel);
}
void wakeUp()
{
  sleep_disable();
  detachInterrupt(digitalPinToInterrupt(interruptPin));
 }
```

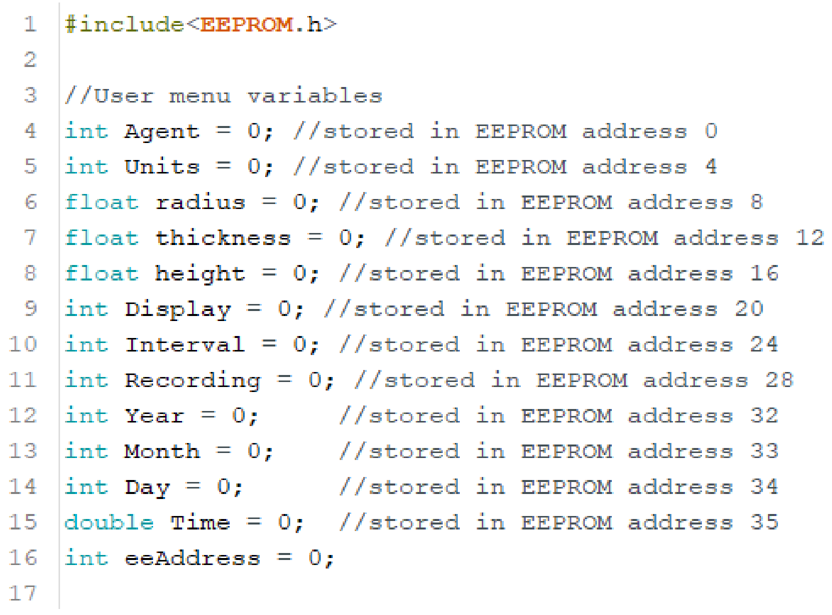
**Figure 38: Going to Sleep Function Section B.**

The last change shown in Figure 38 is the addition of the “wakeUp()” function located on lines 123 through 127. It is this function that tells the microcontroller to return to normal operation when the pushbutton is pressed.

**Phase Five Development**

Phase Five of the project saw the development and implementation of the LLI’s user interface. The first step of the development process was determining what the user interface was going to do. The author determined that the user interface should give the user the ability to select the agent type (FM-200, Novec 1230 and Water), set units of measurement (SI or US Imperial), enter the dimensions of the agent storage cylinder, set the display’s refresh rate (continuous or single), set the data recording interval, enable/disable data recording, and set the date and time. Following that, it was necessary to determine how the user would interact with it. There were several options to choose from such as a graphical user interface (GUI), command line interface (CLI), and a menu-driven user interface to just name a few. For its simplicity and low resource requirements, a command line type interface was chosen. Using a universal asynchronous receiver-transmitter (UART) communications protocol, the CLI would utilize the Arduino UNO’s Atmega328 microcontroller’s UART to transmit and receive data to a host device such as a computer or tablet using the Arduino IDE’s built-in serial monitor or simple terminal emulator program such as Tera Term or Putty. The physical connection between the host device and the LLI (Arduino UNO) utilizes a standard USB-A to USB-B cable to connect to the host device’s USB-A port and the Arduino UNO’s USB-B port (see Figure 18).

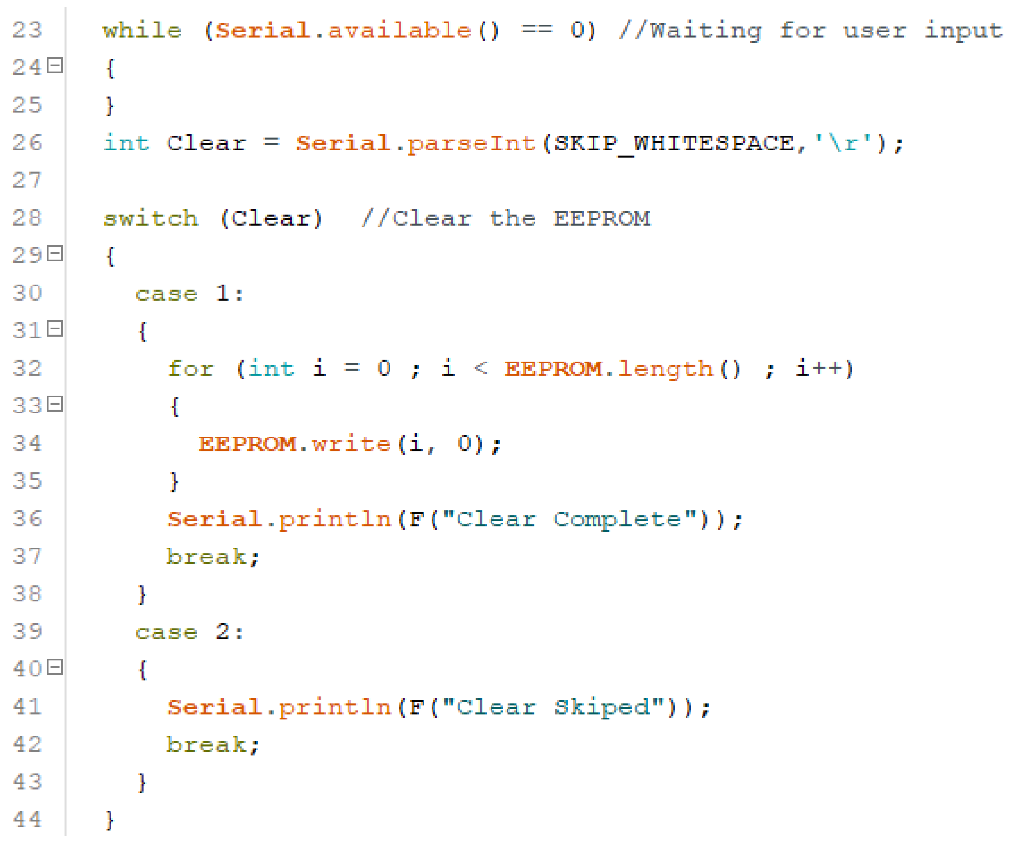
With the user-interface's functionality, type, and physical interface determined, the next step was to develop the code required by the microcontroller to run the user-interface. The code development process was done incrementally by first developing a standalone program just for the user-interface. This allowed the author to fully develop and test the user-interface without having to modify the original LLI firmware shown in Appendix D. The final version of the standalone user-interface program is the "Serial_Comms_Rev_5" code shown in Appendix J. The first section of this code, shown in Figure 39, shows the global variables that store the information entered by the user. Additionally, a new software library was used, which is the EEPROM.h library. This library enables the use of the Adruino UNO's onboard EEPROM chip, which can store up to 1024 bytes of data. With the addition of this functionality, user entered settings and mass measurement data can be permanently stored to the device. So, no settings nor data are lost when the LLI is powered off. The comments detail the EEPROM address location where the variables are stored.



```
#include<EEPROM.h>

//User menu variables
int Agent = 0; //stored in EEPROM address 0
int Units = 0; //stored in EEPROM address 4
float radius = 0; //stored in EEPROM address 8
float thickness = 0; //stored in EEPROM address 12
float height = 0; //stored in EEPROM address 16
int Display = 0; //stored in EEPROM address 20
int Interval = 0; //stored in EEPROM address 24
int Recording = 0; //stored in EEPROM address 28
int Year = 0;      //stored in EEPROM address 32
int Month = 0;     //stored in EEPROM address 33
int Day = 0;       //stored in EEPROM address 34
double Time = 0;   //stored in EEPROM address 35
int eeAddress = 0;

```

**Figure 39: Section One of the Serial_Comms_Rev_5 code.**

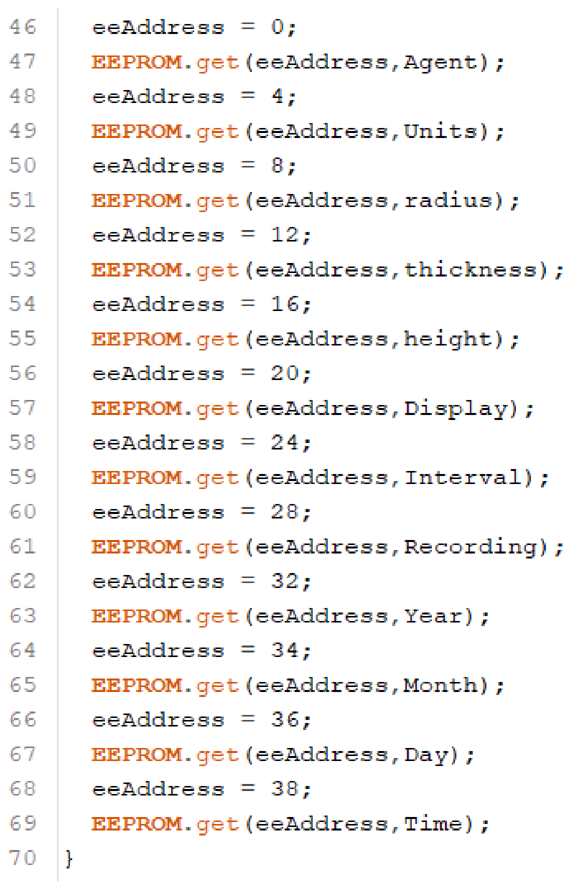
Section Two of the code covers the “setup()” function of the code. Part A of the “setup()” function covering lines 20 to 45 is shown in Figure 40. The code on line 20 should look familiar. It is the line of code that enables the microcontroller’s Serial communication BUS, which was previously used to verify the functionality of the code so the microcontroller could only output data -- now it will receive data. This functionality is shown in the remaining lines of Part A. On line 21, the microcontroller is again outputting data to the Serial BUS, but in this case, it is asking the user a question and then waits for a response (while loop on lines 23 to 25). The question is, does the user want to clear the EEPROM, since the program now writes data to the EEPROM the user needs a way to set the LLI back to its factory settings. Once the user enters ‘1’ or ‘2’, the program stores that value to the variable “Clear”, which is done on line 26. The value of this variable is then used in the switch case to either clear the EEPROM or to skip the clear process and continue with the program. If the value of “Clear” is 1, the program clears the EEPROM by writing zeros to each memory location in the EEPROM. This is done by the ‘for’ loop on lines 32 to 35. Once that process is done, the program via the code on line 36 tells the user that the process is finished. If the value of “Clear” is 2, the program via code on line 41 tells the user that the clear process has been skipped. Once either one of those lines of code is processed by the microcontroller, it continues to run the program via the “break” commands shown on lines 37 and 42.



```
while (Serial.available() == 0) //Waiting for user input
{
}
int Clear = Serial.parseInt(SKIP_WHITESPACE,'\r');

switch (Clear)  //Clear the EEPROM
{
  case 1:
  {
    for (int i = 0 ; i < EEPROM.length() ; i++)
    {
      EEPROM.write(i, 0);
    }
    Serial.println(F("Clear Complete"));
    break;
  }
  case 2:
  {
    Serial.println(F("Clear Skiped"));
    break;
  }
}
```

**Figure 40: Section Two Part A of the Serial_Comms_Rev_5 code.**

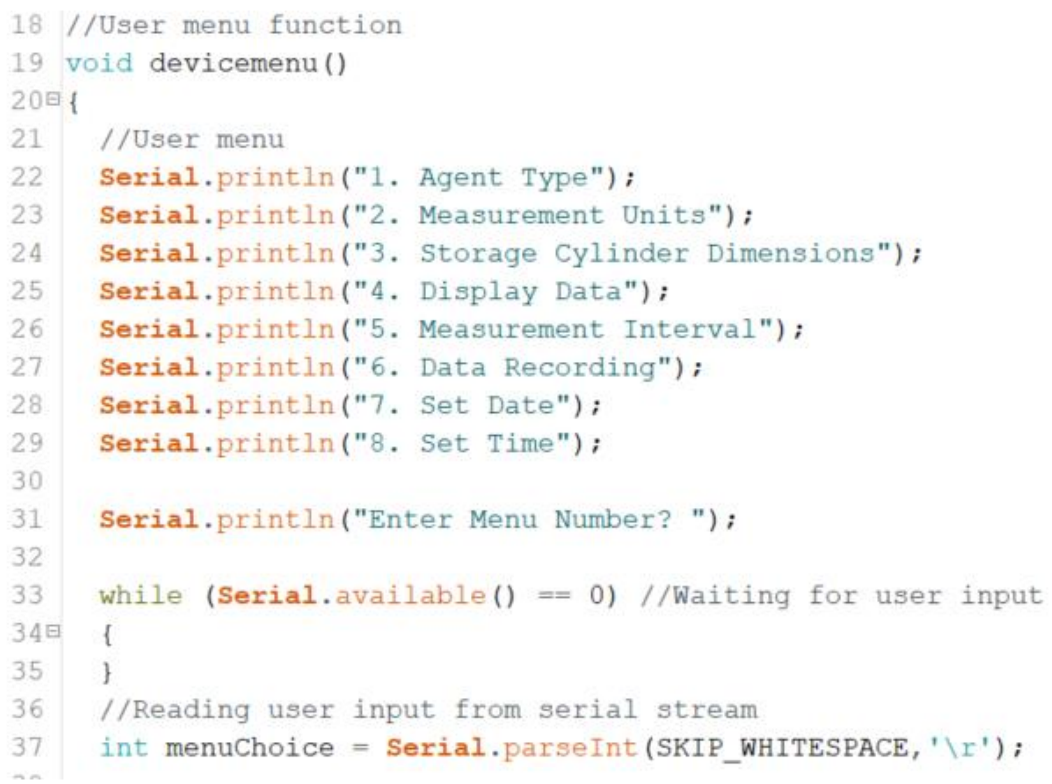
With the processing of the "break" commands on either of those lines completed, the microcontroller can now begin to process the code in Part B of Section Two, which is shown in Figure 41. The purpose of this code is to pull user settings out of the EEPROM. This is done so that the program can function normally if the power is cycled on the LLI. To pull these data from of the EEPROM's memory, the line of code shown on line 47 and all odd lines of code are used. The EEPROM memory address, which is where the setting is stored is determined by the value of the variable "eeAddress", and the data stored at that address is copied to a variable, which in this case is the variable "Agent" and the address is '0'.



```
  eeAddress = 0;
  EEPROM.get(eeAddress,Agent);
  eeAddress = 4;
  EEPROM.get(eeAddress,Units);
  eeAddress = 8;
  EEPROM.get(eeAddress,radius);
  eeAddress = 12;
  EEPROM.get(eeAddress,thickness);
  eeAddress = 16;
  EEPROM.get(eeAddress,height);
  eeAddress = 20;
  EEPROM.get(eeAddress,Display);
  eeAddress = 24;
  EEPROM.get(eeAddress,Interval);
  eeAddress = 28;
  EEPROM.get(eeAddress,Recording);
  eeAddress = 32;
  EEPROM.get(eeAddress,Year);
  eeAddress = 34;
  EEPROM.get(eeAddress,Month);
  eeAddress = 36;
  EEPROM.get(eeAddress,Day);
  eeAddress = 38;
  EEPROM.get(eeAddress,Time);
}
```

**Figure 41: Section Two Part B of the Serial_Comms_Rev_5 code.**

Section Three of the code covers the "loop()" function, but before that section can be discussed, Section Four of the code must be covered. Section Four covers the "devicemenu()" function, which controls the functionality of the user-interface. The "devicemenu()" function is made up nine sections, the first covers the code that outputs the main menu for the user to view via the Arduino IDE's Serial Monitor or similar terminal emulator program. The first line of code, line 19, shown in Figure 42, declares the name of the function, which enables it to be called anywhere in the program. This will be discussed in more detail in Section Three of the code, the "loop" function. Code used on lines 22 to 29 and 31 transmits the user menu to the host device for the user to select from. Like the "setup" function, the "while" loop on lines 33 to 35 causes the program to pause until an input is received from the user.  Once that input is received, it is stored in the “menuChoice” variable, where it is used to bring up the submenu of the selected setting.

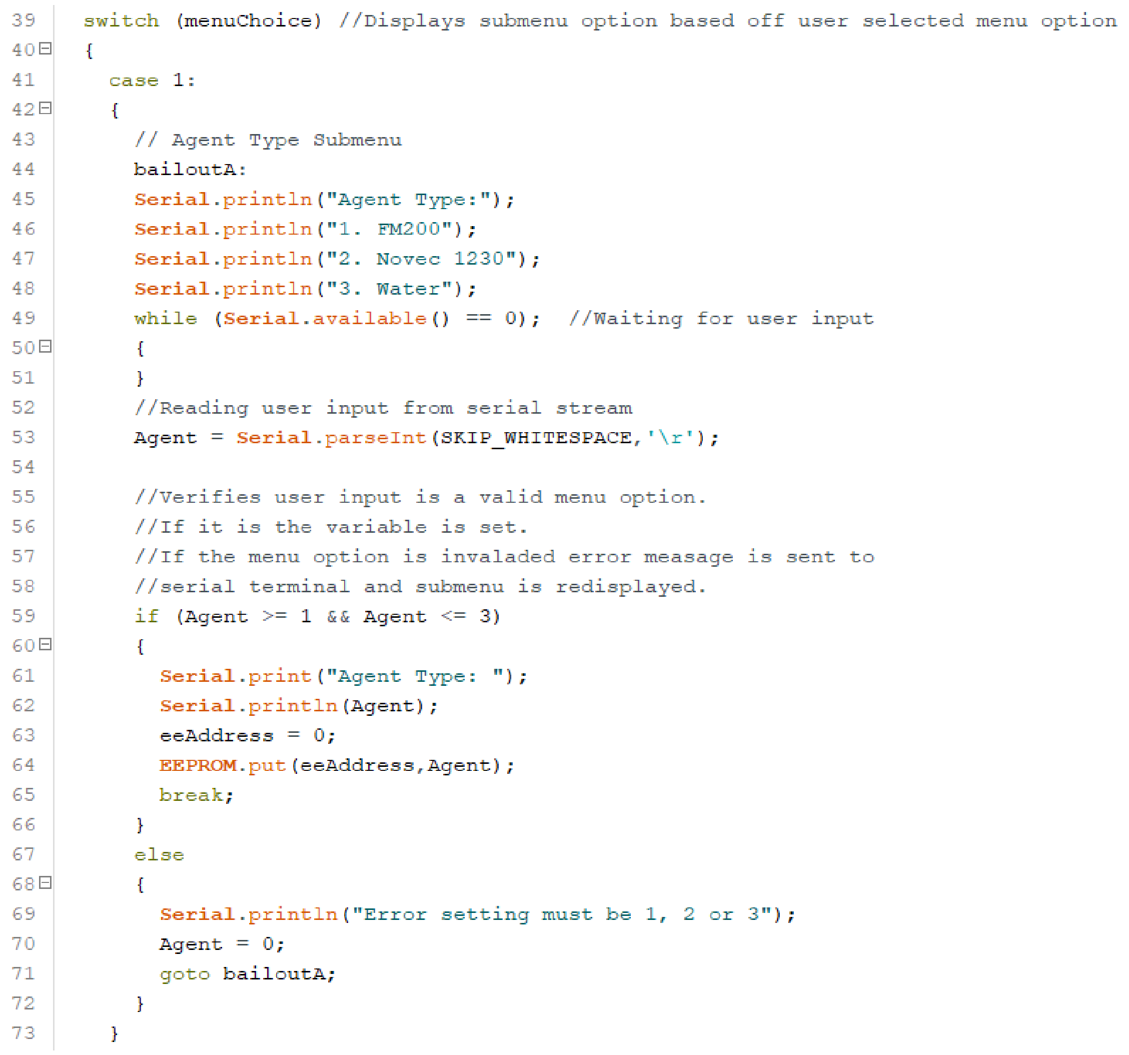


```
//User menu function
void devicemenu()
{
  //User menu
  Serial.println("1. Agent Type");
  Serial.println("2. Measurement Units");
  Serial.println("3. Storage Cylinder Dimensions");
  Serial.println("4. Display Data");
  Serial.println("5. Measurement Interval");
  Serial.println("6. Data Recording");
  Serial.println("7. Set Date");
  Serial.println("8. Set Time");

  Serial.println("Enter Menu Number? ");

  while (Serial.available() == 0) //Waiting for user input
  {
  }
  //Reading user input from serial stream
  int menuChoice = Serial.parseInt(SKIP_WHITESPACE,'\r');
```

**Figure 42:  Devicemenu Function Section A.**

The other eight sections of the “devicemenu” function cover the submenus for the eight different settings.  As in the “setup” function, a switch case control statement is used; however, in this case, there are nine cases that make up the switch case.  Section B of the function covers the declaration of the switch case, which is shown on line 39 of Figure 43 and the control statement’s first case lines 41 to 73.  The variable “menuChoice” determines which case of the control statement is processed.  So, if the user entered a ‘1’, the control statement will process the code in “case 1”, if it was a ‘2’, it would process the code in “case 2” (Section C), if the user input is an integer 1 through

8, the control statement will process the code in the corresponding case. The first case statement “case 1” allows the user to select the type of agent that is in the agent storage cylinder whether that is FM-200, Novec 1230 or Water. Like the previous section, lines 45 to 48 transmit the submenu options to the host device for the user to select from, and the “while” loop on lines 49 to 51 pauses the program until a response from the user is received. Again, that response is stored in a variable -- in this case the global variable “Agent”.

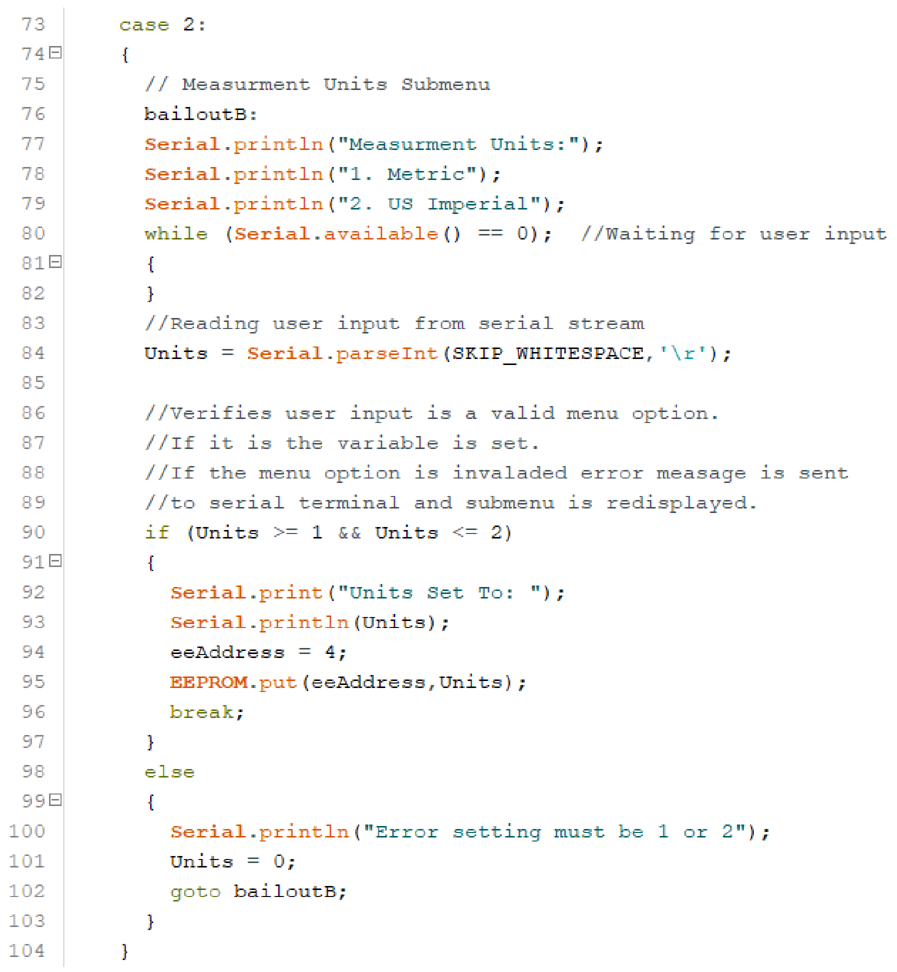


```
39   switch (menuChoice) //Displays submenu option based off user selected menu option
40   {
41     case 1:
42     {
43       // Agent Type Submenu
44       bailoutA:
45       Serial.println("Agent Type:");
46       Serial.println("1. FM200");
47       Serial.println("2. Novec 1230");
48       Serial.println("3. Water");
49       while (Serial.available() == 0);  //Waiting for user input
50       {
51       }
52       //Reading user input from serial stream
53       Agent = Serial.parseInt(SKIP_WHITESPACE,'\r');
54 
55       //Verifies user input is a valid menu option.
56       //If it is the variable is set.
57       //If the menu option is invaladed error measage is sent to
58       //serial terminal and submenu is redisplayed.
59       if (Agent >= 1 && Agent <= 3)
60       {
61         Serial.print("Agent Type: ");
62         Serial.println(Agent);
63         eeAddress = 0;
64         EEPROM.put(eeAddress,Agent);
65         break;
66       }
67       else
68       {
69         Serial.println("Error setting must be 1, 2 or 3");
70         Agent = 0;
71         goto bailoutA;
72       }
73     }
```

**Figure 43: Devicemenu Function Section B.**

As the comments on lines 54 to 56 denote, the value of “Agent” must be an integer between 1 and 3.  If it is not, an error message will be displayed, and the program will retransmit the Agent submenu.  This functionality is controlled by the “if else” control statement shown on lines 57 to 70.  Line 57 verifies that the value stored in the “Agent” variable is greater than or equal to 1 or less than or equal to 3.  If it is, lines 59 to 63 are processed, which stores the selected agent setting into address 0 of the EEPROM and transmits back to the user what setting was entered.  If it is not an integer between 1 and 3, the code in the “else” control statements is processed in lines 67 to 69.  This code transmits an error message to the user (line 67), sets the “Agent” variable back to zero (line 68) and directs the microcontroller back to the beginning of the case statement.  This is done using the control statement “goto bailoutA” on line 69, which redirects the program to line 44.  Those “goto” control statements are used in each of the different switch cases to control user input.

Section C of the “devicemenu” function covers the second case of the switch control statement, which handles the submenu for setting what measurement units (Metric or US Imperial) are used when the mass of the agent and temperature are displayed on the LCD and what is recorded in the history log when data recording is enabled.  This section of code covers lines 73 to 104, as shown in Figure 44.

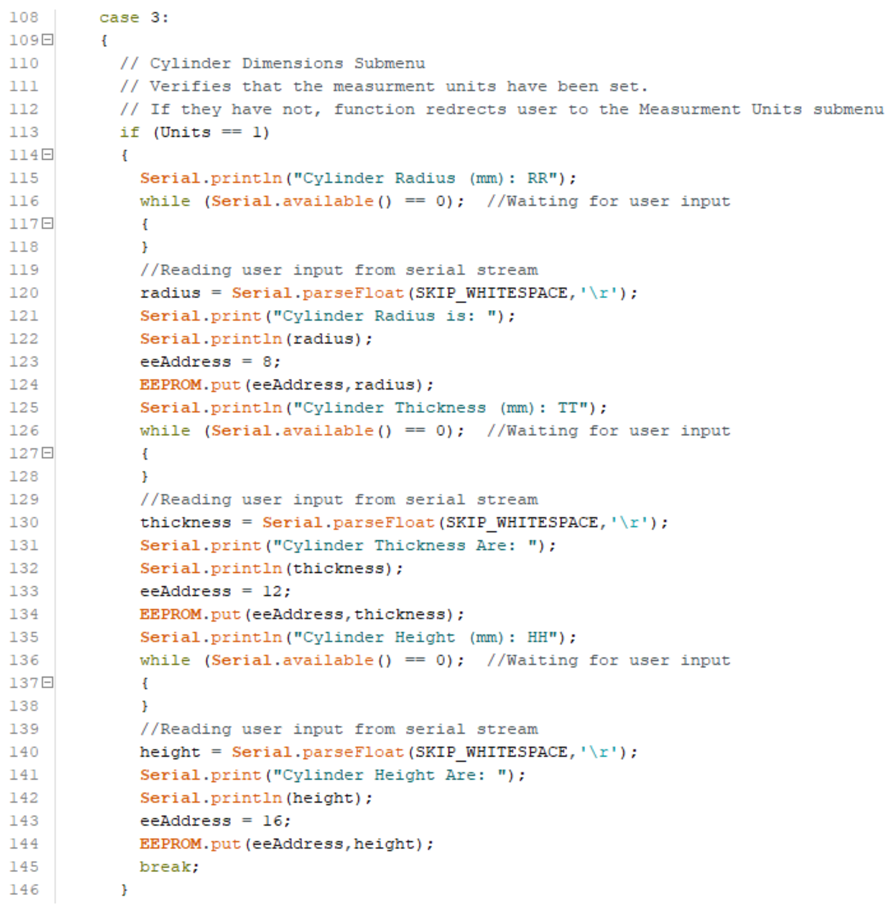


```
case 2:
{
  // Measurment Units Submenu
  bailoutB:
  Serial.println("Measurment Units:");
  Serial.println("1. Metric");
  Serial.println("2. US Imperial");
  while (Serial.available() == 0);  //Waiting for user input
  {
  }
  //Reading user input from serial stream
  Units = Serial.parseInt(SKIP_WHITESPACE,'\r');

  //Verifies user input is a valid menu option.
  //If it is the variable is set.
  //If the menu option is invaladed error measage is sent
  //to serial terminal and submenu is redisplayed.
  if (Units >= 1 && Units <= 2)
  {
    Serial.print("Units Set To: ");
    Serial.println(Units);
    eeAddress = 4;
    EEPROM.put(eeAddress,Units);
    break;
  }
  else
  {
    Serial.println("Error setting must be 1 or 2");
    Units = 0;
    goto bailoutB;
  }
}
```

**Figure 44: Devicemenu Function Section C.**

After a quick review of this section, the reader should notice some familiar lines of code and that the overall structure of this section and all subsequent sections of the "devicemenu" function's switch case control statement are the same. The first few lines transmit the submenu to the user terminal, the program then waits for a user input, that user input is then stored in a variable -- in this case the global variable "Units" -- the contents of that variable are then checked, and if the contents are within the requirements (in this case either a '1' or '2'), the setting is saved to the EEPROM. Otherwise, the variable is set back to zero, the program transmits an error message and then re-transmits the submenu.

As noted, the overall structure of these different “case” statements is relatively the same; the program transmits the submenu, waits for user input, stores that input…. etc. In the case of Section D, shown in Figures 45 and 46, this structure is repeated using an “if else” control statement.

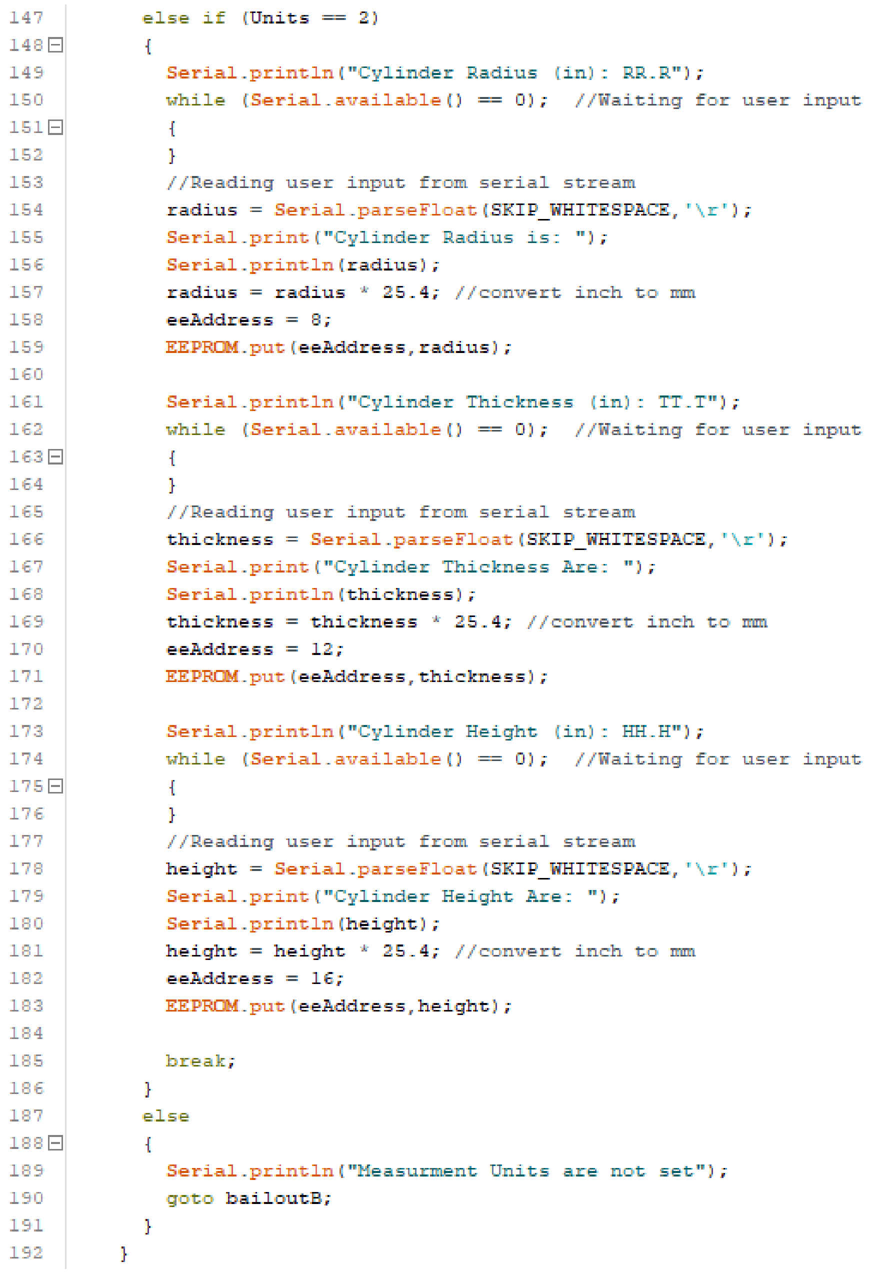


```
108     case 3:
109     {
110       // Cylinder Dimensions Submenu
111       // Verifies that the measurment units have been set.
112       // If they have not, function redrects user to the Measurment Units submenu
113       if (Units == 1)
114       {
115         Serial.println("Cylinder Radius (mm): RR");
116         while (Serial.available() == 0);  //Waiting for user input
117         {
118         }
119         //Reading user input from serial stream
120         radius = Serial.parseFloat(SKIP_WHITESPACE,'\r');
121         Serial.print("Cylinder Radius is: ");
122         Serial.println(radius);
123         eeAddress = 8;
124         EEPROM.put(eeAddress,radius);
125         Serial.println("Cylinder Thickness (mm): TT");
126         while (Serial.available() == 0);  //Waiting for user input
127         {
128         }
129         //Reading user input from serial stream
130         thickness = Serial.parseFloat(SKIP_WHITESPACE,'\r');
131         Serial.print("Cylinder Thickness Are: ");
132         Serial.println(thickness);
133         eeAddress = 12;
134         EEPROM.put(eeAddress,thickness);
135         Serial.println("Cylinder Height (mm): HH");
136         while (Serial.available() == 0);  //Waiting for user input
137         {
138         }
139         //Reading user input from serial stream
140         height = Serial.parseFloat(SKIP_WHITESPACE,'\r');
141         Serial.print("Cylinder Height Are: ");
142         Serial.println(height);
143         eeAddress = 16;
144         EEPROM.put(eeAddress,height);
145         break;
146       }
```

**Figure 45: Devicemenu Function Section D.**

This control statement is used to determine which measurement units were selected by the user and based on that selection, the code directs the user to enter the dimensions of the storage cylinder in either millimeters or inches. The code shown in Figure 45 is processed if the user selected metric units, and the code in Figure 46 on lines 147 to 185 is processed if the user selected SI units. The last few lines of code -- lines 187 to 190 -- are processed only if the user did not set the units setting prior to selecting the “Cylinder

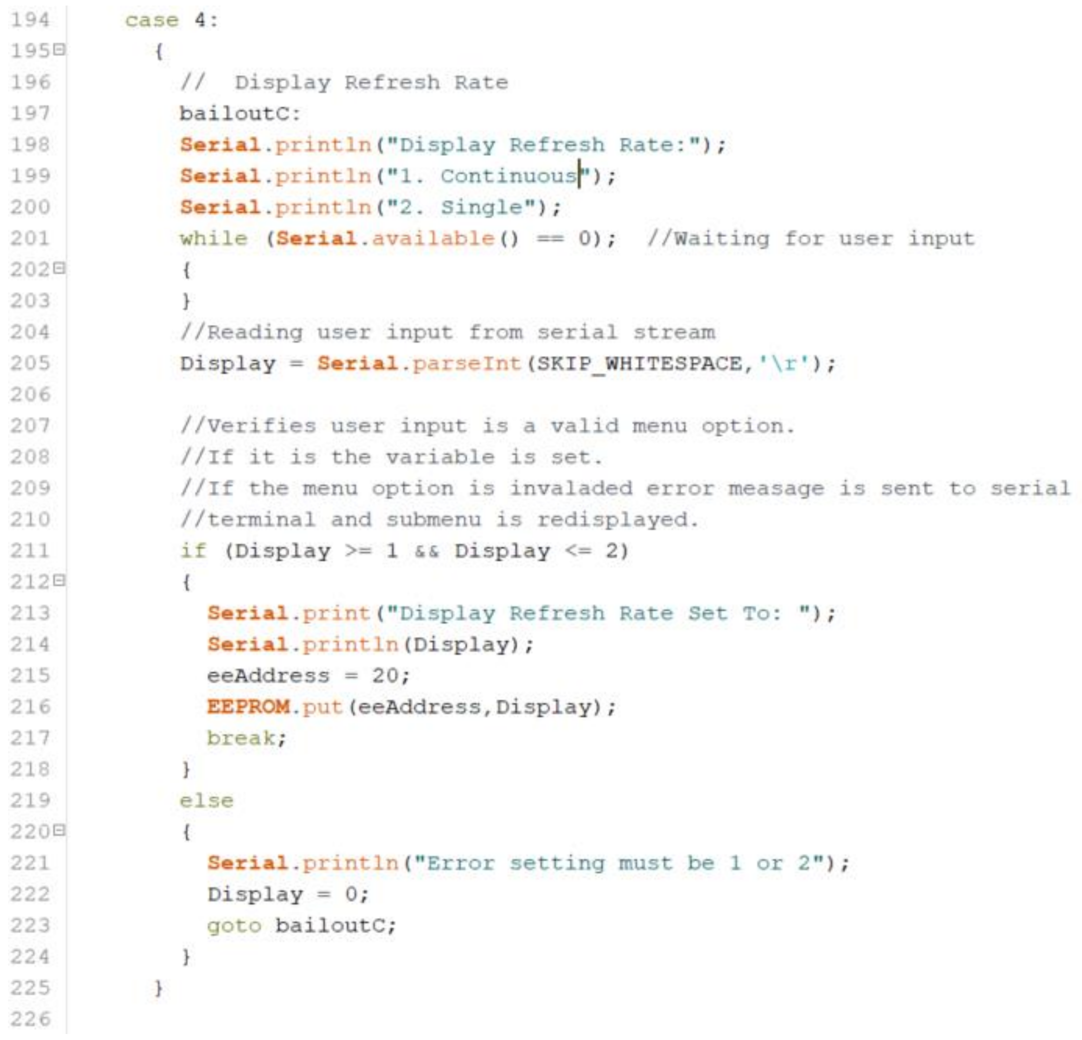
Dimensions” submenu. In this case, the “goto” statement redirects the program to the “Measurement Units” submenu.



```
    else if (Units == 2)
    {
      Serial.println("Cylinder Radius (in): RR.R");
      while (Serial.available() == 0);  //Waiting for user input
      {
      }
      //Reading user input from serial stream
      radius = Serial.parseFloat(SKIP_WHITESPACE,'\r');
      Serial.print("Cylinder Radius is: ");
      Serial.println(radius);
      radius = radius * 25.4; //convert inch to mm
      eeAddress = 8;
      EEPROM.put(eeAddress,radius);

      Serial.println("Cylinder Thickness (in): TT.T");
      while (Serial.available() == 0);  //Waiting for user input
      {
      }
      //Reading user input from serial stream
      thickness = Serial.parseFloat(SKIP_WHITESPACE,'\r');
      Serial.print("Cylinder Thickness Are: ");
      Serial.println(thickness);
      thickness = thickness * 25.4; //convert inch to mm
      eeAddress = 12;
      EEPROM.put(eeAddress,thickness);

      Serial.println("Cylinder Height (in): HH.H");
      while (Serial.available() == 0);  //Waiting for user input
      {
      }
      //Reading user input from serial stream
      height = Serial.parseFloat(SKIP_WHITESPACE,'\r');
      Serial.print("Cylinder Height Are: ");
      Serial.println(height);
      height = height * 25.4; //convert inch to mm
      eeAddress = 16;
      EEPROM.put(eeAddress,height);

      break;
    }
    else
    {
      Serial.println("Measurment Units are not set");
      goto bailoutB;
    }
  }
```

**Figure 46: Devicemenu Function Section D Continued.**

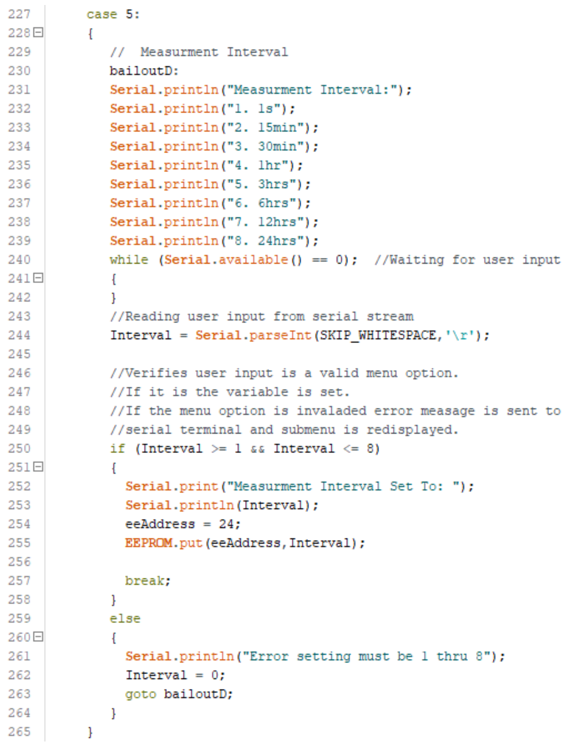
The next three sections of code -- sections E, F and G -- are shown in Figures 47, 48 and 49. The overall structure of these sections of code is the same as Sections B and C, with no real differences like in Section D, except for what global variable is used to store the input from the user. Section E of the code, shown in Figure 47, covers the submenu for setting the display's refresh rate. The two available options are "Continuous" and "Single", which will be discussed later in this report.



```
    case 4:
      {
        //  Display Refresh Rate
        bailoutC:
        Serial.println("Display Refresh Rate:");
        Serial.println("1. Continuous");
        Serial.println("2. Single");
        while (Serial.available() == 0);  //Waiting for user input
        {
        }
        //Reading user input from serial stream
        Display = Serial.parseInt(SKIP_WHITESPACE,'\r');

        //Verifies user input is a valid menu option.
        //If it is the variable is set.
        //If the menu option is invaladed error measage is sent to serial
        //terminal and submenu is redisplayed.
        if (Display >= 1 && Display <= 2)
        {
          Serial.print("Display Refresh Rate Set To: ");
          Serial.println(Display);
          eeAddress = 20;
          EEPROM.put(eeAddress,Display);
          break;
        }
        else
        {
          Serial.println("Error setting must be 1 or 2");
          Display = 0;
          goto bailoutC;
        }
      }

```

**Figure 47: Devicemenu Function Section E.**

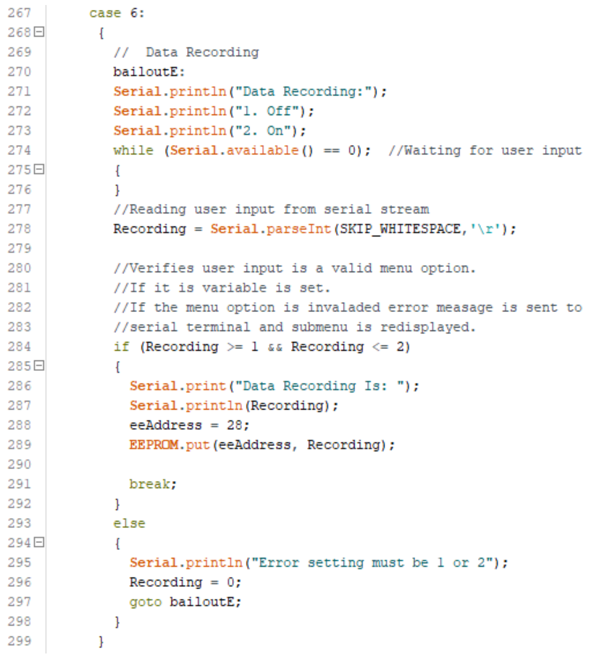
Section F, which is shown in Figure 48, covers the "Measurement Interval" setting submenu. This setting allows the user to set the data recording rate, which tells the program when to save mass data to the history log (EEPROM). As shown on lines 232 to 239, the options are once every second, every 15 or 30 minutes, or every 1, 3, 6, 12 or 24 hours.



```
case 5:
{
  //  Measurment Interval
  bailoutD:
  Serial.println("Measurment Interval:");
  Serial.println("1. 1s");
  Serial.println("2. 15min");
  Serial.println("3. 30min");
  Serial.println("4. 1hr");
  Serial.println("5. 3hrs");
  Serial.println("6. 6hrs");
  Serial.println("7. 12hrs");
  Serial.println("8. 24hrs");
  while (Serial.available() == 0);  //Waiting for user input
  {
  }
  //Reading user input from serial stream
  Interval = Serial.parseInt(SKIP_WHITESPACE,'\r');

  //Verifies user input is a valid menu option.
  //If it is the variable is set.
  //If the menu option is invaladed error measage is sent to
  //serial terminal and submenu is redisplayed.
  if (Interval >= 1 && Interval <= 8)
  {
    Serial.print("Measurment Interval Set To: ");
    Serial.println(Interval);
    eeAddress = 24;
    EEPROM.put(eeAddress,Interval);

    break;
  }
  else
  {
    Serial.println("Error setting must be 1 thru 8");
    Interval = 0;
    goto bailoutD;
  }
}
```

**Figure 48:  Devicemenu Function Section F.**

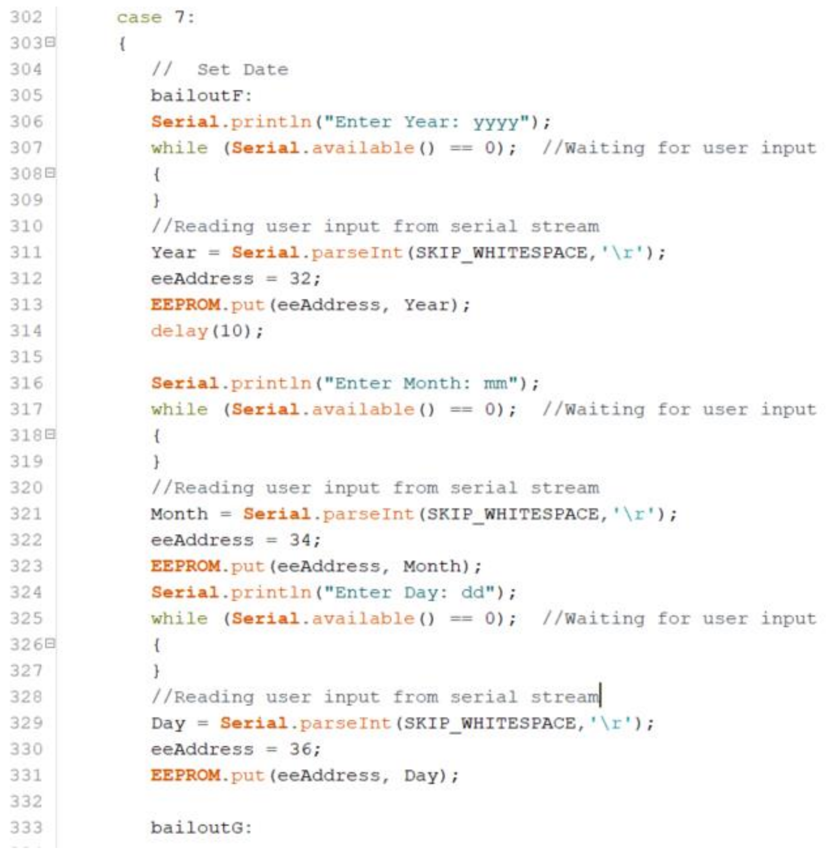
Section G, of the function shown in Figure 49, handles the submenu for the “Data Recording” setting, which enables or disables data recording.



```
case 6:
 {
  //  Data Recording
  bailoutE:
  Serial.println("Data Recording:");
  Serial.println("1. Off");
  Serial.println("2. On");
  while (Serial.available() == 0);  //Waiting for user input
  {
  }
  //Reading user input from serial stream
  Recording = Serial.parseInt(SKIP_WHITESPACE,'\r');

  //Verifies user input is a valid menu option.
  //If it is variable is set.
  //If the menu option is invaladed error measage is sent to
  //serial terminal and submenu is redisplayed.
  if (Recording >= 1 && Recording <= 2)
  {
    Serial.print("Data Recording Is: ");
    Serial.println(Recording);
    eeAddress = 28;
    EEPROM.put(eeAddress, Recording);

    break;
  }
  else
  {
    Serial.println("Error setting must be 1 or 2");
    Recording = 0;
    goto bailoutE;
  }
 }
```

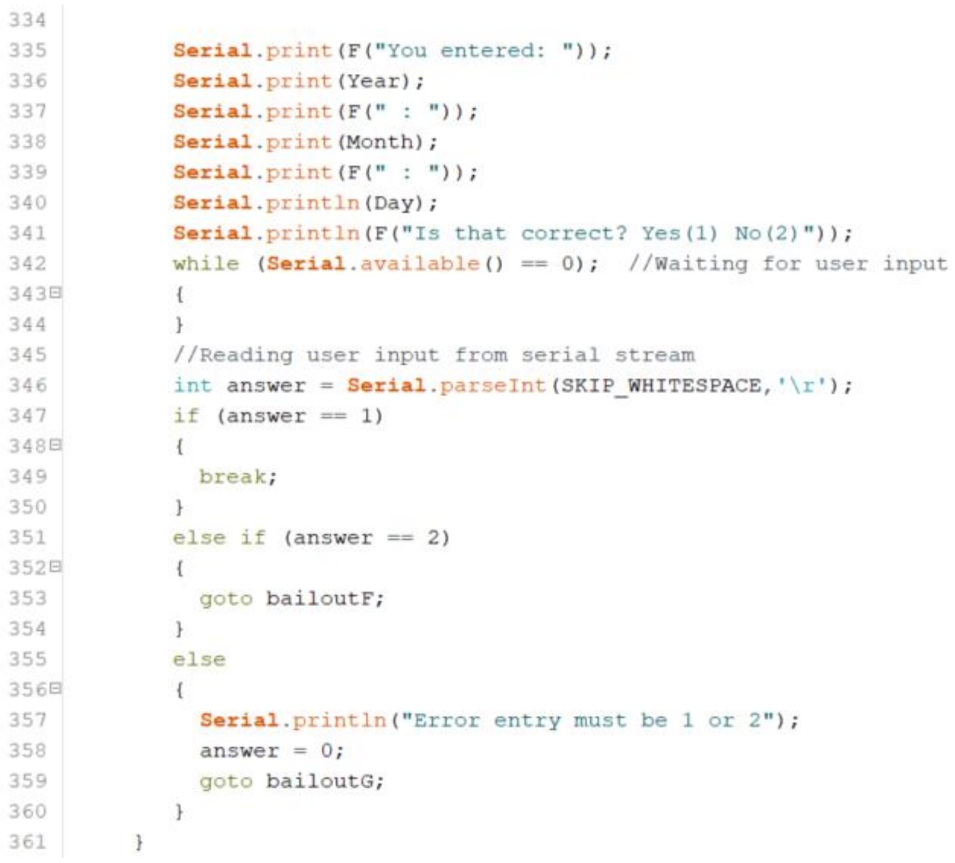
**Figure 49:  Devicemenu Function Section G.**

The last two sections of the “devicemenu” function are Sections H (Figures 50 and 51) and I (Figure 52). Section H of the function controls the submenu for setting the date in numerical form with a four-digit year, two-digit month and two-digit day. These settings are stored in EEPROM memory locations 32, 34 and 36. Section I controls the submenu for setting the time of day using the 24-hour format. The ability to set these two settings were added to the device menu so that the program could track the date and time to put timestamps on each data point that is recorded to the history log. However, this functionality was not fully implemented in the final revision of the LLI firmware due to time constraints. Additionally, Section I also contains the default case shown on lines 379 to 383. Like the default cases in previous sections, this case is only processed if the user input is not valid (any value that is not an integer between ‘1’ and ‘8’).



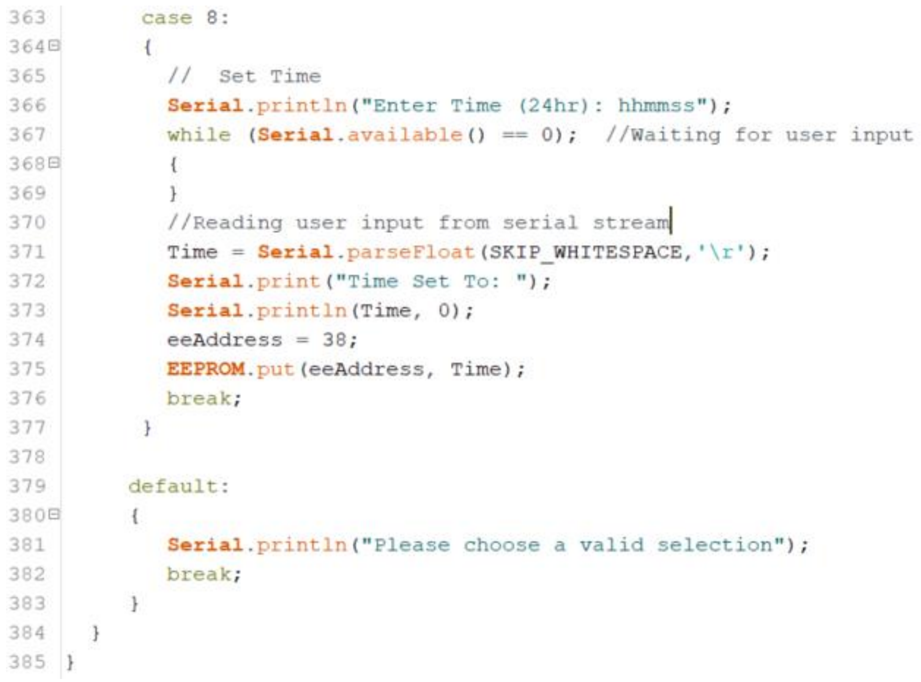
```
    case 7:
    {
       //  Set Date
       bailoutF:
       Serial.println("Enter Year: yyyy");
       while (Serial.available() == 0);  //Waiting for user input
       {
       }
       //Reading user input from serial stream
       Year = Serial.parseInt(SKIP_WHITESPACE,'\r');
       eeAddress = 32;
       EEPROM.put(eeAddress, Year);
       delay(10);

       Serial.println("Enter Month: mm");
       while (Serial.available() == 0);  //Waiting for user input
       {
       }
       //Reading user input from serial stream
       Month = Serial.parseInt(SKIP_WHITESPACE,'\r');
       eeAddress = 34;
       EEPROM.put(eeAddress, Month);
       Serial.println("Enter Day: dd");
       while (Serial.available() == 0);  //Waiting for user input
       {
       }
       //Reading user input from serial stream
       Day = Serial.parseInt(SKIP_WHITESPACE,'\r');
       eeAddress = 36;
       EEPROM.put(eeAddress, Day);

       bailoutG:
```

**Figure 50: Devicemenu Function Section H.**



```

        Serial.print(F("You entered: "));
        Serial.print(Year);
        Serial.print(F(" : "));
        Serial.print(Month);
        Serial.print(F(" : "));
        Serial.println(Day);
        Serial.println(F("Is that correct? Yes(1) No(2)"));
        while (Serial.available() == 0);  //Waiting for user input
        {
        }
        //Reading user input from serial stream
        int answer = Serial.parseInt(SKIP_WHITESPACE,'\r');
        if (answer == 1)
        {
          break;
        }
        else if (answer == 2)
        {
          goto bailoutF;
        }
        else
        {
          Serial.println("Error entry must be 1 or 2");
          answer = 0;
          goto bailoutG;
        }
      }
```

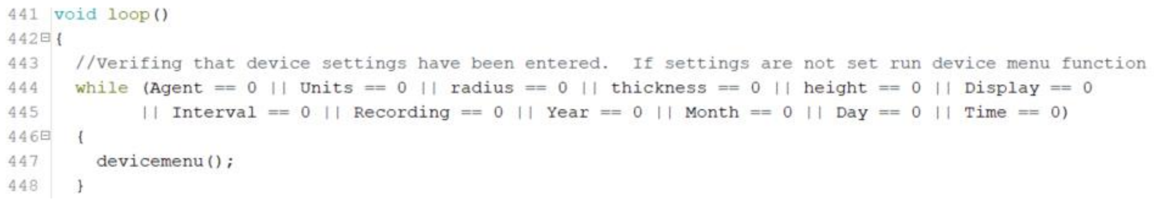
**Figure 51: Devicemenu Function Section H Continued.**



```
       case 8:
       {
         //  Set Time
         Serial.println("Enter Time (24hr): hhmmss");
         while (Serial.available() == 0);  //Waiting for user input
         {
         }
         //Reading user input from serial stream
         Time = Serial.parseFloat(SKIP_WHITESPACE,'\r');
         Serial.print("Time Set To: ");
         Serial.println(Time, 0);
         eeAddress = 38;
         EEPROM.put(eeAddress, Time);
         break;
       }

      default:
      {
         Serial.println("Please choose a valid selection");
         break;
      }
   }
}
```

**Figure 52: Devicemenu Function Section I.**

The last section of the "Serial_Comms_Rev_5" code covers the "loop" function. As stated earlier, the "loop" function is the only function that the microcontroller will run indefinitely. The only portion of this section that is of any importance is the "while" loop on lines 443 to 448, shown in Figure 53. Upon start-up, this bit of code verifies that all of the user setting have been set. If there are any settings that do not set, the code calls the "devicemenue()" function on line 447 so that the user can enter the appropriate device settings. Additionally, once all the device settings have been set, this "while" loop will stop processing and the microcontroller will continue running the programing contained in the "loop" function. The remaining code of this function was used to verify that the user settings were stored correctly in the EEPROM. So the code pulls the individual user settings from memory and then transmits the values to the terminal window on the host device. The code is shown in Appendix J.

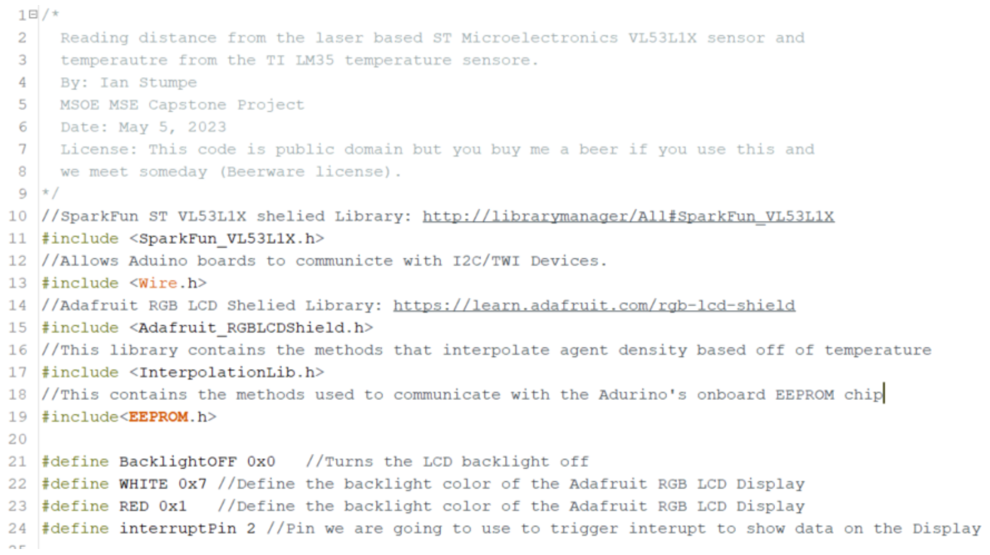


```
441 void loop()
442 {
443   //Verifing that device settings have been entered.  If settings are not set run device menu function
444   while (Agent == 0 || Units == 0 || radius == 0 || thickness == 0 || height == 0 || Display == 0
445         || Interval == 0 || Recording == 0 || Year == 0 || Month == 0 || Day == 0 || Time == 0)
446   {
447     devicemenu();
448   }
```

**Figure 53: Loop Function While Loop.**

Now that the code for the user-interface/device menu was completed and functioned as intended, the next step was to integrate the user-interface code with the "LLI Firmware Rev 2" code. As indicated previously, the second revision of the LLI firmware added the necessary code to convert the distance measurement from the TOF to volume and then used the ambient temperature to interpolate the agent's density to calculate the mass of the agent. Integration of the two codes was not as simple as just moving the important code from the "Serial_Comms_Rev_5" program, such as the "devicemenu()" function, to the "LLI Fimware Rev 2.0" code. It also entailed adding the necessary code that implemented those user settings. For instance, if the user selected FM-200, the program would then calculate the mass of the agent using the density of FM-200. Like the development of the user-interface code, this integration was an iterative process. New code would be added that implemented new functionality, and that new function would be tested and verified to ensure that it worked as intended. This iterative process led to eight revisions of the new firmware with each revision adding new functionality, culminating in the final LLI firmware version shown in Appendix K, "LLI Firmware with User Interface Rev 8". Additionally, there is a flowchart of this code shown in Appendix L.

At first glance, when compared with the previous versions of the firmware, this version looks completely different. However, some if not a lot of the code should look familiar, because it was used in those previous versions. Like the previous versions of the firmware, the first section contains the program header, software libraries and definitions (constant values) used in the program. See Section One of the firmware shown in Figure 54.

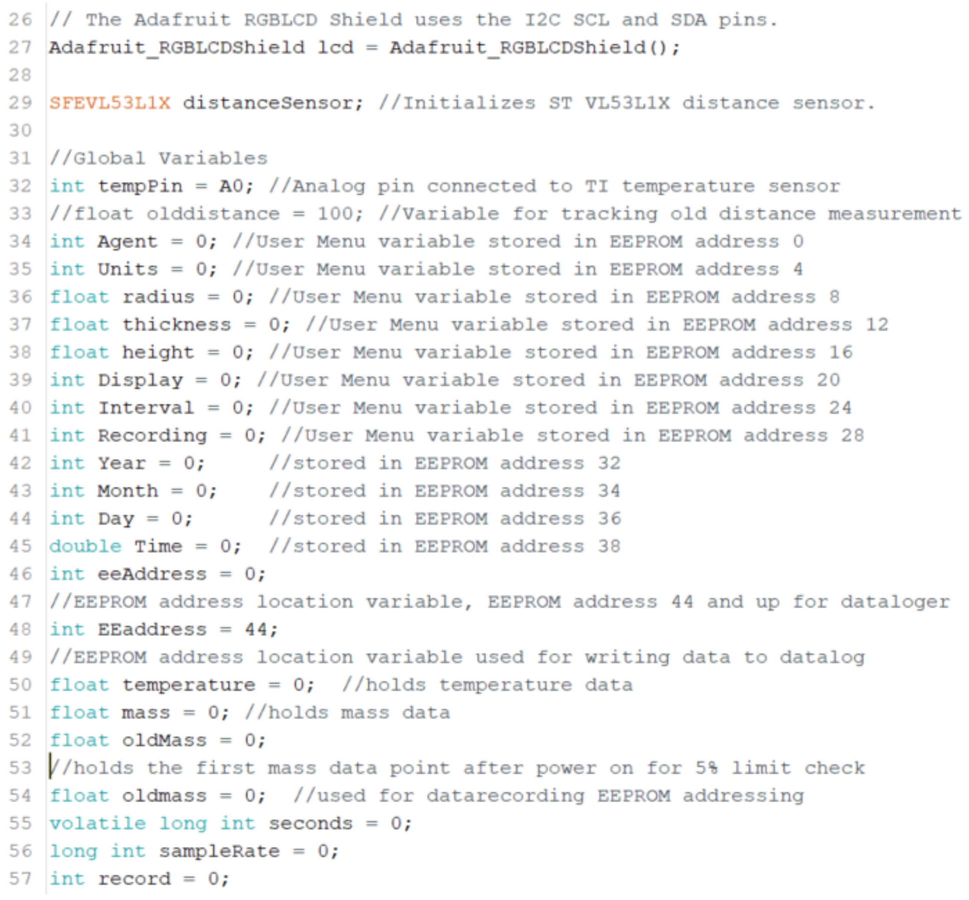


```
1  /*
2    Reading distance from the laser based ST Microelectronics VL53L1X sensor and
3    temperautre from the TI LM35 temperature sensore.
4    By: Ian Stumpe
5    MSOE MSE Capstone Project
6    Date: May 5, 2023
7    License: This code is public domain but you buy me a beer if you use this and
8    we meet someday (Beerware license).
9  */
10 //SparkFun ST VL53L1X shelied Library: http://librarymanager/All#SparkFun_VL53L1X
11 #include <SparkFun_VL53L1X.h>
12 //Allows Aduino boards to communicte with I2C/TWI Devices.
13 #include <Wire.h>
14 //Adafruit RGB LCD Shelied Library: https://learn.adafruit.com/rgb-lcd-shield
15 #include <Adafruit_RGBLCDShield.h>
16 //This library contains the methods that interpolate agent density based off of temperature
17 #include <InterpolationLib.h>
18 //This contains the methods used to communicate with the Adurino's onboard EEPROM chip
19 #include<EEPROM.h>
20
21 #define BacklightOFF 0x0   //Turns the LCD backlight off
22 #define WHITE 0x7 //Define the backlight color of the Adafruit RGB LCD Display
23 #define RED 0x1   //Define the backlight color of the Adafruit RGB LCD Display
24 #define interruptPin 2 //Pin we are going to use to trigger interupt to show data on the Display
```

**Figure 54: LLI Firmware with User-Interface Section One.**

The only new items that have not been covered yet in previous sections are the "#define" on lines 21 and 22, which are used to turn the display backlight off or turn it red, and the "interruptPin 2" "#define" on line 24. This particular "#define" is very important and will be discussed in a later section.

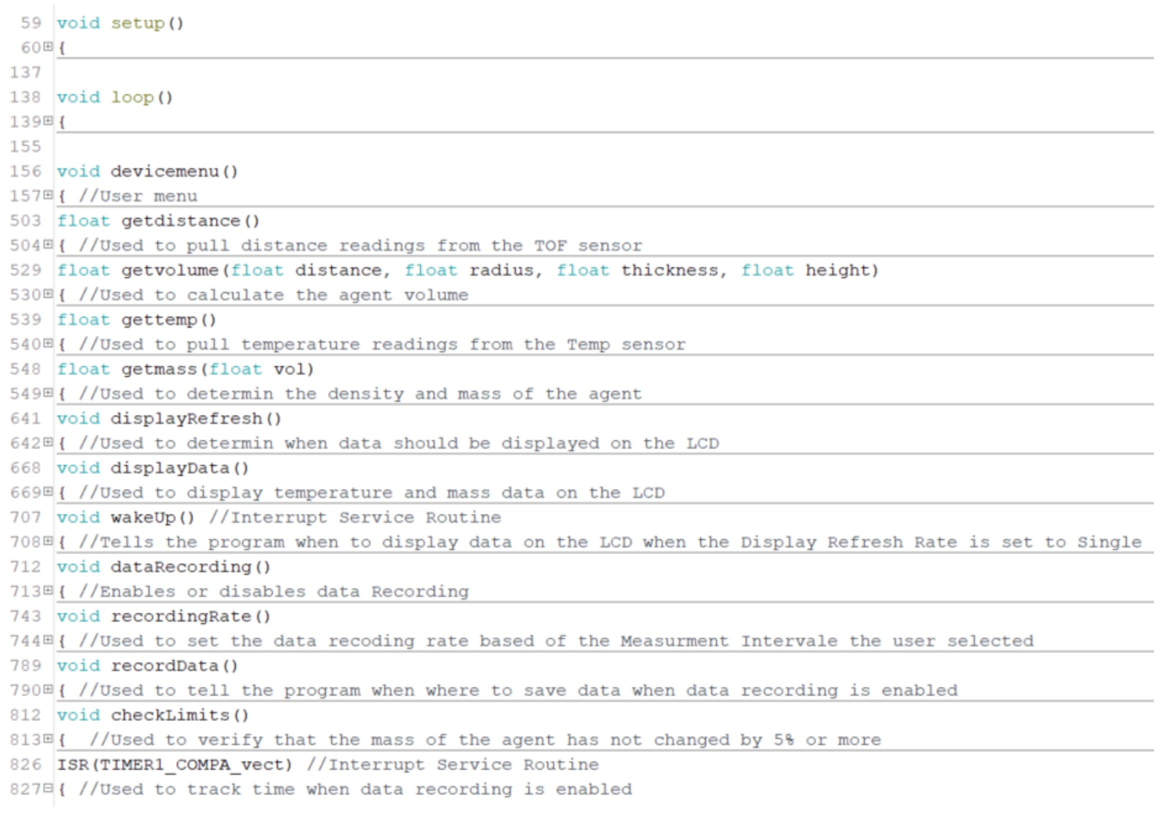
Section Two of the new firmware is shown in Figure 55. This section covers the initialization of the LCD display and the VL53L1X TOF sensor, and the global variables that are used. Most of the variables should look familiar because of their use in previous firmware versions. However, there are a few new ones that are located on line 47 and lines 50 to 54. These new global variables will be discussed in greater detail when the functions that they pertain to are explained.



```
// The Adafruit RGBLCD Shield uses the I2C SCL and SDA pins.
Adafruit_RGBLCDShield lcd = Adafruit_RGBLCDShield();

SFEVL53L1X distanceSensor; //Initializes ST VL53L1X distance sensor.

//Global Variables
int tempPin = A0; //Analog pin connected to TI temperature sensor
//float olddistance = 100; //Variable for tracking old distance measurement
int Agent = 0; //User Menu variable stored in EEPROM address 0
int Units = 0; //User Menu variable stored in EEPROM address 4
float radius = 0; //User Menu variable stored in EEPROM address 8
float thickness = 0; //User Menu variable stored in EEPROM address 12
float height = 0; //User Menu variable stored in EEPROM address 16
int Display = 0; //User Menu variable stored in EEPROM address 20
int Interval = 0; //User Menu variable stored in EEPROM address 24
int Recording = 0; //User Menu variable stored in EEPROM address 28
int Year = 0;      //stored in EEPROM address 32
int Month = 0;     //stored in EEPROM address 34
int Day = 0;       //stored in EEPROM address 36
double Time = 0;   //stored in EEPROM address 38
int eeAddress = 0;
//EEPROM address location variable, EEPROM address 44 and up for dataloger
int EEaddress = 44;
//EEPROM address location variable used for writing data to datalog
float temperature = 0;  //holds temperature data
float mass = 0; //holds mass data
float oldMass = 0;
//holds the first mass data point after power on for 5% limit check
float oldmass = 0;  //used for datarecording EEPROM addressing
volatile long int seconds = 0;
long int sampleRate = 0;
int record = 0;
```

**Figure 55: LLI Firmware with User-Interface Section Two.**

In regard to functions, there are quite a few new functions that were added, not to just add new functionality related to the addition of the user-interface, but also to streamline the program and to make it easier to follow. Figure 56 shows all the new functions along with some familiar ones. The Arduino defined functions “setup()” and “loop()” will be discussed last because they both call/command the program to process the code in the other functions listed below. Additionally, the “devicemenu()” function will not be discussed, because it is the same function that was used in the “Serial Comms Rev 5” firmware.

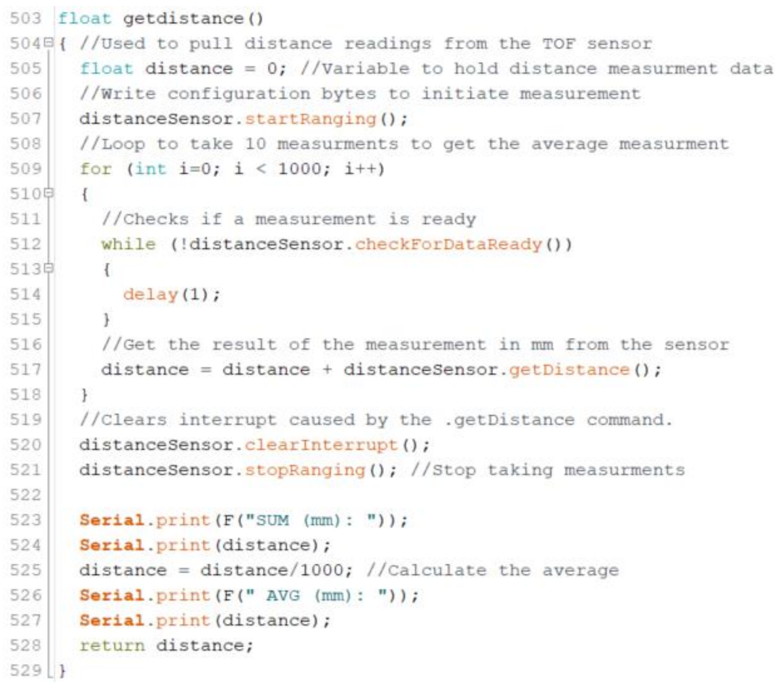


```
void setup()
{

void loop()
{

void devicemenu()
{ //User menu
float getdistance()
{ //Used to pull distance readings from the TOF sensor
float getvolume(float distance, float radius, float thickness, float height)
{ //Used to calculate the agent volume
float gettemp()
{ //Used to pull temperature readings from the Temp sensor
float getmass(float vol)
{ //Used to determin the density and mass of the agent
void displayRefresh()
{ //Used to determin when data should be displayed on the LCD
void displayData()
{ //Used to display temperature and mass data on the LCD
void wakeUp() //Interrupt Service Routine
{ //Tells the program when to display data on the LCD when the Display Refresh Rate is set to Single
void dataRecording()
{ //Enables or disables data Recording
void recordingRate()
{ //Used to set the data recoding rate based of the Measurment Intervale the user selected
void recordData()
{ //Used to tell the program when where to save data when data recording is enabled
void checkLimits()
{  //Used to verify that the mass of the agent has not changed by 5% or more
ISR(TIMER1_COMPA_vect) //Interrupt Service Routine
{ //Used to track time when data recording is enabled
```

**Figure 56: LLI Firmware with User-Interface Defined Functions.**

Regarding the “wakeup()” and “ISR(TIMER1_COMPA_vect)” functions, these two functions are a special type of function called an Interrupt Service Routine (ISR). An interrupt is a mechanism by which the microcontroller suspends normal execution of the program to execute the higher priority code located in the ISR [49]. There are two types of interrupts -- hardware interrupts and software interrupts. A hardware interrupt happens when an external event occurs like an external interrupt pin changes state from LOW to HIGH or HIGH to LOW. The “wakeup()” ISR is a hardware type interrupt, because it gets executed when the state of “#define interuptPIN 2” changes from HIGH to LOW. The “ISR(TIMER1_COMPA_vect)” ISR is an example of a software interrupt, which happens according to the code in the program [49]. This ISR is executed based on when the counters in Timer 1 of the microcontroller match. A timer or counter is a special piece of hardware inside many microcontrollers. The ATmega328P has three timer/counters. The purpose of these timer/counters is to count, up or down, depending on their configuration [50]. These timer/counter configurations will be discussed further when the "datarecording()" function is explained.

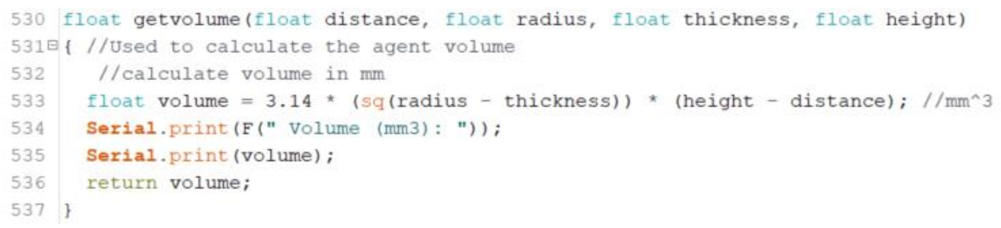
The first function that will be discussed is the "getdistance()" function shown in Figure 57. Unlike the previous functions that have been discussed, this is the first function that returns a variable. Because the function is returning a variable, the type of variable must be declared in the function declaration. The distance measurement is a float (a number with a decimal point), which is the value the function needs to return (output), so the word "float" is substituted in the place of the word "void", which means that the function does not return a variable. Beyond that, the code that follows the function declaration (line 503) on lines 505 to 527 is the same code used in "LLI Firmware Rev 1.0" and LLI Firmware Rev 2.0" to get a distance measurement from the TOF sensor. However, there is one modification that was made with this iteration of the firmware. That modification is in the code used to transmit data to the host device, the "Serial.print" commands. The letter F was added after the first parentheses (see example in line 523) to force the data to be stored in the microcontroller's flash memory instead of its dynamic memory where it stores global variables [51]. If this were not done, all the character data in the "Serial.print" commands would fill the microcontroller's dynamic memory and cause the program to not function correctly. The other thing of note in this function is in line 528. Because the function needs to return the distance measured, the code "return" followed by the variable that is needed is used.



```
float getdistance()
{ //Used to pull distance readings from the TOF sensor
  float distance = 0; //Variable to hold distance measurment data
  //Write configuration bytes to initiate measurement
  distanceSensor.startRanging();
  //Loop to take 10 measurments to get the average measurment
  for (int i=0; i < 1000; i++)
  {
    //Checks if a measurement is ready
    while (!distanceSensor.checkForDataReady())
    {
      delay(1);
    }
    //Get the result of the measurement in mm from the sensor
    distance = distance + distanceSensor.getDistance();
  }
  //Clears interrupt caused by the .getDistance command.
  distanceSensor.clearInterrupt();
  distanceSensor.stopRanging(); //Stop taking measurments

  Serial.print(F("SUM (mm): "));
  Serial.print(distance);
  distance = distance/1000; //Calculate the average
  Serial.print(F(" AVG (mm): "));
  Serial.print(distance);
  return distance;
}
```

**Figure 57: LLI Firmware with User-Interface getdistance() Function.**

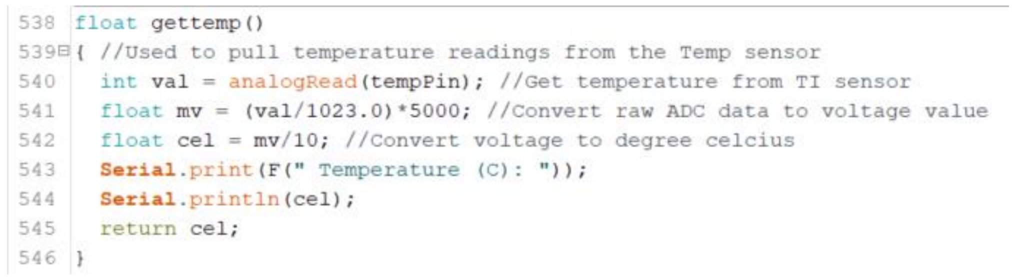
The next function to be discussed is the “getvolume()” function shown in Figure 58. As the name implies, this function is responsible for calculating the volume of the agent and returning that value. Like the previous function, the float data type is used in the function declaration. However, unlike the previous function, this function needs to pass on additional parameters to its code for that volume calculation to work. Those parameters are the distance (from the previous function), the radius, thickness, and height of the cylinder that the user entered. So those variables must be declared in between the parentheses, as shown on line 530. Subsequently, those same variables are used on line 533 to calculate the volume, which then returned on line 536.



```
530 float getvolume(float distance, float radius, float thickness, float height)
531 { //Used to calculate the agent volume
532     //calculate volume in mm
533   float volume = 3.14 * (sq(radius - thickness)) * (height - distance); //mm^3
534   Serial.print(F(" Volume (mm3): "));
535   Serial.print(volume);
536   return volume;
537 }
```

**Figure 58: LLI Firmware with User-Interface getvolume() Function.**

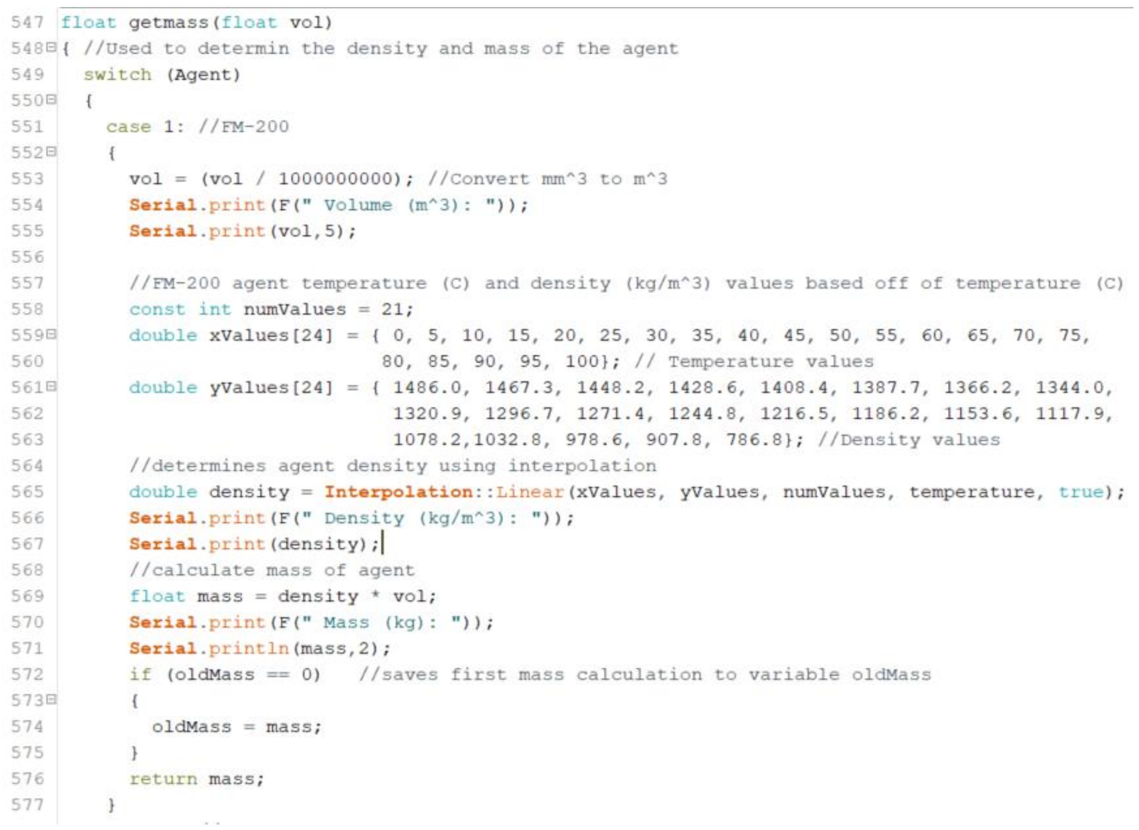
So, there is a function for getting the distance from the TOF sensor. What about getting the ambient temperature from the temperature sensor? That function, “gettemp()”, is the next one to be discussed. The structure of the “gettemp()” is exactly the same as the get distance function. Again, the float data type is used since the temperature value uses a decimal point and the function does not need to pass any parameters to its code, so no variables are declared in the declaration statement on line 538, shown in Figure 59. Beyond that code, was pulled straight from “LLI Firmware Rev 2.0”, so there are no changes to discuss.



```
538 float gettemp()
539 { //Used to pull temperature readings from the Temp sensor
540   int val = analogRead(tempPin); //Get temperature from TI sensor
541   float mv = (val/1023.0)*5000; //Convert raw ADC data to voltage value
542   float cel = mv/10; //Convert voltage to degree celcius
543   Serial.print(F(" Temperature (C): "));
544   Serial.println(cel);
545   return cel;
546 }
```

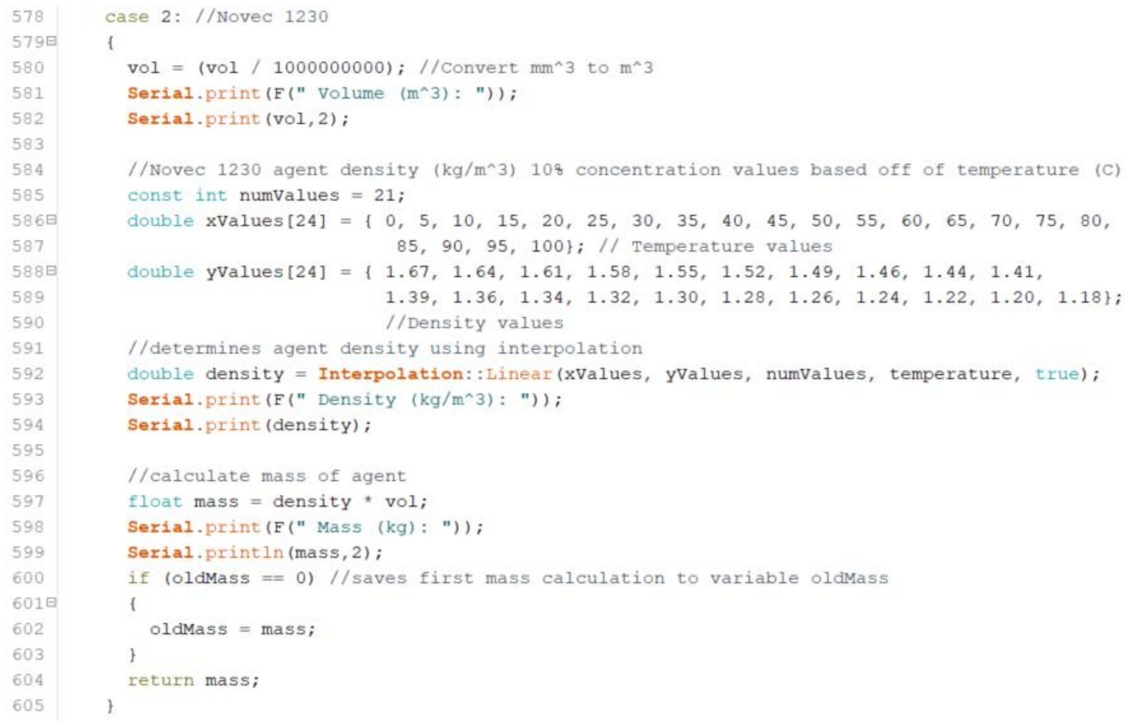
**Figure 59: LLI Firmware with User-Interface gettemp() Function.**

Like the previous functions, the name of the next function, “getmass()”, explains the main purpose of the function, which is calculating the mass of the agent. Just like the declaration for the “getvolume()” function, the declaration of the “getmass()” function passes on a parameter to its code. In this case, the parameter is the volume that was calculated by the “getvolume()” function. There are additional parameters that the function needs such, as temperature, but because they are stored in global variables, they do not need to be redeclared in the function declaration. Another notable observation about the “getmass()” function is that like the “devicemenue()”, it uses a switch case control statement. In this case, the switch is controlled by the global variable “Agent”, which stores which agent was selected by the user. This is important, because as noted before, the mass of the agent is dependent on its density, so to calculate the agent mass correctly, the right agent density must be used. Therefore, if “Agent” is equal to one, “case 1” of the switch is run, which holds the density data for FM-200, and if it equals two, “case 2” is processed, which has the density data for Novec 1230, and if water is selected the variable “Agent” would be equal to three, which would run “case 3” of the switch statement. Another addition to the code is the “if” control statement at the bottom of each case statement. This “if” statement was added in order to save the very first mass calculation as a reference point for the program, which will be discussed further in the section pertaining to the “checkLimits()” function. Other than that, the code used in the three cases is the same code used in “LLI Firmware Rev 2.0” to calculate mass – it is merely repeated for three different agents. With that said, there is one small difference in “case 3,” because the units used for the density of water are different than the units used for FM-200 and Novec 1230. In the case of water, the units are grams per cubic centimeter instead of cubic meters, so the unit conversion calculations are a little different. Because of the size of the function, it was split up into three Figures 60, 61 and 62, each covering a case of the switch statement.



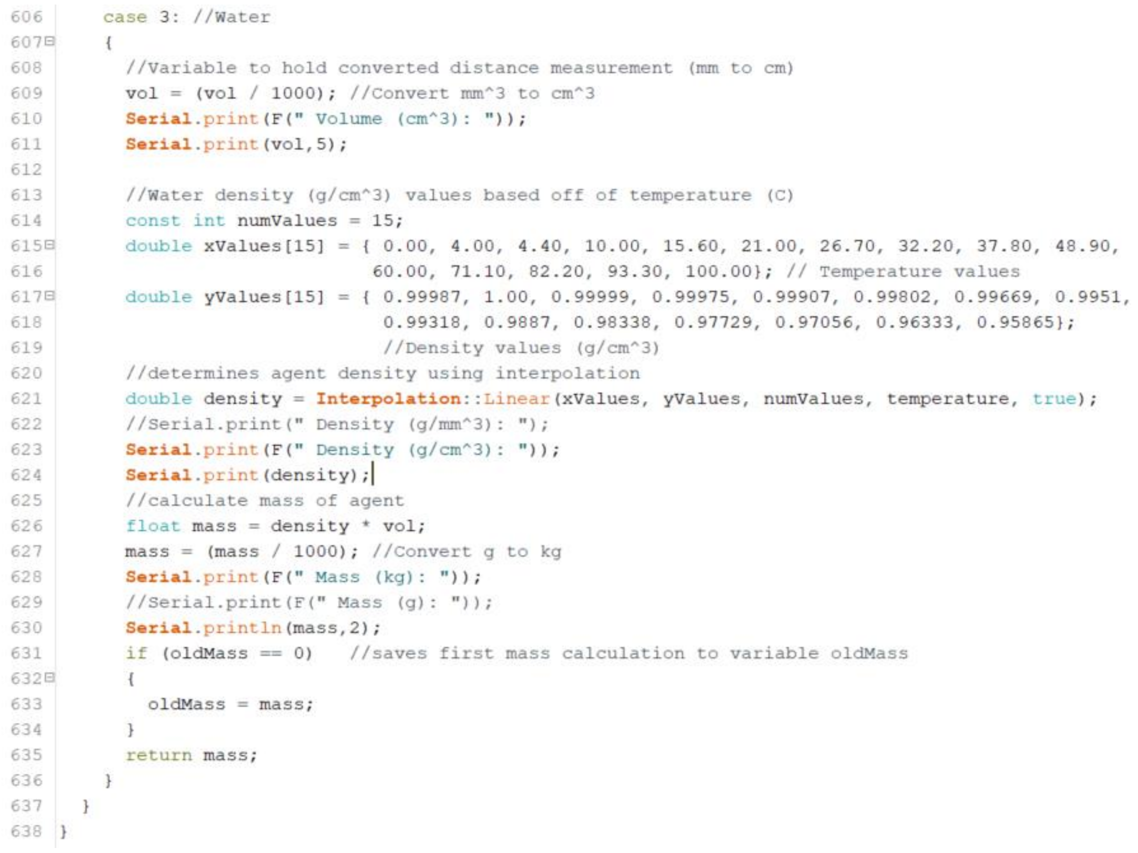
```
float getmass(float vol)
{ //Used to determin the density and mass of the agent
  switch (Agent)
  {
    case 1: //FM-200
    {
      vol = (vol / 1000000000); //Convert mm^3 to m^3
      Serial.print(F(" Volume (m^3): "));
      Serial.print(vol,5);

      //FM-200 agent temperature (C) and density (kg/m^3) values based off of temperature (C)
      const int numValues = 21;
      double xValues[24] = { 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75,
                            80, 85, 90, 95, 100}; // Temperature values
      double yValues[24] = { 1486.0, 1467.3, 1448.2, 1428.6, 1408.4, 1387.7, 1366.2, 1344.0,
                             1320.9, 1296.7, 1271.4, 1244.8, 1216.5, 1186.2, 1153.6, 1117.9,
                             1078.2,1032.8, 978.6, 907.8, 786.8}; //Density values
      //determines agent density using interpolation
      double density = Interpolation::Linear(xValues, yValues, numValues, temperature, true);
      Serial.print(F(" Density (kg/m^3): "));
      Serial.print(density);
      //calculate mass of agent
      float mass = density * vol;
      Serial.print(F(" Mass (kg): "));
      Serial.println(mass,2);
      if (oldMass == 0)   //saves first mass calculation to variable oldMass
      {
        oldMass = mass;
      }
      return mass;
    }
```

Figure 60: LLI Firmware with User-Interface getmass() Function Switch Case 1.



```
    case 2: //Novec 1230
    {
      vol = (vol / 1000000000); //Convert mm^3 to m^3
      Serial.print(F(" Volume (m^3): "));
      Serial.print(vol,2);

      //Novec 1230 agent density (kg/m^3) 10% concentration values based off of temperature (C)
      const int numValues = 21;
      double xValues[24] = { 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80,
                             85, 90, 95, 100}; // Temperature values
      double yValues[24] = { 1.67, 1.64, 1.61, 1.58, 1.55, 1.52, 1.49, 1.46, 1.44, 1.41,
                           1.39, 1.36, 1.34, 1.32, 1.30, 1.28, 1.26, 1.24, 1.22, 1.20, 1.18};
                           //Density values
      //determines agent density using interpolation
      double density = Interpolation::Linear(xValues, yValues, numValues, temperature, true);
      Serial.print(F(" Density (kg/m^3): "));
      Serial.print(density);

      //calculate mass of agent
      float mass = density * vol;
      Serial.print(F(" Mass (kg): "));
      Serial.println(mass,2);
      if (oldMass == 0) //saves first mass calculation to variable oldMass
      {
        oldMass = mass;
      }
      return mass;
    }
```

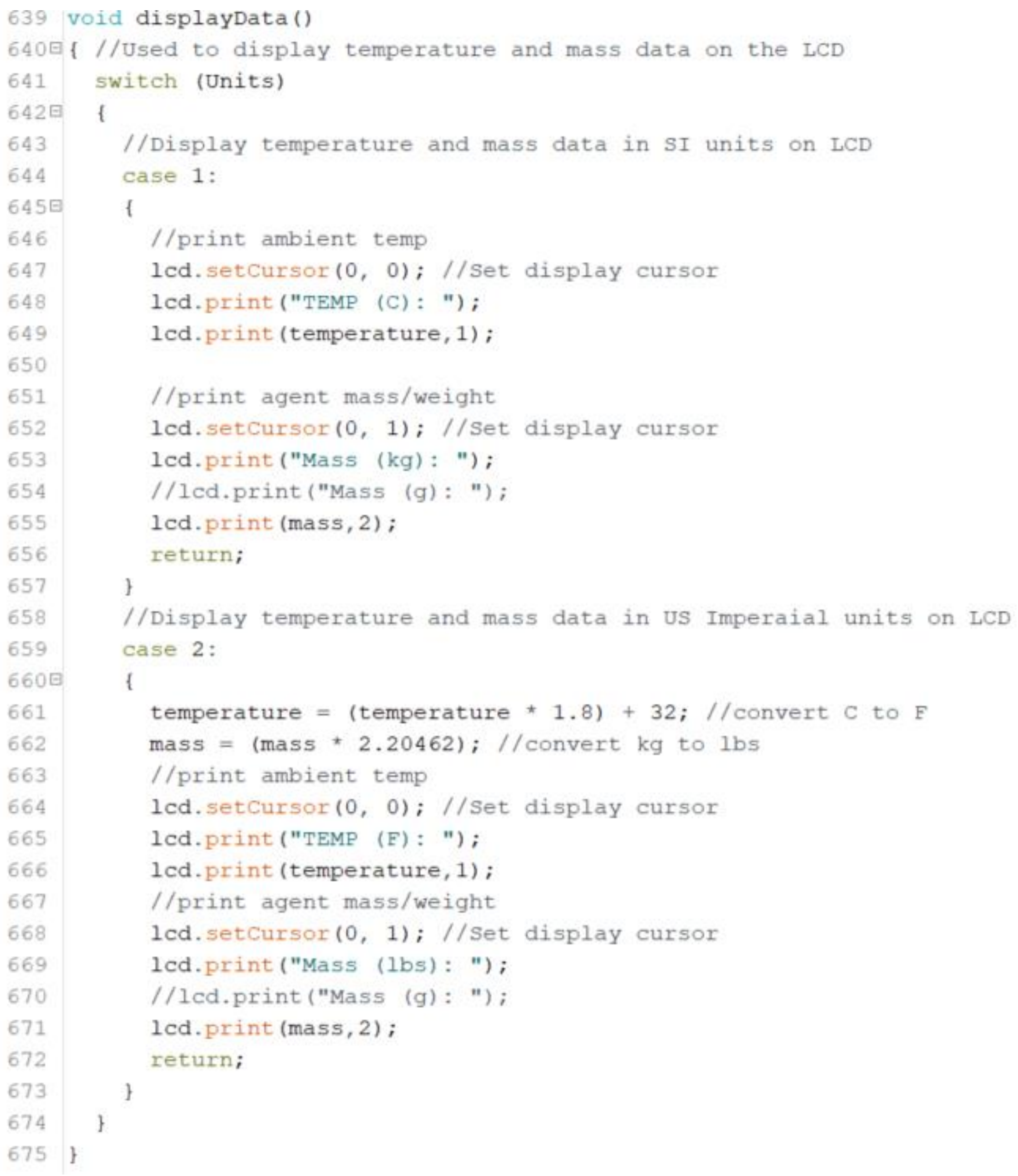
Figure 61: LLI Firmware with User-Interface getmass() Function Switch Case 2.



```
    case 3: //Water
    {
      //Variable to hold converted distance measurement (mm to cm)
      vol = (vol / 1000); //Convert mm^3 to cm^3
      Serial.print(F(" Volume (cm^3): "));
      Serial.print(vol,5);

      //Water density (g/cm^3) values based off of temperature (C)
      const int numValues = 15;
      double xValues[15] = { 0.00, 4.00, 4.40, 10.00, 15.60, 21.00, 26.70, 32.20, 37.80, 48.90,
                             60.00, 71.10, 82.20, 93.30, 100.00}; // Temperature values
      double yValues[15] = { 0.99987, 1.00, 0.99999, 0.99975, 0.99907, 0.99802, 0.99669, 0.9951,
                              0.99318, 0.9887, 0.98338, 0.97729, 0.97056, 0.96333, 0.95865};
                              //Density values (g/cm^3)
      //determines agent density using interpolation
      double density = Interpolation::Linear(xValues, yValues, numValues, temperature, true);
      //Serial.print(" Density (g/mm^3): ");
      Serial.print(F(" Density (g/cm^3): "));
      Serial.print(density);
      //calculate mass of agent
      float mass = density * vol;
      mass = (mass / 1000); //Convert g to kg
      Serial.print(F(" Mass (kg): "));
      //Serial.print(F(" Mass (g): "));
      Serial.println(mass,2);
      if (oldMass == 0)   //saves first mass calculation to variable oldMass
      {
        oldMass = mass;
      }
      return mass;
    }
  }
}
```

**Figure 62: LLI Firmware with User-Interface getmass() Function Switch Case 3.**

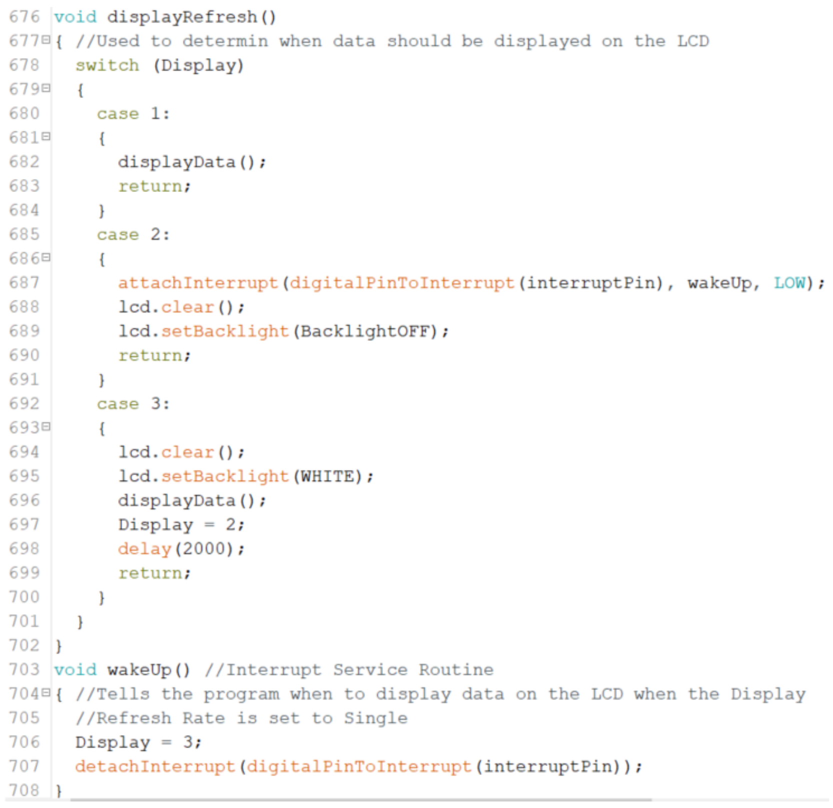
The purpose of the next function, “displayData()”, is to tell the program to display temperature and mass data on the LLI’s display and to do it using the correct units of measurement based on what the user selected, Metric or US Imperial. To do this, the function must know which units of measure were selected by the user. Like the “getmass()” and “devicemenu()” functions, the “displayData()” function (shown in Figure 63) uses a “switch case” control statement which is controlled by the global variable “Units”. The global variable “Units” is used to stores what units the user wanted the measurement date to be in. Because the user had two options, there are two case statements, with “case 1” being processed for metric units and “case 2” for US Imperial units. The code for the two cases should look familiar as it was the same code that appears "LLI Firmware Rev 2.0" to display temperature and mass data, with the only difference being in "case 2" on lines 661 and 662 where temperature is converted from Celsius to Fahrenheit.



```
639 void displayData()
640 { //Used to display temperature and mass data on the LCD
641   switch (Units)
642   {
643     //Display temperature and mass data in SI units on LCD
644     case 1:
645     {
646       //print ambient temp
647       lcd.setCursor(0, 0); //Set display cursor
648       lcd.print("TEMP (C): ");
649       lcd.print(temperature,1);
650
651       //print agent mass/weight
652       lcd.setCursor(0, 1); //Set display cursor
653       lcd.print("Mass (kg): ");
654       //lcd.print("Mass (g): ");
655       lcd.print(mass,2);
656       return;
657     }
658     //Display temperature and mass data in US Imperaial units on LCD
659     case 2:
660     {
661       temperature = (temperature * 1.8) + 32; //convert C to F
662       mass = (mass * 2.20462); //convert kg to lbs
663       //print ambient temp
664       lcd.setCursor(0, 0); //Set display cursor
665       lcd.print("TEMP (F): ");
666       lcd.print(temperature,1);
667       //print agent mass/weight
668       lcd.setCursor(0, 1); //Set display cursor
669       lcd.print("Mass (lbs): ");
670       //lcd.print("Mass (g): ");
671       lcd.print(mass,2);
672       return;
673     }
674   }
675 }
```

**Figure 63: LLI Firmware with User-Interface displayData() Function.**

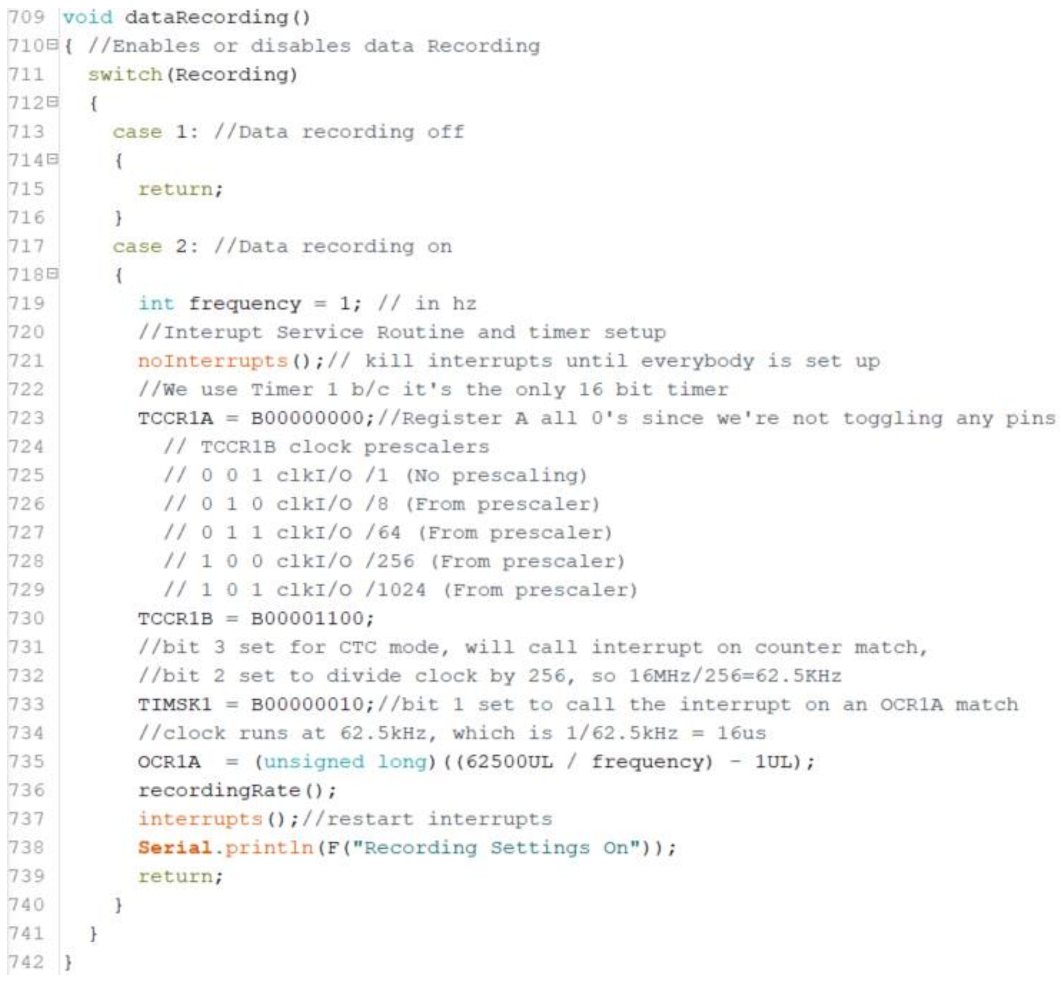
Now that the "displayData()" function has been discussed, the "displayRefresh() function and the "wakeup()" ISR can be discussed, since without these two functions, the program would not know when to display data on the LLI's display. These two functions are shown in Figure 64. Like the previous functions, the refresh rate of the LLI display is dependent on a user setting -- in this case, what did the user set the Display Data setting to -- “Continuous” or “Single” -- which is stored in the global variable “Display”. As previously, a “switch case” control statement is used and is controlled by “Display”. However, unlike the previous function, even though the user is only given two options this “switch case” used in the “displayRefresh()” employs three case statements. With that said, “case 3” is tied to “case 2”, so in essence, there are only two cases. Before that is explained further, it is helpful to discuss “case 1”, which is processed if the user selected “continuous” (“Display” setting is equal to 1), In this case, the LLI display is always being refreshed with new temperature and agent mass data by continuously calling the “displayData()” function. In the case where the user selects “Single”, meaning “Display” is equal to two, the function first processes “case 2”. This enables the hardware interrupt attached to pin 2 of the microcontroller, which is connected to the LLI’s push button, via the code on line 687. It then clears the LLI display and turns the display’s backlight off. Once the user actuates the LLI’s pushbutton, the ISR “wakeup()” is processed, which sets “Display” equal to three (line 706), disables the interrupt, and then returns to normal operation. Now that “Display” is equal to three, the very next time the program calls the “displayRefressh()” function, “case 3” of the switch will be processed, in turn clearing the display, setting the backlight back to white, and calling the “displayData()” function to display the latest temperature and mass data, setting “Display” equal to two and delaying the program for two seconds so that the data are displayed for five seconds before the display is turned off again.



```
void displayRefresh()
{ //Used to determin when data should be displayed on the LCD
  switch (Display)
  {
    case 1:
    {
      displayData();
      return;
    }
    case 2:
    {
      attachInterrupt(digitalPinToInterrupt(interruptPin), wakeUp, LOW);
      lcd.clear();
      lcd.setBacklight(BacklightOFF);
      return;
    }
    case 3:
    {
      lcd.clear();
      lcd.setBacklight(WHITE);
      displayData();
      Display = 2;
      delay(2000);
      return;
    }
  }
}
void wakeUp() //Interrupt Service Routine
{ //Tells the program when to display data on the LCD when the Display
  //Refresh Rate is set to Single
  Display = 3;
  detachInterrupt(digitalPinToInterrupt(interruptPin));
}
```

**Figure 64: LLI Firmware with User-Interface displayRefresh() Function and wakeup() ISR.**

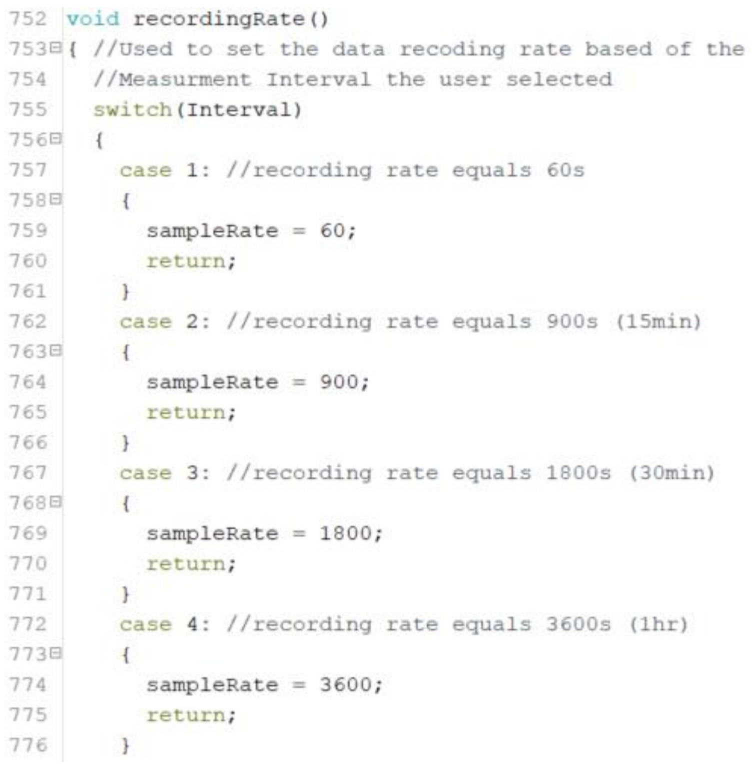
In this section, the following functions will be discussed: “dataRecording()”, “ISR(TIMER1_COMPA_vect)”, “recordingRate()”, and “recordData(). These functions have been grouped together because they all pertain to the LLI’s data recording functionality. Like other LLI functions, its data recording function is tied to two user determined settings, and those settings are “Data Recording” (enables or disables data recording) and “Measurement Interval” (determines the time between entry points in the history log). The global variables that store these settings are “Recording” and “Interval”, respectively. Since the switch case statement in function “dataRecording()” is controlled by the variable “Recording”, it is useful to discuss this function first. The “dataRecording()” function is shown in Figure 65. It is this function that disables or enables the data recording function of the LLI. The first case that will be discussed is if the user selects to disable data recording, meaning the variable “Recording” is equal to one, so “case 1” of the function’s switch statement is processed, making it a pass through. This means the overall function of the program is unaltered, so functions “ISR(TIMER1_COMPA_vect)” and “recordingRate()” are never processed and the “recordData()” function acts as a passthrough as well.



```
709 void dataRecording()
710 { //Enables or disables data Recording
711   switch(Recording)
712   {
713     case 1: //Data recording off
714     {
715       return;
716     }
717     case 2: //Data recording on
718     {
719       int frequency = 1; // in hz
720       //Interupt Service Routine and timer setup
721       noInterrupts();// kill interrupts until everybody is set up
722       //We use Timer 1 b/c it's the only 16 bit timer
723       TCCR1A = B00000000;//Register A all 0's since we're not toggling any pins
724         // TCCR1B clock prescalers
725         // 0 0 1 clkI/O /1 (No prescaling)
726         // 0 1 0 clkI/O /8 (From prescaler)
727         // 0 1 1 clkI/O /64 (From prescaler)
728         // 1 0 0 clkI/O /256 (From prescaler)
729         // 1 0 1 clkI/O /1024 (From prescaler)
730       TCCR1B = B00001100;
731       //bit 3 set for CTC mode, will call interrupt on counter match,
732       //bit 2 set to divide clock by 256, so 16MHz/256=62.5KHz
733       TIMSK1 = B00000010;//bit 1 set to call the interrupt on an OCR1A match
734       //clock runs at 62.5kHz, which is 1/62.5kHz = 16us
735       OCR1A  = (unsigned long)((62500UL / frequency) - 1UL);
736       recordingRate();
737       interrupts();//restart interrupts
738       Serial.println(F("Recording Settings On"));
739       return;
740     }
741   }
742 }
```

**Figure 65: LLI Firmware with User-Interface dataRecording() Function.**

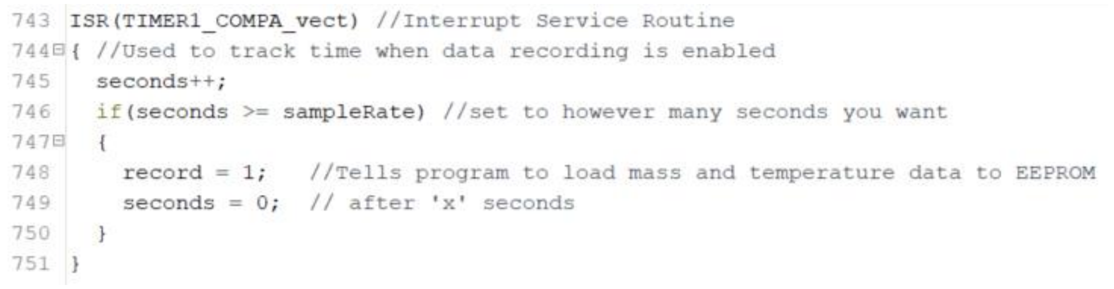
However, this all changes if the user enables data recording, so “Recording” is equal to two. In this case, the code in “case 2” of the “dataRecording()” function’s switch case statement is processed. This code first disables all interrupts (line 721), then lines 723 to 723 enable the microcontroller’s Timer/Counter 1 to trigger a software interrupt every second, calls the “recordingRate()” (line 736) function and then reenables interrupts (line 737). The “recordingRate()” function determines the rate of data recording based on the value of the global variable “Interval”, which stored the setting selected by the user. A portion of this function is shown in Figure 66; the full function is shown in Appendix K. The user can select from eight different recording rates, as mentioned during the discussion of the user-interface. If the user selected a rate of every 60 seconds, “Interval” would be equal to one so “case 1” of the function’s switch case will be processed. In this case, global variable “sampleRate” will be set to 60. If the user selected a rate of one hour, the code in “case 4” would be processed, setting “sampleRate” to 3600.



```
void recordingRate()
{ //Used to set the data recoding rate based of the
  //Measurment Interval the user selected
  switch(Interval)
  {
    case 1: //recording rate equals 60s
    {
      sampleRate = 60;
      return;
    }
    case 2: //recording rate equals 900s (15min)
    {
      sampleRate = 900;
      return;
    }
    case 3: //recording rate equals 1800s (30min)
    {
      sampleRate = 1800;
      return;
    }
    case 4: //recording rate equals 3600s (1hr)
    {
      sampleRate = 3600;
      return;
    }
```

**Figure 66: LLI Firmware with User-Interface recordingRate()” Function Snippet.**

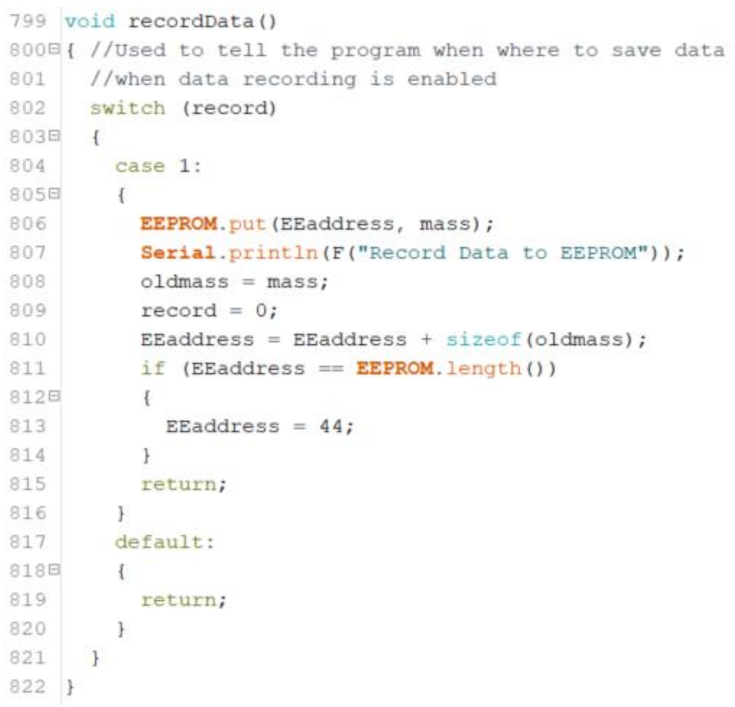
Once “sampleRate” is set, the program returns to the “dataRecoding()” function, reenables the interrupts (shown on line 737 of Figure 65) and then continues with the predefined code in the “loop” function. However, now that interrupts have been enabled, every second a software interrupt occurs, causing the microcontroller to process the code in the ISR “ISR(TIMER1_COMPA_vect)”, which is shown in Figure 67. All the code does is increment the global variable “seconds” and checks to see if that variable is equal to or greater than the global variable “sampleRate” (the variable set in the “recordingRate() function). If “seconds” is equal to or greater than “sampleRate”, the global variable “record” is set to equal one and “second” is set back to zero.



```
ISR(TIMER1_COMPA_vect) //Interrupt Service Routine
{ //Used to track time when data recording is enabled
  seconds++;
  if(seconds >= sampleRate) //set to however many seconds you want
  {
    record = 1;   //Tells program to load mass and temperature data to EEPROM
    seconds = 0;  // after 'x' seconds
  }
}
```

**Figure 67: LLI Firmware with User-Interface ISR(TIMER1_COMPA_vect) ISR.**

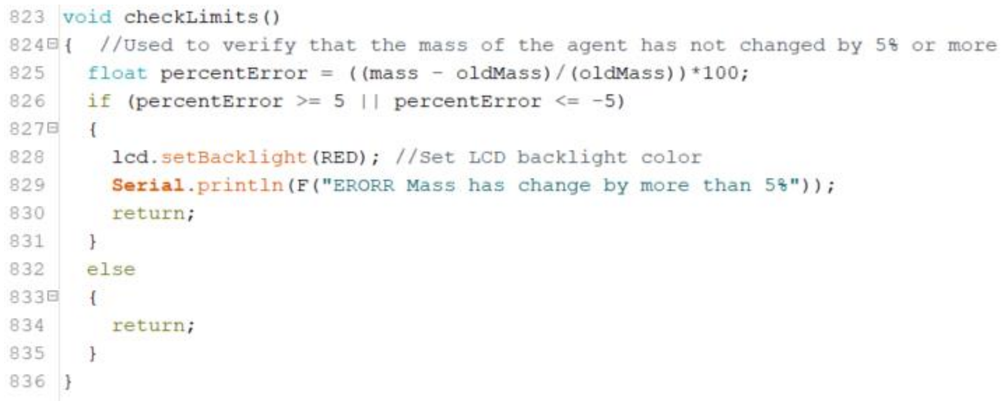
Once finished processing the ISR code, the microcontroller returns to processing the code in the “loop” function, where it now calls the function “recordData()”, which through another switch case control statement, tracks “record”. Now that “record” has been set to equal one, “case 1” is processed running the code on lines 806 to 815, shown on Figure 68. This code writes the mass measurement data to the history log stored on the EEPROM chip, starting at memory address 44. Once the data have been written to the EEPROM, the memory address is incremented (line 810) to be ready for the next write command. The “if” statement on lines 811 to 814 resets the memory address back to 44 once all memory addresses have been written to (a total of 245 data points). At this point, old data starting back at address 44 will be overwritten by new data points. Therefore, a timestamp is necessary. Once the program starts to overwrite the old data, it will be impossible to tell what the most recent data point is if there is no timestamp.



```
void recordData()
{ //Used to tell the program when where to save data
  //when data recording is enabled
  switch (record)
  {
    case 1:
    {
      EEPROM.put(EEaddress, mass);
      Serial.println(F("Record Data to EEPROM"));
      oldmass = mass;
      record = 0;
      EEaddress = EEaddress + sizeof(oldmass);
      if (EEaddress == EEPROM.length())
      {
        EEaddress = 44;
      }
      return;
    }
    default:
    {
      return;
    }
  }
}
```

**Figure 68: LLI Firmware with User-Interface recordData() Function.**

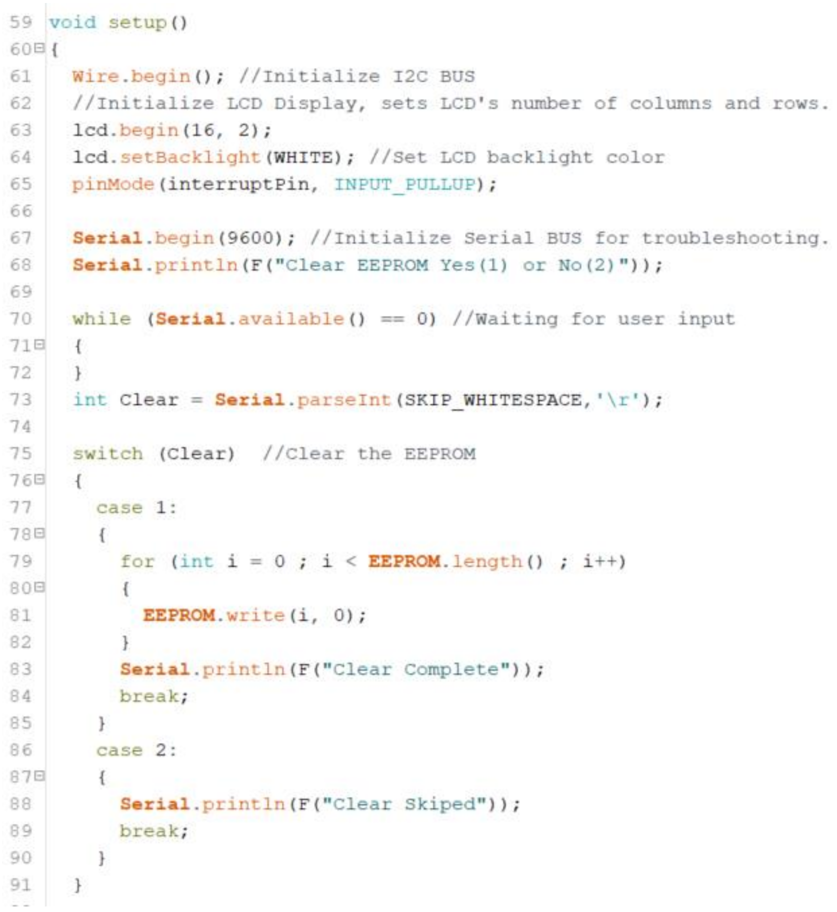
The purpose of the next function, “checkLimits()”, shown in Figure 69, is to verify that the mass of the agent has not changed by five percent or more. This is done by comparing the current mass value to the original mass value that was stored in the global variable “oldMass” by calculating the percent error between the two values. If that percent error is equal to or greater than five percent, the microcontroller will turn the display backlight to red and transmit an error message to the host device.



```
void checkLimits()
{  //Used to verify that the mass of the agent has not changed by 5% or more
  float percentError = ((mass - oldMass)/(oldMass))*100;
  if (percentError >= 5 || percentError <= -5)
  {
    lcd.setBacklight(RED); //Set LCD backlight color
    Serial.println(F("ERORR Mass has change by more than 5%"));
    return;
  }
  else
  {
    return;
  }
}
```

**Figure 69: LLI Firmware with User-Interface checkLimits() Function.**

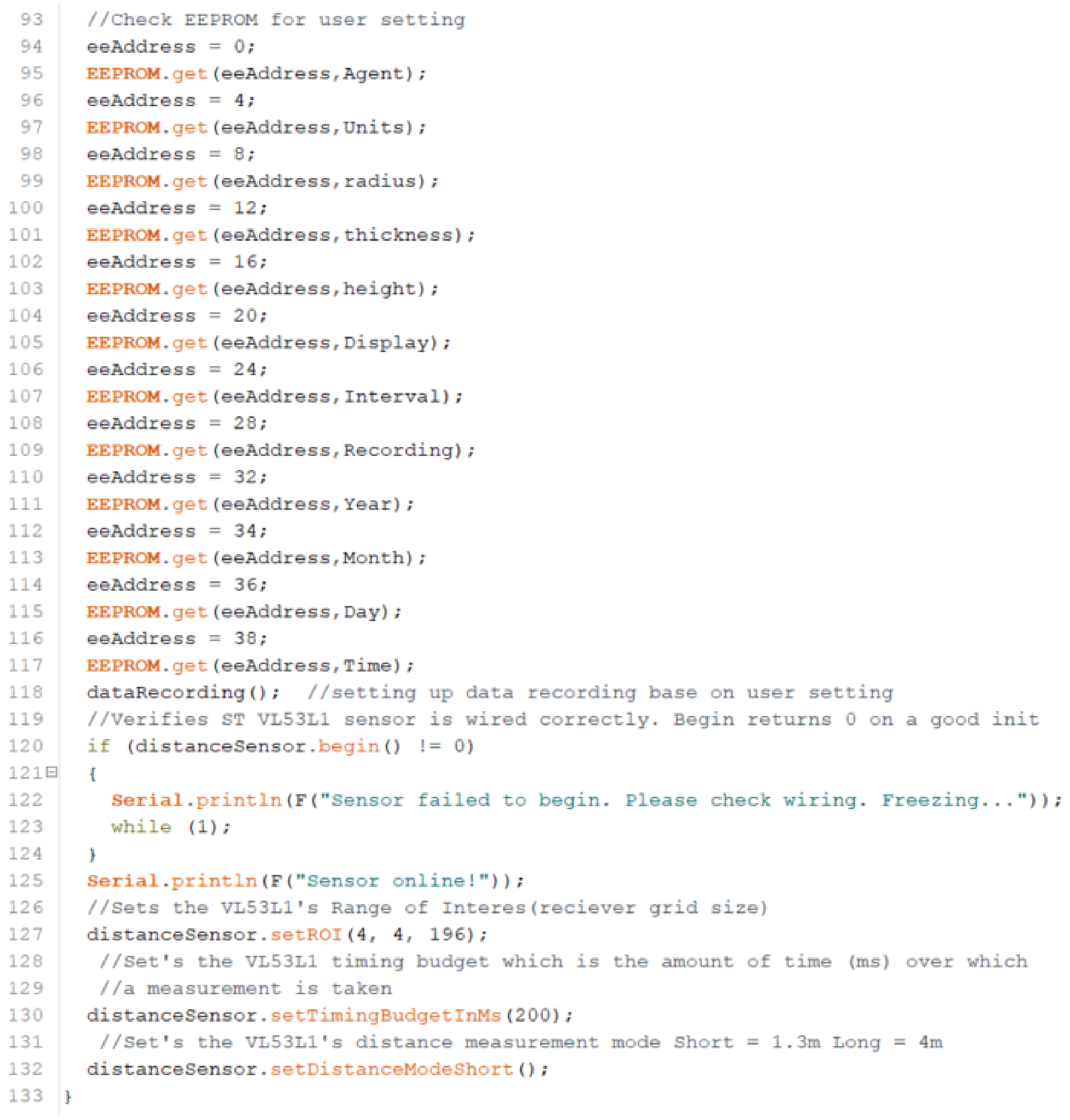
The last two functions to be discussed are the “setup()” and “loop” functions. Because of the addition of the user-interface/device menu, the “setup()” function has changed quite a bit compared to the LLI Firmware Rev 2.0. As shown in Figure 70, lines 68 to 91 were added to allow the user to clear the LLI’s device settings by erasing the EEPROM.



```
59 void setup()
60 {
61   Wire.begin(); //Initialize I2C BUS
62   //Initialize LCD Display, sets LCD's number of columns and rows.
63   lcd.begin(16, 2);
64   lcd.setBacklight(WHITE); //Set LCD backlight color
65   pinMode(interruptPin, INPUT_PULLUP);
66 
67   Serial.begin(9600); //Initialize Serial BUS for troubleshooting.
68   Serial.println(F("Clear EEPROM Yes(1) or No(2)"));
69 
70   while (Serial.available() == 0) //Waiting for user input
71   {
72   }
73   int Clear = Serial.parseInt(SKIP_WHITESPACE,'\r');
74 
75   switch (Clear)  //Clear the EEPROM
76   {
77     case 1:
78     {
79       for (int i = 0 ; i < EEPROM.length() ; i++)
80       {
81         EEPROM.write(i, 0);
82       }
83       Serial.println(F("Clear Complete"));
84       break;
85     }
86     case 2:
87     {
88       Serial.println(F("Clear Skiped"));
89       break;
90     }
91   }
```

**Figure 70: LLI Firmware with User-Interface setup() Function.**

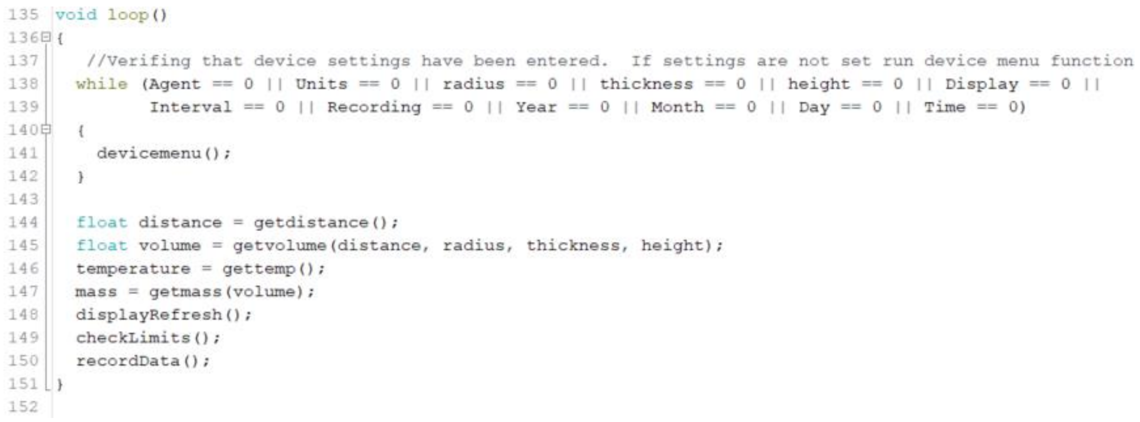
Lines 93 to 117, shown in Figure 71, were added to verify the LLI’s device settings upon startup. Additionally, in line 118, the “dataRecording()” function call was added so that if data recording was enabled and power was lost, once power was returned to the device, data recording would resume. Beyond those changes, the remaining lines of code are the same as in previous firmware versions.



```
  //Check EEPROM for user setting
  eeAddress = 0;
  EEPROM.get(eeAddress,Agent);
  eeAddress = 4;
  EEPROM.get(eeAddress,Units);
  eeAddress = 8;
  EEPROM.get(eeAddress,radius);
  eeAddress = 12;
  EEPROM.get(eeAddress,thickness);
  eeAddress = 16;
  EEPROM.get(eeAddress,height);
  eeAddress = 20;
  EEPROM.get(eeAddress,Display);
  eeAddress = 24;
  EEPROM.get(eeAddress,Interval);
  eeAddress = 28;
  EEPROM.get(eeAddress,Recording);
  eeAddress = 32;
  EEPROM.get(eeAddress,Year);
  eeAddress = 34;
  EEPROM.get(eeAddress,Month);
  eeAddress = 36;
  EEPROM.get(eeAddress,Day);
  eeAddress = 38;
  EEPROM.get(eeAddress,Time);
  dataRecording();  //setting up data recording base on user setting
  //Verifies ST VL53L1 sensor is wired correctly. Begin returns 0 on a good init
  if (distanceSensor.begin() != 0)
  {
    Serial.println(F("Sensor failed to begin. Please check wiring. Freezing..."));
    while (1);
  }
  Serial.println(F("Sensor online!"));
  //Sets the VL53L1's Range of Interes(reciever grid size)
  distanceSensor.setROI(4, 4, 196);
   //Set's the VL53L1 timing budget which is the amount of time (ms) over which
   //a measurement is taken
  distanceSensor.setTimingBudgetInMs(200);
   //Set's the VL53L1's distance measurement mode Short = 1.3m Long = 4m
  distanceSensor.setDistanceModeShort();
}
```

**Figure 71: LLI Firmware with User-Interface setup() Function Continued.**

Regarding the “loop” function, like the “setup()” function, this function has also seen some significant changes due to the implementation of all the other functions. However, the addition of the other functions have significantly simplified the “loop()” and reduced its overall size from 75 lines of code to a mere 16 lines, as shown in Figure 72. As indicated in the “Serial Comms Rev 5” firmware discussion, the “while” loop on lines 137 to 142 was added to verify that all device settings were properly set before normal operations begin. If they were not properly set, the "devicemenu()" is called so that the user can set the device setting properly. Once the settings are set, the microcontroller will then start processing the code on lines 144 to 150, calling each function and executing those functions based on the user settings.



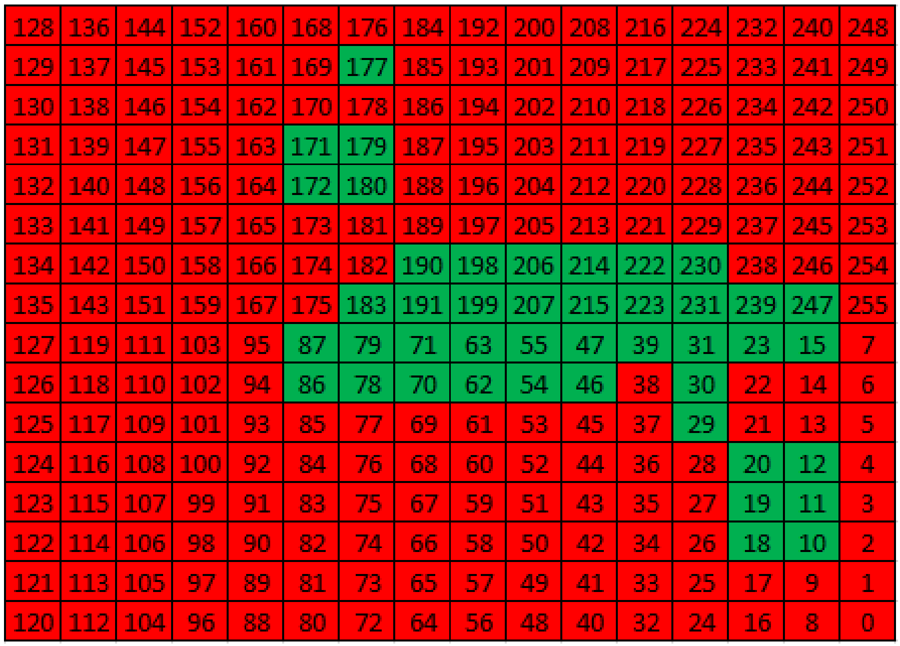
```
135  void loop()
136  {
137     //Verifing that device settings have been entered.  If settings are not set run device menu function
138    while (Agent == 0 || Units == 0 || radius == 0 || thickness == 0 || height == 0 || Display == 0 ||
139           Interval == 0 || Recording == 0 || Year == 0 || Month == 0 || Day == 0 || Time == 0)
140    {
141      devicemenu();
142    }
143
144    float distance = getdistance();
145    float volume = getvolume(distance, radius, thickness, height);
146    temperature = gettemp();
147    mass = getmass(volume);
148    displayRefresh();
149    checkLimits();
150    recordData();
151  }
152
```

**Figure 72: LLI Firmware with User-Interface loop() Function.**

## Results

### VL53L1X TOF Sensor's Region of Interest Calibration

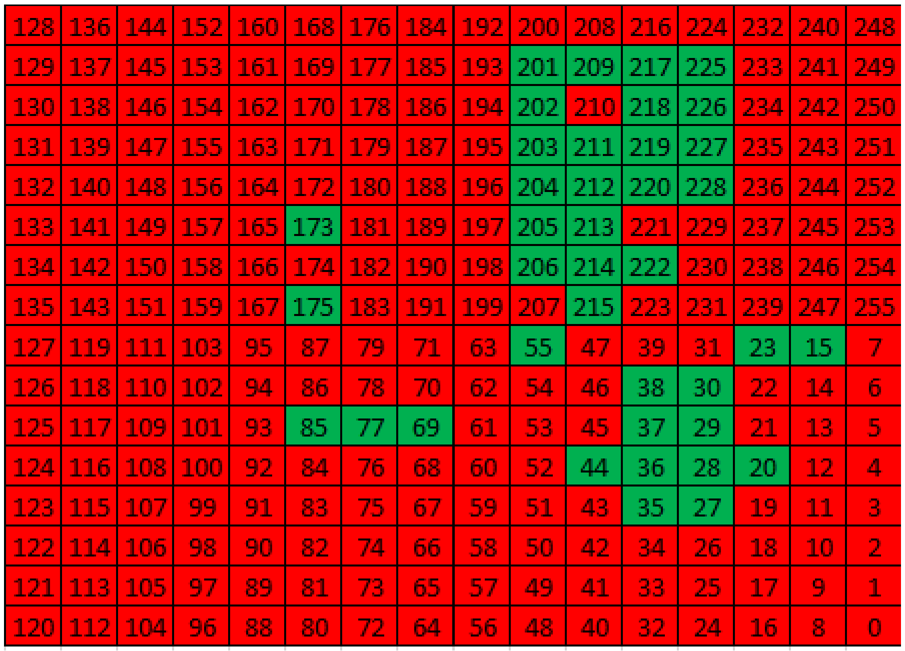
The ROI calibration process showed that the optimal center point was 230 with a timing budget of 200ms. This was determined by making the following plots using the raw output data that were generated from the ROI Calibration program. These data were the ROI center points that gave an average measurement that was within ±1mm of the set distance. The first distance tested was at 102 mm, and the ROI center points shown in green in Figure 73 are the points that returned a measurement of 102mm ±1mm. As shown in the plot, the optimal center-points tended to be clustered in the center of the SPAD array.



| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 128 | 136 | 144 | 152 | 160 | 168 | 176 | 184 | 192 | 200 | 208 | 216 | 224 | 232 | 240 | 248 |
| 129 | 137 | 145 | 153 | 161 | 169 | 177 | 185 | 193 | 201 | 209 | 217 | 225 | 233 | 241 | 249 |
| 130 | 138 | 146 | 154 | 162 | 170 | 178 | 186 | 194 | 202 | 210 | 218 | 226 | 234 | 242 | 250 |
| 131 | 139 | 147 | 155 | 163 | 171 | 179 | 187 | 195 | 203 | 211 | 219 | 227 | 235 | 243 | 251 |
| 132 | 140 | 148 | 156 | 164 | 172 | 180 | 188 | 196 | 204 | 212 | 220 | 228 | 236 | 244 | 252 |
| 133 | 141 | 149 | 157 | 165 | 173 | 181 | 189 | 197 | 205 | 213 | 221 | 229 | 237 | 245 | 253 |
| 134 | 142 | 150 | 158 | 166 | 174 | 182 | 190 | 198 | 206 | 214 | 222 | 230 | 238 | 246 | 254 |
| 135 | 143 | 151 | 159 | 167 | 175 | 183 | 191 | 199 | 207 | 215 | 223 | 231 | 239 | 247 | 255 |
| 127 | 119 | 111 | 103 | 95 | 87 | 79 | 71 | 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |
| 126 | 118 | 110 | 102 | 94 | 86 | 78 | 70 | 62 | 54 | 46 | 38 | 30 | 22 | 14 | 6 |
| 125 | 117 | 109 | 101 | 93 | 85 | 77 | 69 | 61 | 53 | 45 | 37 | 29 | 21 | 13 | 5 |
| 124 | 116 | 108 | 100 | 92 | 84 | 76 | 68 | 60 | 52 | 44 | 36 | 28 | 20 | 12 | 4 |
| 123 | 115 | 107 | 99 | 91 | 83 | 75 | 67 | 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 |
| 122 | 114 | 106 | 98 | 90 | 82 | 74 | 66 | 58 | 50 | 42 | 34 | 26 | 18 | 10 | 2 |
| 121 | 113 | 105 | 97 | 89 | 81 | 73 | 65 | 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 |
| 120 | 112 | 104 | 96 | 88 | 80 | 72 | 64 | 56 | 48 | 40 | 32 | 24 | 16 | 8 | 0 |

**Figure 73: Optimal ROI Center-Points for 102mm ±1mm (Green).**

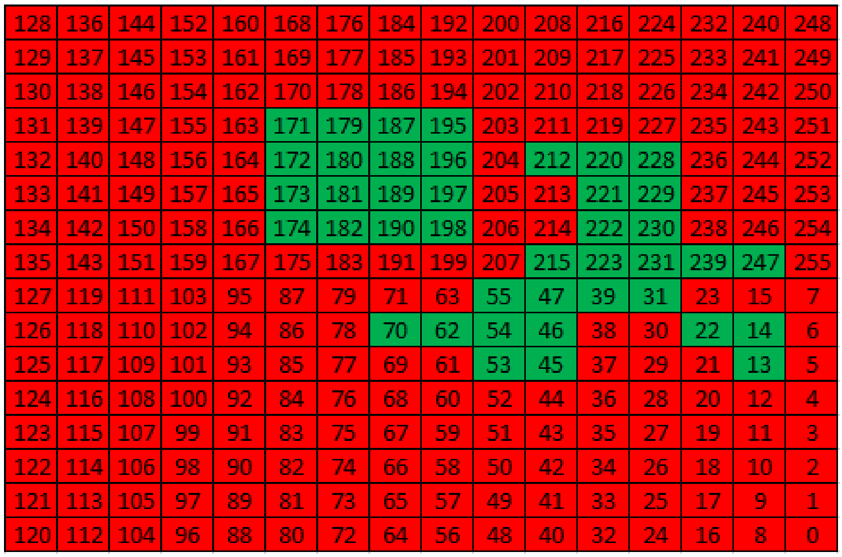
The next position tested was at 152mm, and the optimal center points for this position are shown in Figure 74.



| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 128 | 136 | 144 | 152 | 160 | 168 | 176 | 184 | 192 | 200 | 208 | 216 | 224 | 232 | 240 | 248 |
| 129 | 137 | 145 | 153 | 161 | 169 | 177 | 185 | 193 | 201 | 209 | 217 | 225 | 233 | 241 | 249 |
| 130 | 138 | 146 | 154 | 162 | 170 | 178 | 186 | 194 | 202 | 210 | 218 | 226 | 234 | 242 | 250 |
| 131 | 139 | 147 | 155 | 163 | 171 | 179 | 187 | 195 | 203 | 211 | 219 | 227 | 235 | 243 | 251 |
| 132 | 140 | 148 | 156 | 164 | 172 | 180 | 188 | 196 | 204 | 212 | 220 | 228 | 236 | 244 | 252 |
| 133 | 141 | 149 | 157 | 165 | 173 | 181 | 189 | 197 | 205 | 213 | 221 | 229 | 237 | 245 | 253 |
| 134 | 142 | 150 | 158 | 166 | 174 | 182 | 190 | 198 | 206 | 214 | 222 | 230 | 238 | 246 | 254 |
| 135 | 143 | 151 | 159 | 167 | 175 | 183 | 191 | 199 | 207 | 215 | 223 | 231 | 239 | 247 | 255 |
| 127 | 119 | 111 | 103 | 95 | 87 | 79 | 71 | 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |
| 126 | 118 | 110 | 102 | 94 | 86 | 78 | 70 | 62 | 54 | 46 | 38 | 30 | 22 | 14 | 6 |
| 125 | 117 | 109 | 101 | 93 | 85 | 77 | 69 | 61 | 53 | 45 | 37 | 29 | 21 | 13 | 5 |
| 124 | 116 | 108 | 100 | 92 | 84 | 76 | 68 | 60 | 52 | 44 | 36 | 28 | 20 | 12 | 4 |
| 123 | 115 | 107 | 99 | 91 | 83 | 75 | 67 | 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 |
| 122 | 114 | 106 | 98 | 90 | 82 | 74 | 66 | 58 | 50 | 42 | 34 | 26 | 18 | 10 | 2 |
| 121 | 113 | 105 | 97 | 89 | 81 | 73 | 65 | 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 |
| 120 | 112 | 104 | 96 | 88 | 80 | 72 | 64 | 56 | 48 | 40 | 32 | 24 | 16 | 8 | 0 |

**Figure 74: Optimal ROI Center-Points for 152mm ±1mm (Green).**

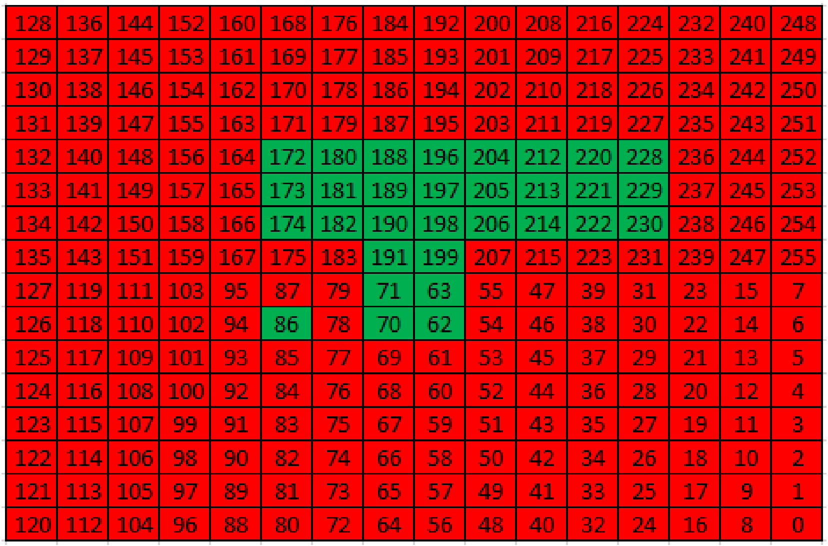
The optimal center points for the third position, 202mm, are shown in Figure 75.



| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 128 | 136 | 144 | 152 | 160 | 168 | 176 | 184 | 192 | 200 | 208 | 216 | 224 | 232 | 240 | 248 |
| 129 | 137 | 145 | 153 | 161 | 169 | 177 | 185 | 193 | 201 | 209 | 217 | 225 | 233 | 241 | 249 |
| 130 | 138 | 146 | 154 | 162 | 170 | 178 | 186 | 194 | 202 | 210 | 218 | 226 | 234 | 242 | 250 |
| 131 | 139 | 147 | 155 | 163 | 🟩 171 | 🟩 179 | 🟩 187 | 🟩 195 | 203 | 211 | 219 | 227 | 235 | 243 | 251 |
| 132 | 140 | 148 | 156 | 164 | 🟩 172 | 🟩 180 | 🟩 188 | 🟩 196 | 204 | 🟩 212 | 🟩 220 | 🟩 228 | 236 | 244 | 252 |
| 133 | 141 | 149 | 157 | 165 | 🟩 173 | 🟩 181 | 🟩 189 | 🟩 197 | 205 | 213 | 🟩 221 | 🟩 229 | 237 | 245 | 253 |
| 134 | 142 | 150 | 158 | 166 | 🟩 174 | 🟩 182 | 🟩 190 | 🟩 198 | 206 | 214 | 🟩 222 | 🟩 230 | 238 | 246 | 254 |
| 135 | 143 | 151 | 159 | 167 | 175 | 183 | 191 | 199 | 207 | 🟩 215 | 🟩 223 | 🟩 231 | 🟩 239 | 🟩 247 | 255 |
| 127 | 119 | 111 | 103 | 95 | 87 | 79 | 71 | 63 | 🟩 55 | 🟩 47 | 🟩 39 | 🟩 31 | 23 | 15 | 7 |
| 126 | 118 | 110 | 102 | 94 | 86 | 78 | 🟩 70 | 🟩 62 | 🟩 54 | 🟩 46 | 38 | 30 | 🟩 22 | 🟩 14 | 6 |
| 125 | 117 | 109 | 101 | 93 | 85 | 77 | 69 | 61 | 🟩 53 | 🟩 45 | 37 | 29 | 21 | 🟩 13 | 5 |
| 124 | 116 | 108 | 100 | 92 | 84 | 76 | 68 | 60 | 52 | 44 | 36 | 28 | 20 | 12 | 4 |
| 123 | 115 | 107 | 99 | 91 | 83 | 75 | 67 | 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 |
| 122 | 114 | 106 | 98 | 90 | 82 | 74 | 66 | 58 | 50 | 42 | 34 | 26 | 18 | 10 | 2 |
| 121 | 113 | 105 | 97 | 89 | 81 | 73 | 65 | 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 |
| 120 | 112 | 104 | 96 | 88 | 80 | 72 | 64 | 56 | 48 | 40 | 32 | 24 | 16 | 8 | 0 |

**Figure 75: Optimal ROI Center-Points for 202mm ±1mm (Green).**

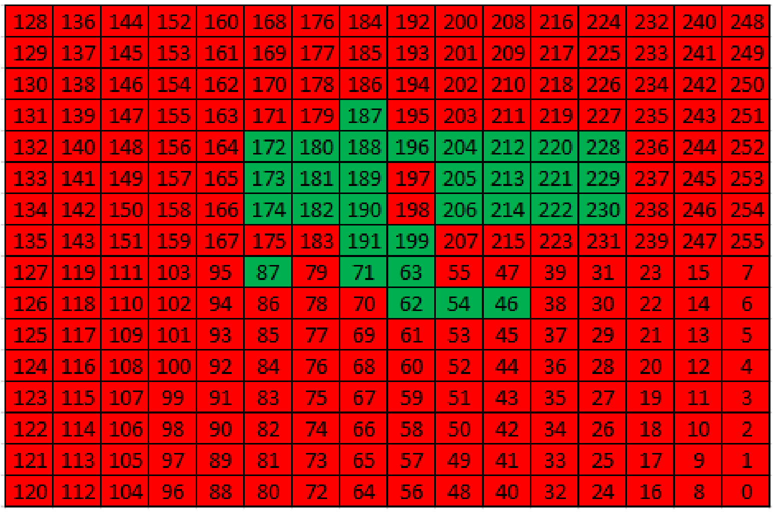
For the fourth position, 252mm was used. The optimal center points for this position are shown in Figure 76.



| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 128 | 136 | 144 | 152 | 160 | 168 | 176 | 184 | 192 | 200 | 208 | 216 | 224 | 232 | 240 | 248 |
| 129 | 137 | 145 | 153 | 161 | 169 | 177 | 185 | 193 | 201 | 209 | 217 | 225 | 233 | 241 | 249 |
| 130 | 138 | 146 | 154 | 162 | 170 | 178 | 186 | 194 | 202 | 210 | 218 | 226 | 234 | 242 | 250 |
| 131 | 139 | 147 | 155 | 163 | 171 | 179 | 187 | 195 | 203 | 211 | 219 | 227 | 235 | 243 | 251 |
| 132 | 140 | 148 | 156 | 164 | 🟩 172 | 🟩 180 | 🟩 188 | 🟩 196 | 🟩 204 | 🟩 212 | 🟩 220 | 🟩 228 | 236 | 244 | 252 |
| 133 | 141 | 149 | 157 | 165 | 🟩 173 | 🟩 181 | 🟩 189 | 🟩 197 | 🟩 205 | 🟩 213 | 🟩 221 | 🟩 229 | 237 | 245 | 253 |
| 134 | 142 | 150 | 158 | 166 | 🟩 174 | 🟩 182 | 🟩 190 | 🟩 198 | 🟩 206 | 🟩 214 | 🟩 222 | 🟩 230 | 238 | 246 | 254 |
| 135 | 143 | 151 | 159 | 167 | 175 | 183 | 🟩 191 | 🟩 199 | 207 | 215 | 223 | 231 | 239 | 247 | 255 |
| 127 | 119 | 111 | 103 | 95 | 87 | 79 | 🟩 71 | 🟩 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |
| 126 | 118 | 110 | 102 | 94 | 🟩 86 | 78 | 🟩 70 | 🟩 62 | 54 | 46 | 38 | 30 | 22 | 14 | 6 |
| 125 | 117 | 109 | 101 | 93 | 85 | 77 | 69 | 61 | 53 | 45 | 37 | 29 | 21 | 13 | 5 |
| 124 | 116 | 108 | 100 | 92 | 84 | 76 | 68 | 60 | 52 | 44 | 36 | 28 | 20 | 12 | 4 |
| 123 | 115 | 107 | 99 | 91 | 83 | 75 | 67 | 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 |
| 122 | 114 | 106 | 98 | 90 | 82 | 74 | 66 | 58 | 50 | 42 | 34 | 26 | 18 | 10 | 2 |
| 121 | 113 | 105 | 97 | 89 | 81 | 73 | 65 | 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 |
| 120 | 112 | 104 | 96 | 88 | 80 | 72 | 64 | 56 | 48 | 40 | 32 | 24 | 16 | 8 | 0 |

**Figure 76: Optimal ROI Center-Points for 252mm ±1mm (Green).**

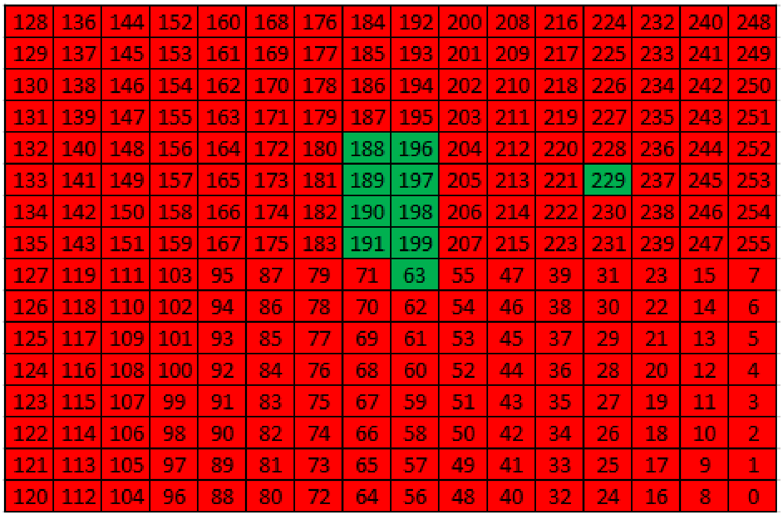
For the fifth position, 301 mm was used, and those optimal center points are shown in Figure 77.



| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 128 | 136 | 144 | 152 | 160 | 168 | 176 | 184 | 192 | 200 | 208 | 216 | 224 | 232 | 240 | 248 |
| 129 | 137 | 145 | 153 | 161 | 169 | 177 | 185 | 193 | 201 | 209 | 217 | 225 | 233 | 241 | 249 |
| 130 | 138 | 146 | 154 | 162 | 170 | 178 | 186 | 194 | 202 | 210 | 218 | 226 | 234 | 242 | 250 |
| 131 | 139 | 147 | 155 | 163 | 171 | 179 | 🟩 187 | 195 | 203 | 211 | 219 | 227 | 235 | 243 | 251 |
| 132 | 140 | 148 | 156 | 164 | 🟩 172 | 🟩 180 | 🟩 188 | 🟩 196 | 🟩 204 | 🟩 212 | 🟩 220 | 🟩 228 | 236 | 244 | 252 |
| 133 | 141 | 149 | 157 | 165 | 🟩 173 | 🟩 181 | 🟩 189 | 197 | 🟩 205 | 🟩 213 | 🟩 221 | 🟩 229 | 237 | 245 | 253 |
| 134 | 142 | 150 | 158 | 166 | 🟩 174 | 🟩 182 | 🟩 190 | 198 | 🟩 206 | 🟩 214 | 🟩 222 | 🟩 230 | 238 | 246 | 254 |
| 135 | 143 | 151 | 159 | 167 | 175 | 183 | 🟩 191 | 🟩 199 | 207 | 215 | 223 | 231 | 239 | 247 | 255 |
| 127 | 119 | 111 | 103 | 95 | 🟩 87 | 79 | 🟩 71 | 🟩 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |
| 126 | 118 | 110 | 102 | 94 | 86 | 78 | 70 | 🟩 62 | 🟩 54 | 🟩 46 | 38 | 30 | 22 | 14 | 6 |
| 125 | 117 | 109 | 101 | 93 | 85 | 77 | 69 | 61 | 53 | 45 | 37 | 29 | 21 | 13 | 5 |
| 124 | 116 | 108 | 100 | 92 | 84 | 76 | 68 | 60 | 52 | 44 | 36 | 28 | 20 | 12 | 4 |
| 123 | 115 | 107 | 99 | 91 | 83 | 75 | 67 | 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 |
| 122 | 114 | 106 | 98 | 90 | 82 | 74 | 66 | 58 | 50 | 42 | 34 | 26 | 18 | 10 | 2 |
| 121 | 113 | 105 | 97 | 89 | 81 | 73 | 65 | 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 |
| 120 | 112 | 104 | 96 | 88 | 80 | 72 | 64 | 56 | 48 | 40 | 32 | 24 | 16 | 8 | 0 |

**Figure 77: Optimal ROI Center-Points for 301mm ±1mm (Green).**

The sixth and final position used to determine the optimal ROI center points was at 352mm. As shown with each subsequent plot, the number of optimal ROI center points decreased the further away from the sensor. This position was no different, as shown in Figure 78.



| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 128 | 136 | 144 | 152 | 160 | 168 | 176 | 184 | 192 | 200 | 208 | 216 | 224 | 232 | 240 | 248 |
| 129 | 137 | 145 | 153 | 161 | 169 | 177 | 185 | 193 | 201 | 209 | 217 | 225 | 233 | 241 | 249 |
| 130 | 138 | 146 | 154 | 162 | 170 | 178 | 186 | 194 | 202 | 210 | 218 | 226 | 234 | 242 | 250 |
| 131 | 139 | 147 | 155 | 163 | 171 | 179 | 187 | 195 | 203 | 211 | 219 | 227 | 235 | 243 | 251 |
| 132 | 140 | 148 | 156 | 164 | 172 | 180 | 🟩 188 | 🟩 196 | 204 | 212 | 220 | 228 | 236 | 244 | 252 |
| 133 | 141 | 149 | 157 | 165 | 173 | 181 | 🟩 189 | 🟩 197 | 205 | 213 | 221 | 🟩 229 | 237 | 245 | 253 |
| 134 | 142 | 150 | 158 | 166 | 174 | 182 | 🟩 190 | 🟩 198 | 206 | 214 | 222 | 230 | 238 | 246 | 254 |
| 135 | 143 | 151 | 159 | 167 | 175 | 183 | 🟩 191 | 🟩 199 | 207 | 215 | 223 | 231 | 239 | 247 | 255 |
| 127 | 119 | 111 | 103 | 95 | 87 | 79 | 71 | 🟩 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |
| 126 | 118 | 110 | 102 | 94 | 86 | 78 | 70 | 62 | 54 | 46 | 38 | 30 | 22 | 14 | 6 |
| 125 | 117 | 109 | 101 | 93 | 85 | 77 | 69 | 61 | 53 | 45 | 37 | 29 | 21 | 13 | 5 |
| 124 | 116 | 108 | 100 | 92 | 84 | 76 | 68 | 60 | 52 | 44 | 36 | 28 | 20 | 12 | 4 |
| 123 | 115 | 107 | 99 | 91 | 83 | 75 | 67 | 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 |
| 122 | 114 | 106 | 98 | 90 | 82 | 74 | 66 | 58 | 50 | 42 | 34 | 26 | 18 | 10 | 2 |
| 121 | 113 | 105 | 97 | 89 | 81 | 73 | 65 | 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 |
| 120 | 112 | 104 | 96 | 88 | 80 | 72 | 64 | 56 | 48 | 40 | 32 | 24 | 16 | 8 | 0 |

**Figure 78: Optimal ROI Center-Points for 352mm ±1mm (Green).**

With the optimal ROI center points found, the next step was to see if there were any common points between the six positions. As shown in the plots, some of the positions share optimal center points, but others do not. For example, center point 228 is common for positions 2, 3, 4, and 5, but not common for positions 1 and 6. However, position 1 had an optimal center point of 230 and position 6 had an optimal center point of 229, which would be included in the 4x4 ROI grid if a center point of 228 was used. This is where a little trial and error was performed. Several center points were tested to see which would work best. The center points were chosen based on their commonality with other positions and how they would impact the overall 4x4 ROI grid. Those center points were 196, 228, 204 and 230. At each of these center points, 30 distance measurements were taken at each of the six positions. These measurement data are shown in Appendix F. From these data, the standard deviation, average standard deviation, average and percent error for each proposed center point were calculated using Microsoft Excel. The values of these calculations are shown in Table 4. Reviewing the data in Table 4, position 1 at 102mm was the deciding factor for choosing a ROI center point of 230. This is because it is the only center point for that position that had a precent error less than 1%.

**Table 4: ROI Center-Point Calibration Calculations.**

| | Distance = 352 ±1 | | | | Distance = 301 ±1 | | | | Distance = 252 ±1 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 230cp | 204cp | 228cp | 196cp | 230cp | 204cp | 228cp | 196cp | 230cp | 204cp | 228cp | 196cp |
| ST.Dev: | 0.179 | 0.179 | 0.101 | 0.155 | 0.157 | 0.159 | 0.136 | 0.144 | 0.217 | 0.116 | 0.164 | 0.150 |
| Average Dev: | 0.151 | 0.137 | 0.075 | 0.117 | 0.135 | 0.128 | 0.097 | 0.105 | 0.192 | 0.090 | 0.137 | 0.121 |
| Average: | 350.392 | 350.839 | 351.453 | 348.944 | 300.307 | 299.849 | 301.211 | 301.126 | 250.775 | 250.366 | 250.924 | 252.785 |
| %error | 0.457 | 0.330 | 0.155 | 0.868 | 0.230 | 0.383 | -0.070 | -0.042 | 0.486 | 0.649 | 0.427 | -0.311 |
| | Distance = 202 ±1 | | | | Distance = 152 ±1 | | | | Distance = 102 ±1 | | | |
| | 230cp | 204cp | 228cp | 196cp | 230cp | 204cp | 228cp | 196cp | 230cp | 204cp | 228cp | 196cp |
| ST.Dev: | 0.455 | 0.202 | 0.188 | 0.428 | 0.370 | 0.189 | 0.185 | 0.154 | 0.072 | 0.183 | 0.194 | 0.255 |
| Average Dev: | 0.361 | 0.159 | 0.141 | 0.363 | 0.315 | 0.151 | 0.162 | 0.120 | 0.062 | 0.151 | 0.151 | 0.204 |
| Average: | 201.314 | 201.756 | 201.883 | 201.909 | 151.436 | 154.543 | 153.063 | 153.467 | 101.960 | 99.311 | 99.810 | 99.837 |
| %error | 0.340 | 0.121 | 0.058 | 0.045 | 0.371 | -1.673 | -0.699 | -0.965 | 0.040 | 2.637 | 2.147 | 2.121 |

**Accuracy and Repeatability Verification of the VL53L1X TOF Sensor**

The following results are from the distance verification testing. The purpose of this testing was to see what effect temperature had on the TOF's measurement data. For the testing, three different positions were checked. These positions were at 151mm, 251mm and 351mm. At each position, distance measurements were taken at three different temperatures including 0C, 23C, and 37.8C. The results are shown in Figures 79, 80, and 81. As shown in the figures, temperature does influence the distance measurement, and this effect is dependent on the position of the inner float, which the author found curious. At the 351mm position (Figure 81), the data show that high temperatures do not have an effect, but cold temperatures did. Similarly, at the 151mm position, the cold temperature did not seem to impact the measurement but the high temperature did. The consistency/repeatability of the measurement for the most part is not apparent. However, this was not the case at 151mm, where the data show that consistency was significantly impacted at 37.8C.

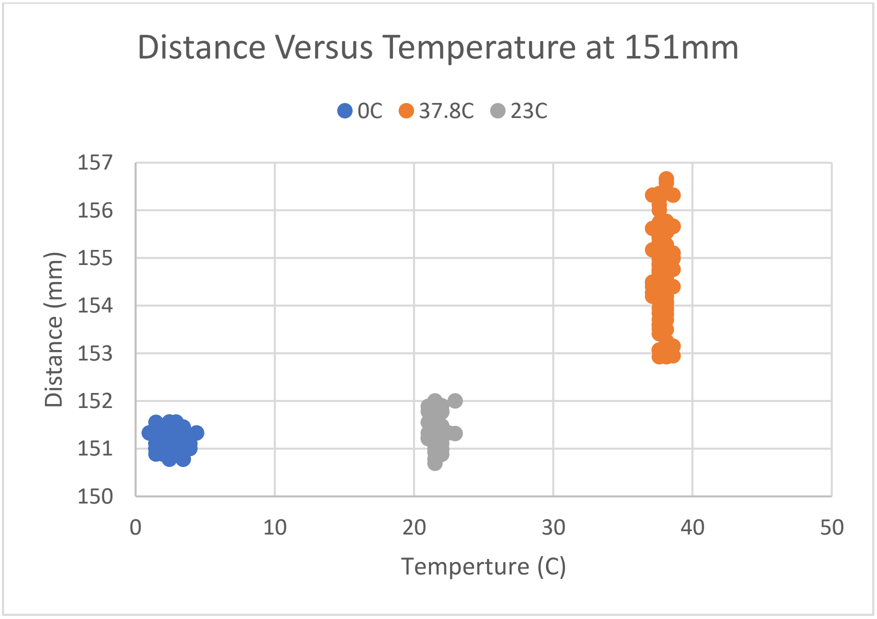


**Figure 79: 151mm Time-of-Flight Sensor Data at Three Different Temperatures.**

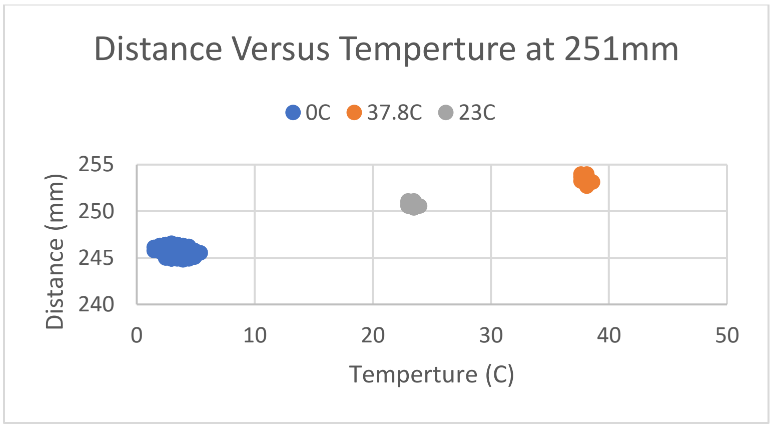


**Figure 80: 251mm Time-of-Flight Sensor Data at Three Different Temperatures.**

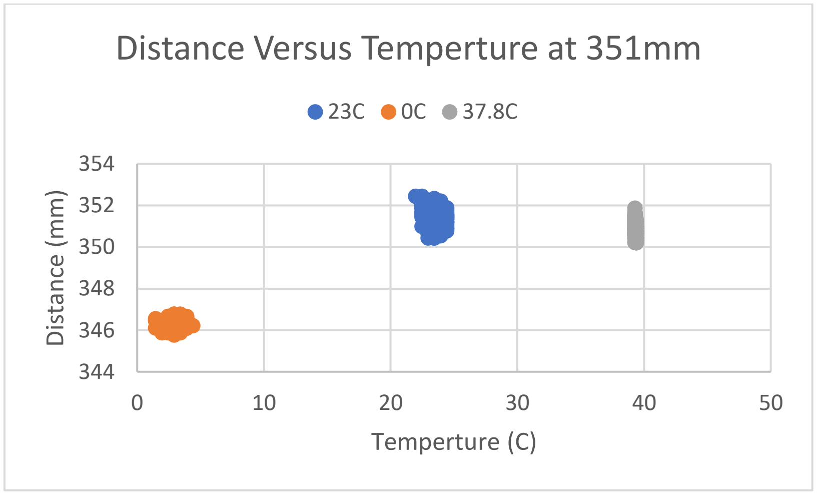


**Figure 81: 351mm Time-of-Flight Sensor Data at Three Different Temperatures.**

## Accuracy and Repeatability Verification of the Temperature Sensor

In addition to verifying the accuracy and repeatability of the TOF sensor, another purpose of the testing was to verify the accuracy of the LM35 temperature sensor. The specific LM35 sensor used in the POC was the LM35DZ, which has an accuracy of ±1.5°C, which does not meet the requirements. This sensor was used instead of the LM35CZ, which does feature an accuracy of ±1.0°C, because of availability issues. The result of this testing is shown in Figure 82. Figure 82 shows that the LM35DZ is not suitable for this application due to its limited accuracy of ±1.5°C [38].

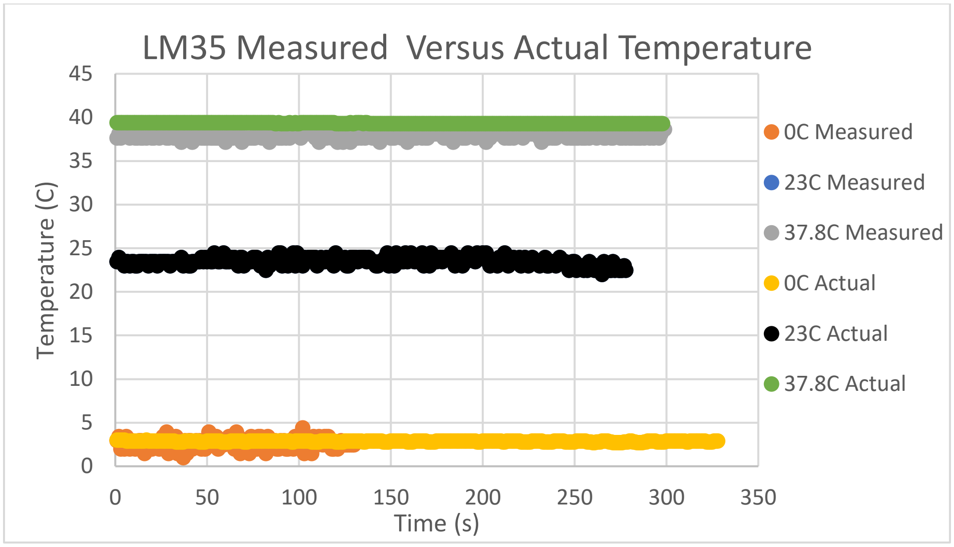


**Figure 82: LM35DV Temperature Measurement Verification.**

## LLI Firmware Rev 2.0 Verification

The last bit of testing that was done was verifying that the agent volume-to-weight calculation functioned correctly. Because FM-200 nor 3M Novec 1230 could be used, water was used instead, since the specific gravities of the three chemicals are similar. For this test, a 5-gallon pail was used as the agent storage cylinder, along with a calibrated scale to verify the weight and the author’s laptop, as shown in Figure 83.

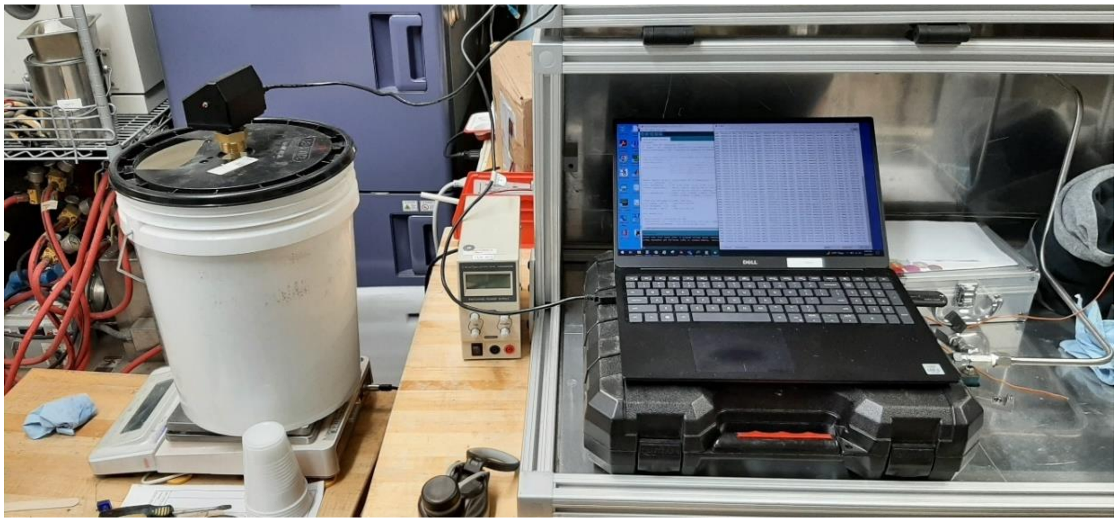


**Figure 83: Agent Volume-to-Weight Conversion Verification Setup.**

Figure 84 shows that if the dimensional information for the storage container is known and is accurate, the TOF - based LLI sensor, and its volume-to-weight conversion algorithm perform as they should.  In addition, it shows that the TOF - based LLI can be extremely accurate.

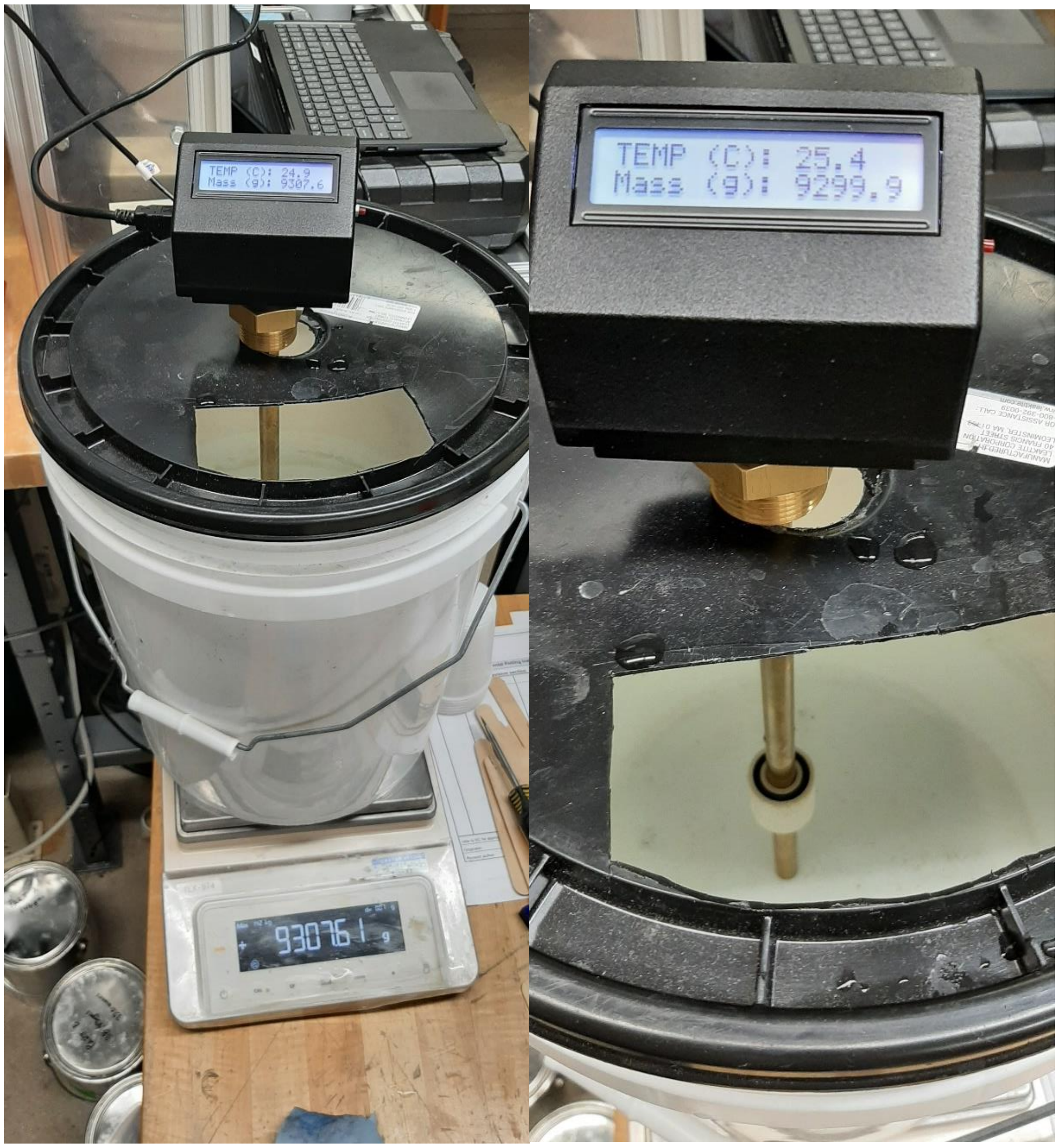


**Figure 84: Verification of the LLI's Volume to Weight Conversion Algorithm.**

### LLI Firmware with User Interface Rev 8.0 Verification

Through the testing process, two bugs were discovered in the "LLI Firmware with User Interface Rev 8.0" program. They are as follows:

- Bug One - The first bug is tied to the serial communications between the LLI and the host device. If the user transmits an alphabetical character instead of an integer, it will result in the LLI program entering a never-ending loop of transmitting the device menu and the error message associated with an incorrect entry.
  - Root Cause – Has not been determined, but it is rooted in how the program handles non-integer data types communications.
- Bug Two - When data recording is enabled, and the display refresh rate is set to continuous, the LLI must be reset for data recording to be enabled. However, if data recording is enabled and the display refresh rate is set to single, the LLI does not need to be reset for data recording to work.
  - Root Cause – Has not been determined, but the issue is believed to be caused by when and how the interrupts associated with those functions are enabled.

## Conclusions

The purpose of this MSOE MSE Capstone Project was to design a TOF-based LLI to replace the current Dip Tape LLIs used to verify the weight of FM-200 and 3M Novec 1230 fire suppression agent used in Clean Agent Fire Suppression System agent storage cylinders. The assembled POC units have shown that in general, TOF-based sensors -- specifically the ST FlightSense$^{TM}$ VL53L1X -- are capable of accurately determining agent weight based on distance to a float. With the addition of the user interface, the POC design has taken another step to becoming a marketable product. Through the addition of the user interface the end user now has the power to choose between multiple agent types, set the dimensions of the cylinder that the LLI is attached to, set the refresh rate of the LLI's display, set the device to record agent mass with eight different recording rates, and to set the units of measure to either Metric or US Imperial.

## Recommendations

The work that has been done so far on the VL53L1X TOF LLI has shown that the technology can perform the task. However, through this development process, several challenges have emerged that need further investigation or development. For example, to have a marketable product, a more robust and efficient calibration method will be required. The current method is cumbersome and time consuming and requires far too much human input. Beyond the calibration inefficiencies, further investigation is required on the effects that temperature has on the VL53L1X sensor measurements and ways to mitigate those effects. Additionally, further testing is required to verify that the sensor will work with FM-200 and 3M Novec 1230. Furthermore, the results of the accuracy and repeatability testing for the LM35DZ temperature sensor shows that it does not meet the required accuracy of ±1.0°C, so a readily available temperature sensor that meets that requirement will need to be found [38].

On the software side, additional testing and development is also required. During the limited testing of the latest LLI firmware, two bugs were discovered. Both bugs require the LLI to be reset to recover from the issues. On the development side, using an

EEPROM to store data may be problematic due to the limited number of writes.  One workaround for this issue, if a suitable alternative cannot be found, would be to limit the data recording rates to twice per day.  Additionally, the memory size of the EEPROM is inadequate for a data recording application.  Lastly, the current user interface lacks polish and should be replaced with a more user-friendly application-based user interface.

## References

[1] TLX Technologies, 2022, “Custom Solenoids and Solenoid Valves”, [Internet, WWW], Address: https://www.tlxtech.com [Accessed: 11 November 2022].

[2] Dickinson, Micah. 11 March 2022. “What is a Clean Agent?”. [Internet, WWW]. Available: Available from Vanguard Website. Address: https://vanguard-fire.com/what-is-a-clean-agent/ [Accessed: 11 November 2022]. A copy of this article is available from the website.

[3] Instrumentation Tools. 2022. “What is Dip Tape Level Measurement?” [Internet, WWW]. *Available:* Available from Instrumentation Tool’s Website. Address: https://instrumentationtools.com/dip-tape-level-measurement/ [Accessed: 11 November 2022]. A copy of this article is available from the website.

[4] Wallulis, Karl. 30 April 2018. “How to Prevent Parallax Error”. [Internet, WWW]. *Available:* Available from Sciencing Website. Address: https://sciencing.com/prevent-parallax-error-10000073.html [Accessed: 11 November 2022]. A copy of this article is available from the website.

[5] National Fire Protection Association. 2022. “NFPA 2001 Standard on Clean Agent Fire Extinguishing Systems”. [Internet, WWW, PDF]. *Available:* Available from NFPA Website. Address: https://link.nfpa.org/free-access/publications/2001/2022 [Accessed: 11 November 2022]. A copy of this article is in the student’s possession and may be consulted by contacting the student at icstumpe@msoe.edu. A copy of this article is available from the website.

[6] Chemetron Fire Systems. 16 April 2004. “FM-200$^{TM}$ Sigma Series Engineered System Design, Installation, Operation & Maintenance Manual”. [Internet,

WWW, PDF]. *Available:* Available from Chemetron Fire Systems Website. Address: https://fm200.co.id/documentation/manual-book/Chemetron%20FM-200%20Sigma%20Series%20Engineered%20System.pdf [Accessed: 11 November 2022]. A copy of this article is available from the website.

[7] Fire Trace. 6 July 2018. “35 Bar Engineered Clean Agent Fire Suppression System Design, Installation, Operation, and Maintenance Manual”. [Internet, WWW, PDF]. *Available:* Available from Fire Trace Website. Address: http://www.firetrace.com/hubfs/DIOMs/FTF000003-B%2035-Bar-Novec-1230-DIOM-Manual.pdf [Accessed: 11 November 2022]. A copy of this article is available from the website.

[8] ST. 2023. “Time-of-Flight Sensor – Overview”. [Internet, WWW]. *Available:* Available from ST Website. Address: https://www.st.com/en/imaging-and-photonics-solutions/time-of-flight-sensors.html [Accessed: 30 January 2023]. A copy of this article is available from the website.

[9] Seed Studio. 2023. “What is a Time-of-Flight Sensor and How does a TOF Sesnor Work”. [Internet, WWW]. *Available:* Available from Seed Studio Website. Address: https://www.seeedstudio.com/blog/2020/01/08/what-is-a-time-of-flight-sensor-and-how-does-a-tof-sensor-work/ [Accessed: 30 January 2023]. A copy of this article is available from the website.

[10] Terabee. 2023. “Time-of-Flight Principle”. [Internet, WWW]. *Available:* Available from Terabee Website. Address: https://www.terabee.com/time-of-flight-principle/#:~:text=Time%2Dof%2DFlight%20(ToF)%20sensors%20are%20used

%20for,the%20distance%20between%20the%20points [Accessed: 30 January 2023]. A copy of this article is available from the website.

[11] Piech, Marcin [Inventor] Jedryczka, Cezary [Inventor] Szelag, Wojciech [Inventor] Wojciecjowski, Rafal [Inventor] and Witczak, Tadeusz Pawel. 4 June 2020. "Magnetic Trap Suppression Tank Level Sensor" International Publication Number W0 2020/112241 A1.

[12] Dickinson, Micah. 11 March 2022. "What is a Clean Agent?". [Internet, WWW]. Available: Available from Vanguard Website. Address: https://vanguard-fire.com/what-is-a-clean-agent/ [Accessed: 11 November 2022]. A copy of this article is available from the website.

[13] Puchovsky, Milosh. May 2011. "Clean agent fire suppression systems". [Internet, WWW, Database]. *Available:* Available from MSOE Library's Summon Discovery Service. Address: https://www.proquest.com/docview/1023363682?accountid=9445&parentSessionId=lS0oVYHJqom0yTsHDVl%2F0XbRB%2B0i6MH%2FUH5XDaFkl00%3D [Accessed: 11 November 2022]. A copy of this article is in the student's possession and may be consulted by contacting the student at icstumpe@msoe.edu.

[14] Draper III, M. Lee PE, May 2017. "Clean Agent Fire Suppression for Mission Critical Facilities". [Internet, WWW, Database]. *Available:* Available from MSOE Library's Summon Discovery Service. Address: [Accessed: 11 November 2022]. A copy of this article is in the student's possession and may be consulted by contacting the student at icstumpe@msoe.edu.

[15] Hurley, Morgan J. 2016. “SFPE Handbook of Fire Protection Engineering, 5" Ed.”. [Book]. *Available:* Available from MSOE Library. [Accessed: 11 November 2022].

[16] Frontier Fire Protection. 2023. “FM-200 Description”. [Internet, WWW]. *Available:* Frontier Fire Protection Website. Address: https://www.frontierfireco.com/productsservices/fm-200/ [Accessed: 30 January 2023]. A copy of this article is available from the website.

[17] Vigstol, Derek. 21 April 2021. “OSHA and NFPA 70E: A History of Powerful Protection for Employees on the Job”. [Internet, WWW]. *Available:* National Fire Protection Association Website. Address: https://www.nfpa.org/News-and-Research/Publications-and-media/Blogs-Landing-Page/NFPA-Today/Blog-Posts/2021/04/21/OSHA-and-NFPA-70E-A-History-of-Powerful-Protection-for-Employees-on-the-Job

[18] FM Approvals. April 2013. “Approval Standard for Clean Agent Extinguishing Systems Class Number 5600”. [Internet, WWW, PDF]. *Available:* Available from FM Approvals Website. Address: https://www.fmapprovals.com/approval-standards [Accessed: 11 November 2022]. A copy of this article is in the student’s possession and may be consulted by contacting the student at icstumpe@msoe.edu.

[19] UL Solutions. 29 August 2012. “UL Standard for Safety For Halocarbon Clean Agent Extinguishing System Units UL 2166”. [Internet, WWW, PDF]. *Available:* Available form UL Solutions Website. Address: https://www.shopulstandards.com/ [Accessed: 11 November 2022]. A copy of

this article is in the student’s possession and may be consulted by contacting the student at icstumpe@msoe.edu.

[20] UL Solutions. 29 October 2003. “UL 864 Control Units and Accessories for Fire Alarm Systems”. [Internet, WWW, PDF]. *Available:* Available form UL Solutions Website. Address: https://www.shopulstandards.com/ [Accessed: 11 November 2022]. A copy of this article is in the student’s possession and may be consulted by contacting the student at icstumpe@msoe.edu.

[21] Gems Sensors & Controls. 2021. “Products Liquid Level Sensors”. [Internet, WWW]. *Available:* Available from Gems Sensors and Controls Website. Address: https://www.gemssensors.com/product/liquid-level-sensors [Accessed: 11 November 2022]. A copy of this article is available from the website.

[22] Hunt James A. 3 November 2007. “Level Sensing of Liquids and Solids – A Review of the Technologies”. [Internet, WWW, Database]. *Available:* Available from MSOE Library’s Summon Discovery Service. Address: https://www.proquest.com/docview /226858206/fulltextPDF/CF2A120285E148BCPQ/1?accountid=9445 [Accessed: 11 November 2022]. A copy of this article is in the student’s possession and may be consulted by contacting the student at icstumpe@msoe.edu.

[23] Sino-Inst. “Measure & Control Solutions”. [Internet, WWW]. *Available:* Available from Sino-Inst Website. Address: https://www.drurylandetheatre.com/ senors-for-tank-level-measurement/ [Accessed: 11 November 2022]. A copy of this article is available from the website.

[24] Kari E. Maatta and Juha Tapio Kostamovaara "High-accuracy liquid level meter based on pulsed time of flight principle", Proc. SPIE 3100, Sensors, Sensor

Systems, and Sensor Data Processing, (25 September 1997);
https://doi.org/10.1117/12.287754

[25] X. Liu, B. J. Corbin, Stacy Morris Bamberg, William R. Provancher, and Eberhard Bamberg "Optical system to detect volume of medical samples in labeled test tubes," Optical Engineering 47(9), 094402 (1 September 2008). https://doi.org/10.1117/1.2978953

[26] Clark, Reece R. [Inventor] and Barmore, Gaston C. Jr. 4 June 2020. “Optical Tank-Level Gauge” United States Paten Number US5585786

[27] Fraser, Gordon, Bryce. 9 December 1982. “Fluid Level Indicator” International Publication Number WO82/04316A1

[28] Lee, Kyung Mi. 16 May 2014. “Water Level Sensing Apparatus” South Korea Patent Number KR20150004237U

[29] Chen, Jianwen. 12 December 2015. “Magneto-strictive Liquid Level” China Patent Number CN105486382A

[30] Chan, Eric Y [Inventor] and Koshinz, Dennis G. 02 May 2020. “Non-Contact Time-of-Flight Fuel Level Sensor Using Plastic Optical Fiber” European Paten Number US5585786

[31] Fuping, Qu. 24 April 2017. “Liquid Extinguisher Steel Cylinder Liquid Level Detection Device” China Patent Number CN206772394U

[32] Liverani, Maurizio. 11 November 2016. “Device for measuring the level of a liquid in a container, in particular for fire protection systems” German Patent Number DE202016006955U1

[33] Piech, Marcin [Inventor] Witczak, Tadeusz Pawel [Inventor] Wawrzyniak, Beata I [Inventor] Majchrzak, Lukasz [Inventor] Milcarek, Dawid. 04 June 2020. “Adaptable Suppression Tank Level Sensor” International Publication Number WO2020/112218A1

[34] Cypress Semiconductor Corporation. 2009-20010. “Quadrature Decoder Document Number: 001-61295 Rev. *A”. [Internet, WWW, PDF]. *Available:* Available from Infineon Technologies Website. Address: https://www.infineon.com/dgdl/Infineon-Quadrature_Decoder_(QuadDec)_Component_QuadDec_V1.50-Software%20Module%20Datasheets-v03_00-EN.pdf?fileId=8ac78c8c7d0d8da4017d0e9659412050 [Accessed: 01 January 2023]. A copy of this article is available from the website.

[35] University of Cambridge. 18 August 2017. “Maths in a Minute: How does laser Interferometry Work?”. [Internet, WWW]. *Available:* Available from Plus Website. Address: https://plus.maths.org/content/maths-minute-how-does-laser-interferometry-work [Accessed: 01 February 2023]. A copy of this article is available from the website.

[36] Collins, Danielle. 22 May 2018. “How Do Magnetostrictive Sensors Work?”. [Internet, WWW]. *Available:* Available from Linear Motion Tips Website. Address: https://www.linearmotiontips.com/how-do-magnetostrictive-sensors-work/ [Accessed: 01 February 2023]. A copy of this article is available from the website.

[37] ST Microelectronics. 2018 “VL53L1X Datasheet”. [Internet, WWW, PDF]. *Available:* Available from ST Microelectronics Website. Address:

https://www.st.com/resource/en/datasheet/vl53l1x.pdf [Accessed: 01 February 2023]. A copy of this article is available from the website.

[38] Utmel Electronic. 24 August 2021. "An Overview of Development Board". [Internet, WWW]. *Available:* Available from Utmel Electronic Website. Address: https://www.utmel.com/blog/categories/pcb/an-overview-of-development-board [Accessed: 01 February 2023]. A copy of this article is available from the website.

[39] SparkFun Electronics. 2022. "Qwiic Distance Sensor (VL53L1X, VL53L4CD) Hookup Guide". [Internet, WWW]. *Available:* Available from SparkFun Electronics Website. Address: https://learn.sparkfun.com/tutorials/qwiic-distance-sensor-vl53l1x-vl53l4cd-hookup-guide [Accessed: 01 February 2023]. A copy of this article is available from the website.

[40] Arduino. 2021. "Arduino Uno Rev3: Overview". [Internet, WWW]. *Available:* Available from Arduino Website. Address: https://store.arduino.cc/products/arduino-uno-rev3 [Accessed: 01 February 2023]. A copy of this article is available from the website.

[41] Atmel Corporation. 2015. "Atmel Atmega328P 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash Datasheet". [Internet, WWW, PDF]. *Available:* Available from the Microchip Website. Address: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf [Accessed: 01 February 2023]. A copy of this article is available from the website.

[42] National Semiconductor. November 2000. "LM35 Precision Centigrade Temperature Sensors". [Internet, WWW, PDF]. *Available:* Available from the Texas Instruments Website. Address:

https://rocelec.widen.net/view/pdf/lhmv29xclw/NATLS06060-1.pdf?t.download=true&u=5oefqw [Accessed: 01 February 2023]. A copy of this article is available from the website.

[43] Ada, Lady. 2022. “RGB LCD Shield Overview”. [Internet, WWW]. *Available:* Available from the Adafruit Website. Address: https://learn.adafruit.com/rgb-lcd-shield/using-the-rgb-lcd-shield?view=all&gclid=Cj0KCQiAz9ieBhCIARIsACB0oGKmqtyW_PWbCwhXE1mMbYNIwlK67BfDr5B6OmEaimHas4-tkm91kH4aAutxEALw_wcB [Accessed: 01 February 2023]. A copy of this article is available from the website.

[44] ST Microelectronics. 2018 “Application note - AN5191 - Using the programmable region of interest (ROI) with the VL53L1X”. [Internet, WWW, PDF]. *Available:* Available from ST Microelectronics Website. Address: https://www.st.com/resource/en/application_note/an5191-using-the-programmable-region-of-interest-roi-with-the-vl53l1x-stmicroelectronics.pdf [Accessed: 01 February 2023]. A copy of this article is available from the website.

[45] ST Microelectronics. 2018 “UM2356 User Manual VL53L1X API User Manual”. [Internet, WWW, PDF]. *Available:* Available from ST Microelectronics Website. Address: https://www.st.com/resource/en/user_manual/um2356-vl53l1x-api-user-manual-stmicroelectronics.pdf [Accessed: 01 February 2023]. A copy of this article is available from the website.

[46] Llamas, Luis. 12 April 2022. “Arduino-Interpolation”. [Internet, WWW]. *Available:* Available from GitHub Inc Website. Address: https://github.com/luisllamasbinaburo/Arduino-Interpolation [Accessed: 01 February 2023]. A copy of this article is available from the website.

[47] Toppr. 2022 “Linear Interpolation Formula”. [Internet, WWW]. *Available:* Available from Toppr Website. Address: https://www.toppr.com/guides/maths-formulas/linear-interpolation-formula/#:~:text=Formula%20of%20Linear%20Interpolation,-Its%20simplest%20formula&text=This%20formula%20is%20using%20coordinates,1%7D%20are%20the%20first%20coordinates [Accessed: 01 February 2023]. A copy of this article is available from the website.

[48] DuPont. November 2009. “DuPont$^{TM}$ FM-200 (HFC-227ea) Fire Extinguishing Agent Properties, Uses, Storage, and Handling”. [Internet, WWW, PDF]. *Available:* Available from the Fire Security Website. Address: https://www.firesecurity.gr/Pdf/k23261_FM-200_PUSH.pdf [Accessed: 11 November 2022]. A copy of this article is available from the website.

[49] Thangavel, Author. February 2019. “Arduino Interrupts Tutorial”. [Internet, WWW]. *Available:* Available from Circuit Digest Website. Address: https://circuitdigest.com/microcontroller-projects/arduino-interrupt-tutorial-with-examples# [Accessed: 06 May 2023]. A copy of this article is available from the website.

[50] Hymel, Shawn. May 2023. “Getting Started with STM32 - Timers and Timer Interrupts”. [Internet, WWW]. *Available:* Available from Digi-Key Electronics Website. Address: https://www.digikey.com/en/maker/projects/getting-started-with-stm32-timers-and-timer-interrupts/d08e6493cefa486fb1e79c43c0b08cc6#:~:text=Timers%20are%20one%20of%20the,and%20even%20run%20operating%20systems [Accessed: 06 May 2023]. A copy of this article is available from the website.

[51] Arduino [Author] Bagur, José [Author] Chung, Taddy. April 2023. “Arduino Memory Guide”. [Internet, WWW]. *Available:* Available from Arduino Website. Address: https://docs.arduino.cc/learn/programming/memory-guide [Accessed: 06 May 2023]. A copy of this article is available from the website.

# Appendix A: LLI Firmware Rev 1.0

```
/*
  Reading distance from the laser-based ST Microelectronics VL53L1X sensor and
  temperature from the TI LM35 temperature sensor.
  By: Ian Stumpe
  MSOE MSE Capstone Project
  Date: December 10, 2022
  License: This code is public domain, but you buy me a beer if you use this and we meet
someday (Beerware license).
*/
#include <SparkFun_VL53L1X.h> //SparkFun ST VL53L1X shield Library:
http://librarymanager/All#SparkFun_VL53L1X
#include <Wire.h> //Allows Arduino boards to communicate with I2C/TWI Devices.
#include <Adafruit_RGBLCDShield.h>  //Adafruit RGB LCD Shield Library:
https://learn.adafruit.com/rgb-lcd-shield
#include <avr/sleep.h> //this AVR library contains the methods that control the sleep
modes
// The Adafruit RGBLCD Shield uses the I2C SCL and SDA pins.
Adafruit_RGBLCDShield lcd = Adafruit_RGBLCDShield();
// These #defines set the backlight color of the Adafruit RGB LCD Display
#define OFF 0x0
#define RED 0x1
#define YELLOW 0x3
```

```
#define GREEN 0x2

#define TEAL 0x6

#define BLUE 0x4

#define VIOLET 0x5

#define WHITE 0x7

#define interruptPin 2 //Pin we are going to use to wake up the Arduino

SFEVL53L1X distanceSensor; //Initializes ST VL53L1X distance sensor.

//Variables

int val; //Raw ADC temperature value

int tempPin = A0; //Analog pin connected to TI temperature sensor

//float olddistance = 100; //Variable for tracking old distance measurement

void setup(void)

{

  Wire.begin(); //Initialize I2C BUS

  lcd.begin(16, 2); //Initialize LCD Display, sets LCD's number of columns and rows.

  lcd.setBacklight(WHITE); //Set LCD backlight color

  Serial.begin(115200); //Initialize Serial BUS for troubleshooting.

  //Verifies ST VL53L1 sensor is wired correctly. Begin returns 0 on a good init

  if (distanceSensor.begin() != 0)

  {

    Serial.println("Sensor failed to begin. Please check wiring. Freezing...");

    while (1);

  }
  Serial.println("Sensor online!");
  distanceSensor.setROI(4, 4, 196); //Sets the VL53L1's Range of Interest (receiver grid
size)
   //Sets the VL53L1 timing budget which is the amount of time (ms) over which a
measurement is taken
   distanceSensor.setTimingBudgetInMs(200);
   //Sets the VL53L1's distance measurement mode Short = 1.3m Long = 4m
   distanceSensor.setDistanceModeShort();
}
void loop(void)
{
  float distance = 0; //Variable to hold distance measurement data
  distanceSensor.startRanging(); //Write configuration bytes to initiate measurement
  for (int i=0; i < 1000; i++) //Loop to take 10 measurements to get the average
measurement
  {
    while (!distanceSensor.checkForDataReady()) //Checks if a measurement is ready
    {
      delay(1);
    }
    distance = distance + distanceSensor.getDistance(); //Get the result of the measurement
in mm from the sensor
    //Serial.println(distanceSensor.getSignalPerSpad());
  }
  distanceSensor.clearInterrupt(); //Clears interrupt caused by the .getDistance command.
  distanceSensor.stopRanging(); //Stop taking measurements
  Serial.print("SUM (mm): ");
  Serial.print(distance);
  distance = distance/1000; //Calculate the average
  Serial.print(" AVG (mm): ");
  Serial.print(distance);
  //Variable to hold converted distance measurement (mm to inches)
  //float distanceInches = distance;
  //float distanceInches = (distance * 0.0393701); //Convert mm to inches
  //Variable to hold converted distance measurement (inches to feet)
  //float distanceFeet = distanceInches / 12.0; //Convert inches to feet
  val = analogRead(tempPin); //Get temperature from TI sensor
  float mv = (val/1023.0)*5000; //Convert raw ADC data to voltage value
  float cel = mv/10; //Convert voltage to degree Celsius
  Serial.print(" Temperature (C): ");
  Serial.println(cel);
  lcd.setCursor(0, 0); //Set display cursor
  lcd.print("DIST (mm): ");
  lcd.print(distance,0);
  lcd.setCursor(0, 1); //Set display cursor
  // print ambient temp
```

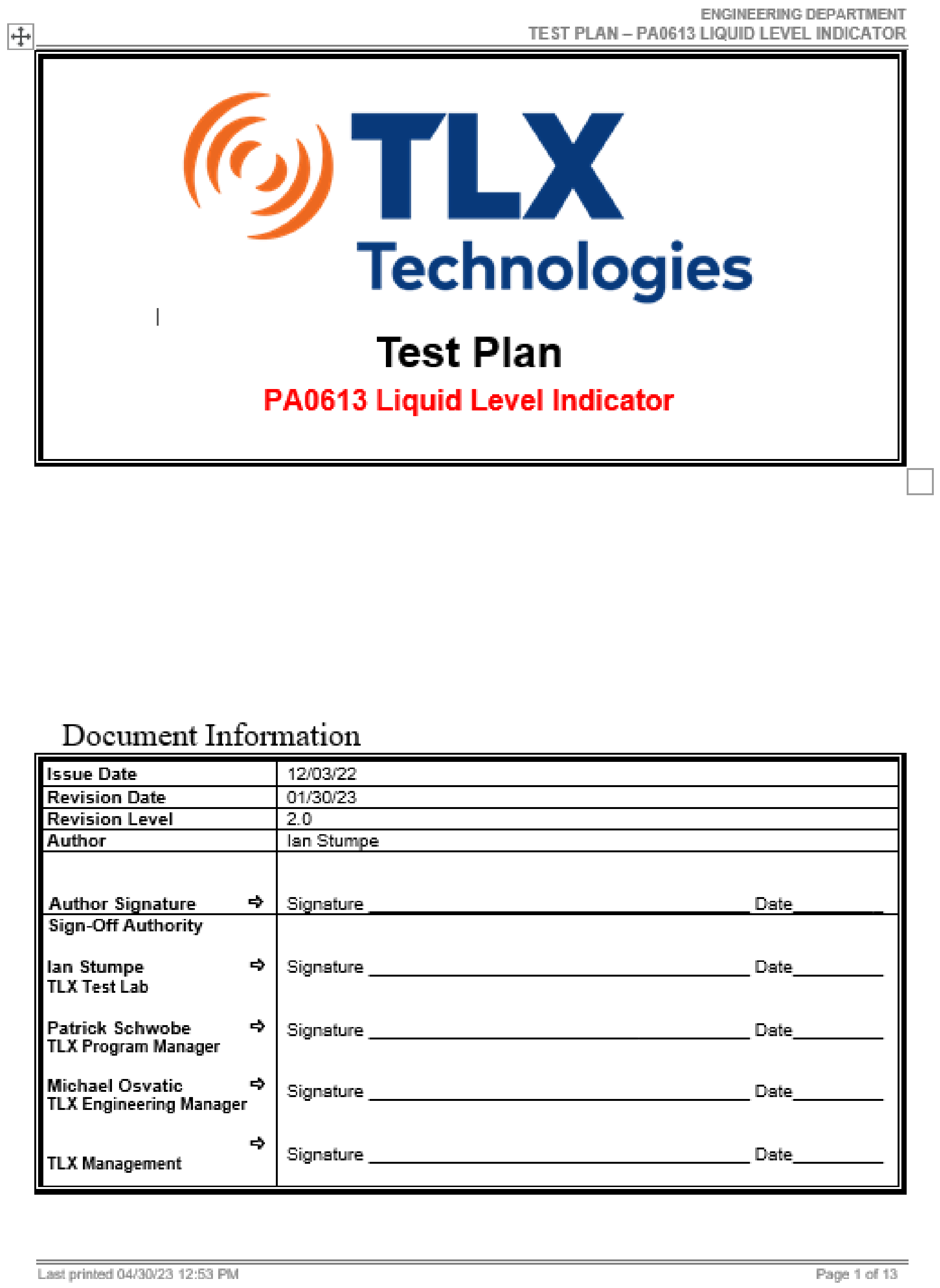
```
  lcd.print("TEMP (C): ");
  lcd.print(cel,1);
}
```

## Appendix B:  ROI Calibration Code

```
#include <SparkFun_VL53L1X.h> //SparkFun ST VL53L1A shield Library:
http://librarymanager/All#SparkFun_VL53L1X
#include <Wire.h> //Allows Arduino boards to communicate with I2C/TWI Devices.
SFEVL53L1X distanceSensor; //Initializes ST VL53L1X distance sensor.
//Uncomment the following line to use the optional shutdown and interrupt pins.
//SFEVL53L1X distanceSensor(Wire, SHUTDOWN_PIN, INTERRUPT_PIN); //Not
currently used
//Variables
int val; //Raw ADC temperature value
int tempPin = A0; //Analog pin connected to TI temperature sensor
int dist = 352;   //Actual distance to float
int ROIcent = 0;
void setup(void)
{
  Wire.begin(); //Initialize I2C BUS

  Serial.begin(115200); //Initialize Serial BUS for troubleshooting.

  //Verifies ST VL53L1 sensor is wired correctly. Begin returns 0 on a good init
  if (distanceSensor.begin() != 0)
  {
    Serial.println("Sensor failed to begin. Please check wiring. Freezing...");
    while (1);
  }
  Serial.println("Sensor online!");
  //Serial.println(distanceSensor.getSignalPerSpad());
 }
void loop(void)
{
  distanceSensor.setROI(4, 4, ROIcent); //Sets the VL53L1's Range of Interest (receiver
grid size)
  //Sets the VL53L1 timing budget which is the amount of time (ms) over which a
measurement is taken
  distanceSensor.setTimingBudgetInMs(200);
  // Sets the period in between measurements. Must be greater than or equal to the timing
budget. Default is 100 ms.
  //distanceSensor.setIntermeasurementPeriod(4000);
  //Sets the VL53L1's distance measurement mode Short = 1.3m Long = 4m
  distanceSensor.setDistanceModeLong();
  float distance = 0; //Variable to hold distance measurement data
  distanceSensor.startRanging(); //Write configuration bytes to initiate measurement
 //Serial.println(distanceSensor.getSignalPerSpad());
  for (int i=0; i < 500; i++) //Loop to take 10 measurements to get the average
measurement
  {
    while (!distanceSensor.checkForDataReady()) //Checks if a measurement is ready
    {
      delay(1);
    }
    distance = distance + distanceSensor.getDistance(); //Get the result of the measurement
in mm from the sensor
  }
  distance = distance/500; //Calculate the average
  distanceSensor.clearInterrupt(); //Clears interrupt caused by the .getDistance command.
  distanceSensor.stopRanging(); //Stop taking measurements
  //Serial.print(" ROI: ");
  //Serial.println(ROIcent);
  while (distance >= 111 && distance <= 113)
  {
    Serial.print("ROI Good: ");
    Serial.println(ROIcent);
    delay(1000);
    break;
  }
  while (ROIcent == 255)
  {
    Serial.print("Stop");
  }
    ROIcent = ROIcent + 1;
}
```

## Appendix C: MSE Capstone LLI Test Plan Rev 2.0

This appendix features a copy of the test plan for the TOF - based LLI that is the focus of the MSE Capstone Project.



TLX Technologies

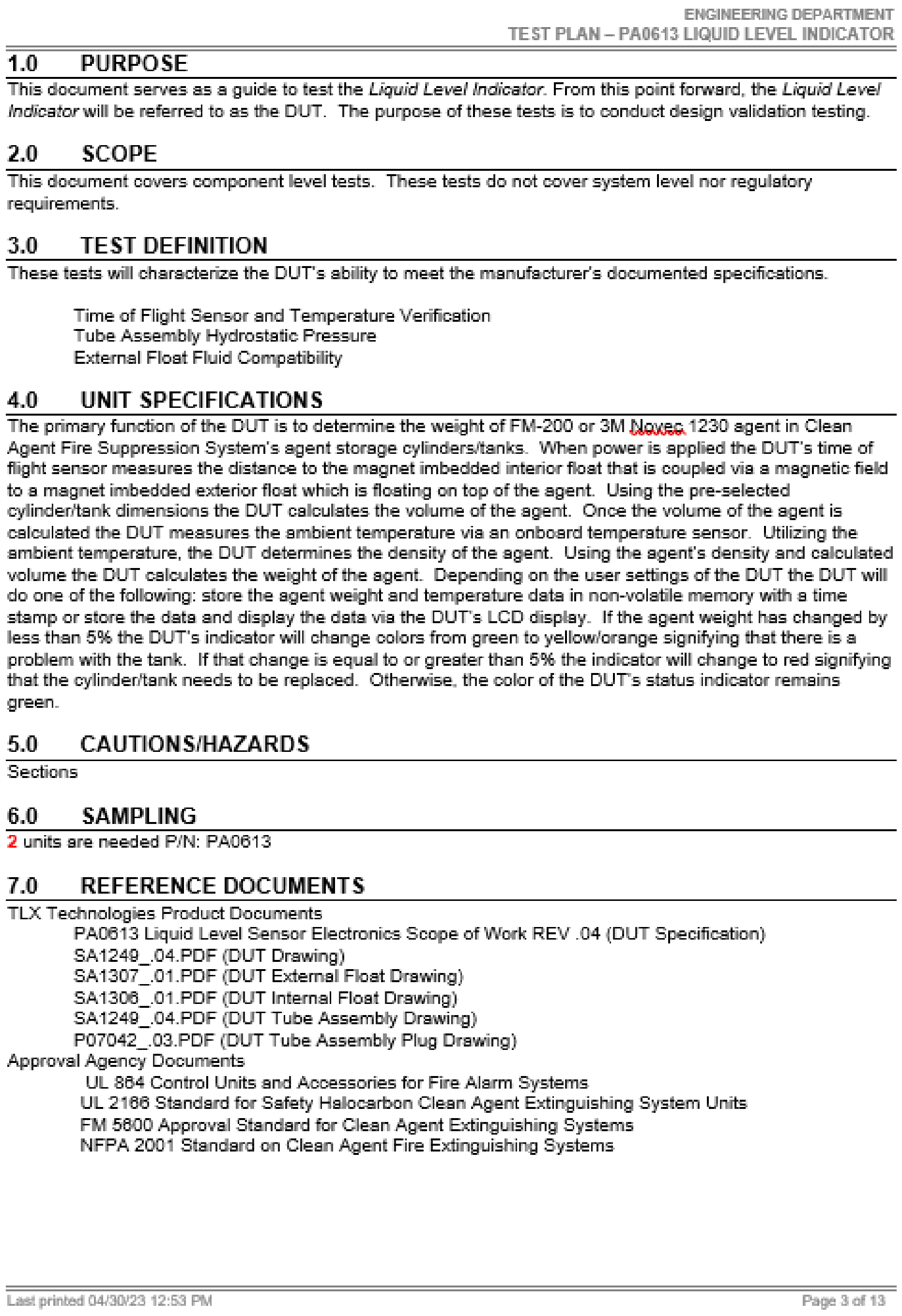
**Test Plan**

**PA0613 Liquid Level Indicator**

Document Information

| | |
|---|---|
| Issue Date | 12/03/22 |
| Revision Date | 01/30/23 |
| Revision Level | 2.0 |
| Author | Ian Stumpe |
| Author Signature ➪ | Signature ______________ Date ________ |
| Sign-Off Authority<br><br>Ian Stumpe ➪<br>TLX Test Lab | Signature ______________ Date ________ |
| Patrick Schwobe ➪<br>TLX Program Manager | Signature ______________ Date ________ |
| Michael Osvatic ➪<br>TLX Engineering Manager | Signature ______________ Date ________ |
| ➪<br>TLX Management | Signature ______________ Date ________ |

## Table of Contents



## 1.0 PURPOSE

This document serves as a guide to test the *Liquid Level Indicator*. From this point forward, the *Liquid Level Indicator* will be referred to as the DUT. The purpose of these tests is to conduct design validation testing.

## 2.0 SCOPE

This document covers component level tests. These tests do not cover system level nor regulatory requirements.

## 3.0 TEST DEFINITION

These tests will characterize the DUT's ability to meet the manufacturer's documented specifications.

Time of Flight Sensor and Temperature Verification
Tube Assembly Hydrostatic Pressure
External Float Fluid Compatibility

## 4.0 UNIT SPECIFICATIONS

The primary function of the DUT is to determine the weight of FM-200 or 3M Novec 1230 agent in Clean Agent Fire Suppression System's agent storage cylinders/tanks. When power is applied the DUT's time of flight sensor measures the distance to the magnet imbedded interior float that is coupled via a magnetic field to a magnet imbedded exterior float which is floating on top of the agent. Using the pre-selected cylinder/tank dimensions the DUT calculates the volume of the agent. Once the volume of the agent is calculated the DUT measures the ambient temperature via an onboard temperature sensor. Utilizing the ambient temperature, the DUT determines the density of the agent. Using the agent's density and calculated volume the DUT calculates the weight of the agent. Depending on the user settings of the DUT the DUT will do one of the following: store the agent weight and temperature data in non-volatile memory with a time stamp or store the data and display the data via the DUT's LCD display. If the agent weight has changed by less than 5% the DUT's indicator will change colors from green to yellow/orange signifying that there is a problem with the tank. If that change is equal to or greater than 5% the indicator will change to red signifying that the cylinder/tank needs to be replaced. Otherwise, the color of the DUT's status indicator remains green.

## 5.0 CAUTIONS/HAZARDS

Sections

## 6.0 SAMPLING

2 units are needed P/N: PA0613

## 7.0 REFERENCE DOCUMENTS

TLX Technologies Product Documents
PA0613 Liquid Level Sensor Electronics Scope of Work REV .04 (DUT Specification)
SA1249_.04.PDF (DUT Drawing)
SA1307_.01.PDF (DUT External Float Drawing)
SA1306_.01.PDF (DUT Internal Float Drawing)
SA1249_.04.PDF (DUT Tube Assembly Drawing)
P07042_.03.PDF (DUT Tube Assembly Plug Drawing)
Approval Agency Documents
UL 864 Control Units and Accessories for Fire Alarm Systems
UL 2166 Standard for Safety Halocarbon Clean Agent Extinguishing System Units
FM 5600 Approval Standard for Clean Agent Extinguishing Systems
NFPA 2001 Standard on Clean Agent Fire Extinguishing Systems

## 8.0 TEST DATA

**8.1** Test Levels

- 8.1.1 Status I – The function performs as designed during and after test
- 8.1.2 Status II – The function does not perform as designed during the test but returns automatically to normal operation after the test.
- 8.1.3 Status III – The function does not perform as designed during the test and does not return to normal operation without a simple driver/passenger intervention such as turning off/on the DUT or cycling the ignition switch after the disturbance is removed.
- 8.1.4 Status IV – The function does not perform as designed during and after the test and cannot be returned to proper operation without more extensive intervention such as disconnecting and reconnecting the battery or power feed. The function shall not have sustained any permanent damage because of the testing.

**8.2** Pass/Fail Criteria

- 8.2.1 Functional Check
  - 8.2.1.1 DUT's Time of Flight Sensor and Temperature Verification
    - 8.2.1.1.1 DUT's Time of Flight measurements shall be within ±1% of actual measurement.
    - 8.2.1.1.2 DUT's Temperature measurements shall be within ±1% of actual measurement.
  - 8.2.1.2 DUT's Tube Assembly Hydrostatic Pressure Testing
    - 8.2.1.2.1 No defects shall be observed.
  - 8.2.1.3 DUT's External Float Fluid Compatibility
    - 8.2.1.3.1 No defects shall be observed.
    - 8.2.1.3.2 Weight shall not change more than ±1% of the original weight

## 9.0 TEST SET-UPS

**9.1** Overview

9.1.1 This section shall assist in defining testing setups by providing basic diagrams for each of the following areas of validation testing:

9.1.1.1 Functional Checkouts

9.1.2 These diagrams are purposefully basic. Their intent is to assist in illustrating the following:

9.1.2.1 Situations where testing should be restricted to one DUT at a time

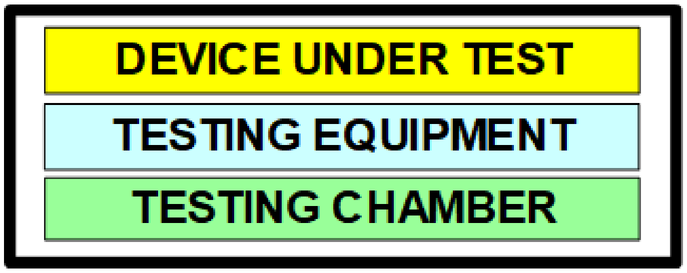
9.1.2.2 Opportunities where testing may be applied to multiple DUTs concurrently

9.1.2.3 Specific circuits that should be individually tested

9.1.2.4 Specific DUT testing configurations (power and load connections)

9.1.3 It is important to remember that each required test has its own unique setup as defined in the applicable test specification. The setup in the test specification shall be the ruling setup. The diagrams shown in this section shall be used as a starting foundation from which to construct each individual test setup per the applicable test specification.

**9.2** Diagram Legend



## 9.3 Functional Checks

### 9.3.1 DUT's Time of Flight Sensor and Temperature Verification Setup

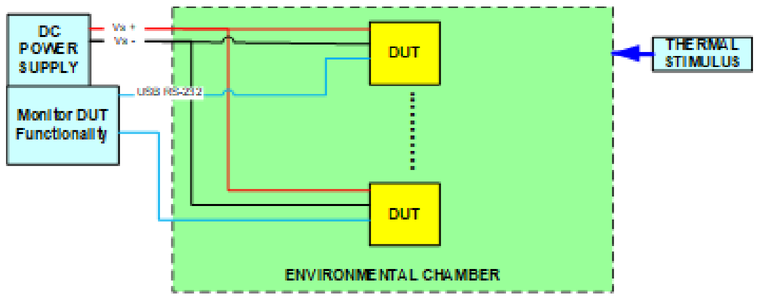


Figure 2

9.3.1.1 Setup:

9.3.1.1.1 Power shall be applied to the DUT using a 7 to 9V DC source via the DUT's 2.1mm center-positive power jack or using the DUT's USB connection.

9.3.1.1.2 The DUT shall be tested at three different temperatures 0°C, 23°C and 49°C.

9.3.1.1.3 The DUT shall soak at each temperature for 30mins before height measurements are recorded. Additionally, record the measured temperature via the DUT's internal temperature sensor and compare to actual ambient temperature.

9.3.1.1.4 At each temperature three different float positions (heights) shall be used to verify the accuracy of the DUT. These three different positions shall be recorded.

9.3.1.1.5 At each float position 10 height measurements shall be taken and recorded. The height measurements can either be taken from the DUT's display or via the DUT's USB serial port connection.

9.3.1.1.6 The average of the 10 height measurements shall be compared to the actual height of the float.

9.3.1.2 Test Monitoring:

9.3.1.2.1 Chamber temperature shall be monitored using a calibrated temperature sensor. This temperature shall be used to verify the temperature given by the DUT.

9.3.1.2.2 DUT's functionality shall be monitored via the DUT's USB serial port connection.

### 9.3.2 DUT's Tube Assembly Hydrostatic Pressure Testing Setup

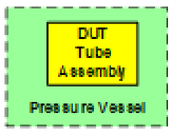


Figure 3

9.3.3 General Setup:

9.3.3.1 The DUT Tube Assembly shall be connected to the pressure vessel using the same methods as in the production application.

9.3.3.2 The pressure shall be slowly increased to a maximum pressure of 1000 psi, checking the DUT for proper functionality periodically.

9.3.4 Test Monitoring:

9.3.4.1 The pressure of the pressure vessel shall be monitored using a calibrated pressure gauge or transducer.

9.3.5 DUT's External Float Fluid Compatibility Testing Setup Diagram

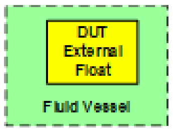


Figure 4

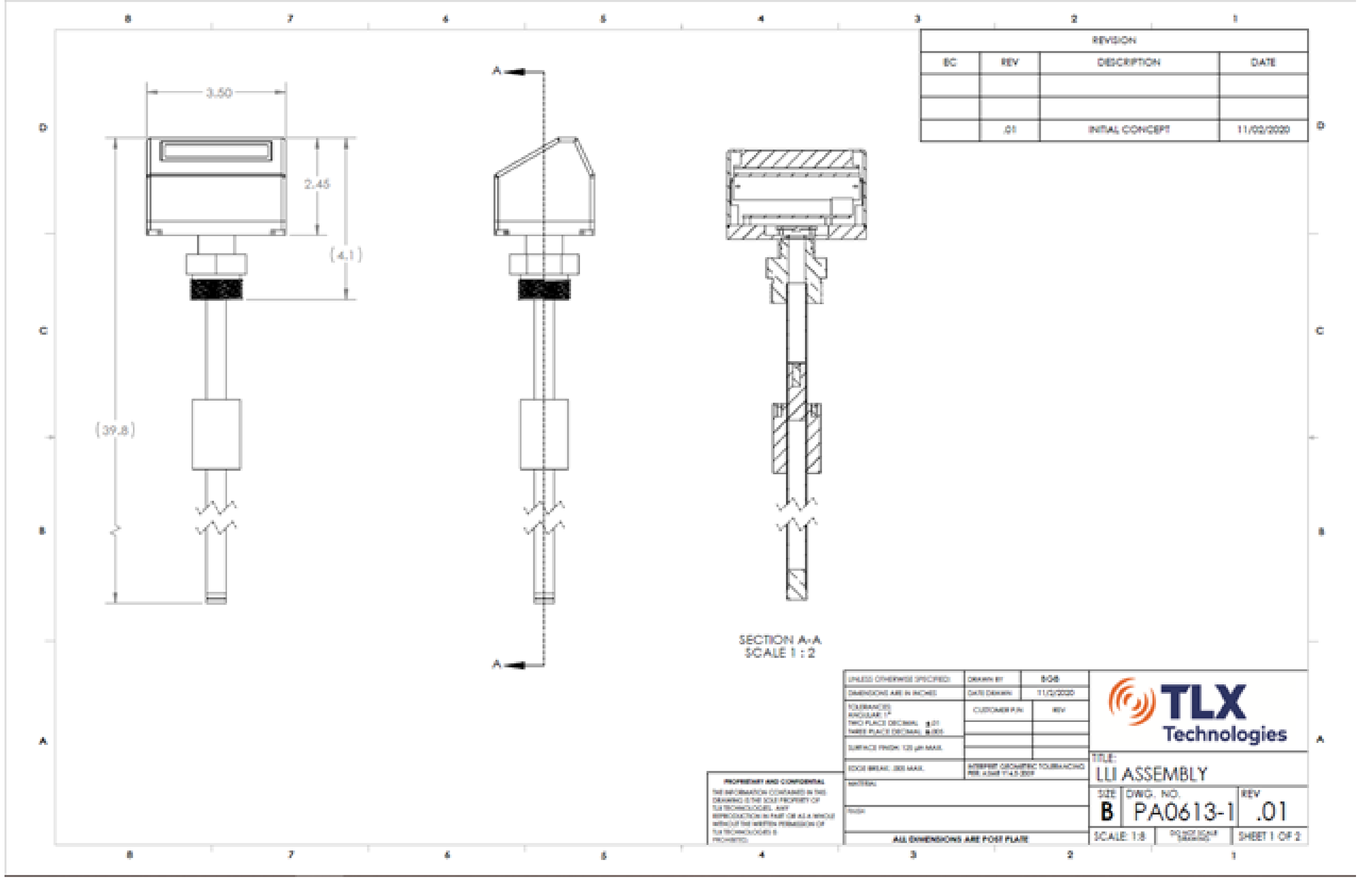
9.3.5.1 General Setup:

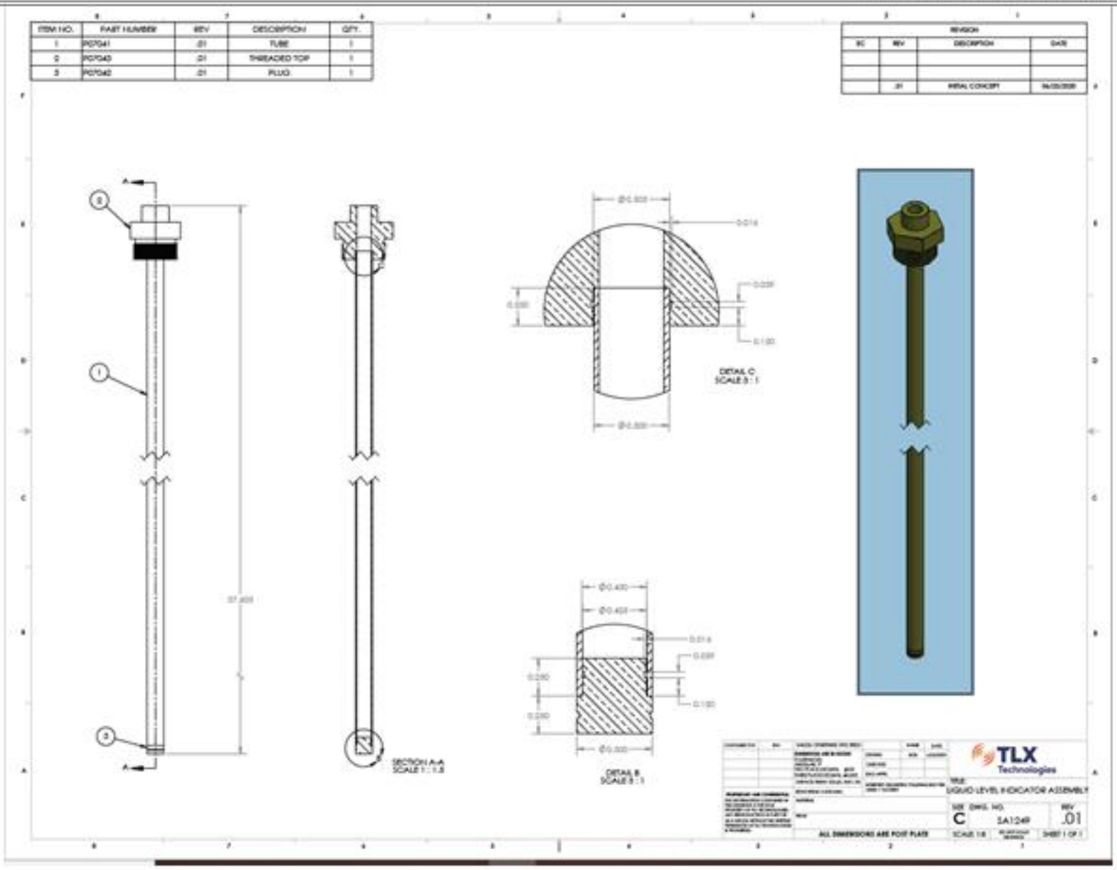
9.3.5.1.1 The DUT's External Float shall be exposed to the agent for 7 days.
a) The following agents shall be tested FM-200 and 3M Novec 1230

9.3.5.1.2 The weight of the DUT's External Float shall be recorded prior to testing and checked daily during testing.

9.3.5.1.3 The DUT's External Float shall be visually inspected prior to testing and shall be inspected daily for any change in physical appearance.

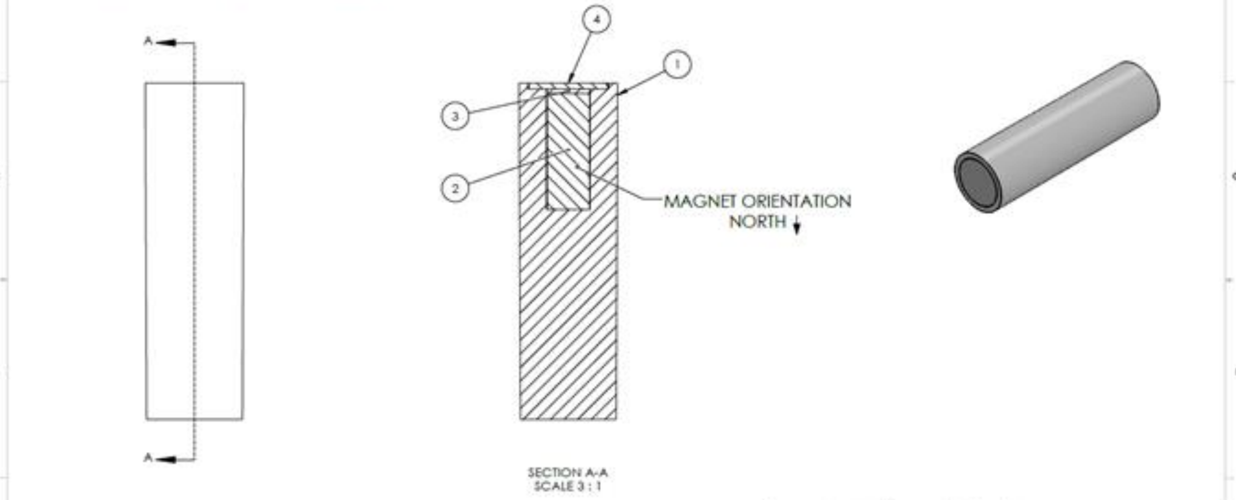
## 10.0 DRAWINGS




ITEM NO.
PART NUMBER
REV
DESCRIPTION
QTY.
1
P07041
.01
TUBE
1
2
P07043
.01
THREADED TOP
1
3
P07042
.01
PLUG
1
REVISION
DESCRIPTION
DATE
.01
INITIAL CONCEPT
SECTION A-A
SCALE 1 : 1.5
DETAIL C
SCALE 5 : 1
DETAIL B
SCALE 5 : 1
TLX Technologies
TITLE:
LIQUID LEVEL INDICATOR ASSEMBLY
SIZE C
DWG. NO. SA1249
REV .01
ALL DIMENSIONS ARE POST PLATE
SCALE 1:8
SHEET 1 OF 1

| ITEM NO. | PART NUMBER | REV | DESCRIPTION | QTY. |
|---|---|---|---|---|
| 1 | P07381 | .03 | INTERNAL FLOAT | 1 |
| 2 | P07380 | .01 | MAGNET | 1 |
| 3 | P07821 | .01 | ADHESIVE | 1 |
| 4 | P07378 | .03 | WASHER | 1 |

| REVISION | | | |
|---|---|---|---|
| EC | REV | DESCRIPTION | DATE |
| | .01 | INITIAL CONCEPT | 11/2/2020 |
| | .02 | P07821 WAS P07389 | 5/19/2021 |
| | .03 | P07381 & P07378 TO REV .03 | 06/03/2021 |



NOTES:

1. MAGNET IS PRESS FIT INTO FLOAT

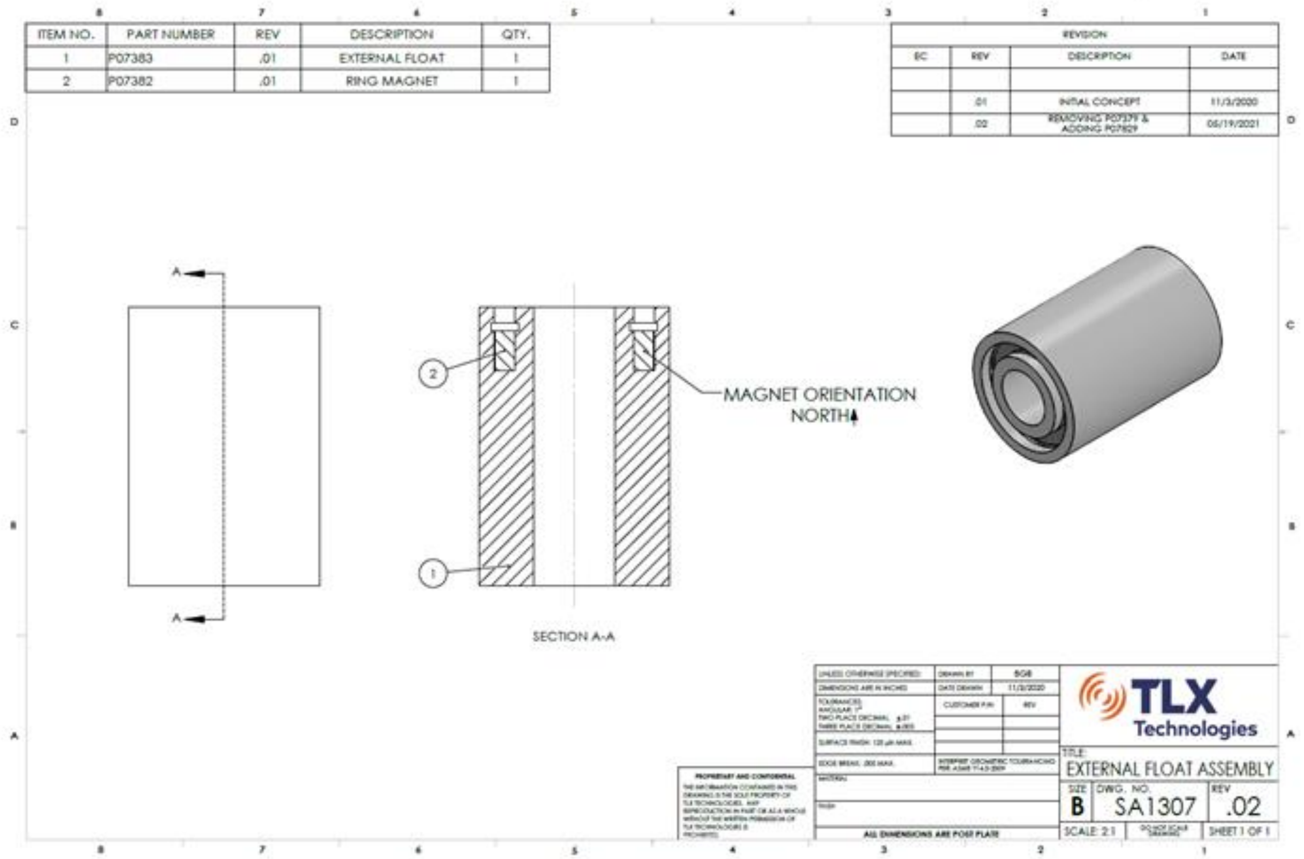
| | | |
|---|---|---|
| UNLESS OTHERWISE SPECIFIED: | DRAWN BY | BQB |
| DIMENSIONS ARE IN INCHES | DATE DRAWN | 11/2/2020 |
| TOLERANCES: ANGULAR: 1° TWO PLACE DECIMAL ±.01 THREE PLACE DECIMAL ±.005 | CUSTOMER P/N | REV |
| SURFACE FINISH: 125 µin MAX. | | |
| EDGE BREAK: .005 MAX. | INTERPRET GEOMETRIC TOLERANCING PER ASME Y14.5-2009 | |
| MATERIAL | | |
| FINISH | | |
| ALL DIMENSIONS ARE POST PLATE | | |

PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF TLX TECHNOLOGIES. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF TLX TECHNOLOGIES IS PROHIBITED.

TLX Technologies

TITLE:
INTERNAL FLOAT ASSEMBLY

| SIZE | DWG. NO. | REV |
|---|---|---|
| B | SA1306 | .03 |
| SCALE: 2:1 | DO NOT SCALE DRAWING | SHEET 1 OF 1 |



| ITEM NO. | PART NUMBER | REV | DESCRIPTION | QTY. |
|---|---|---|---|---|
| 1 | P07383 | .01 | EXTERNAL FLOAT | 1 |
| 2 | P07382 | .01 | RING MAGNET | 1 |

| REVISION | | | |
|---|---|---|---|
| EC | REV | DESCRIPTION | DATE |
| | | | |
| | .01 | INITIAL CONCEPT | 11/3/2020 |
| | .02 | REMOVING P07379 & ADDING P07829 | 06/19/2021 |

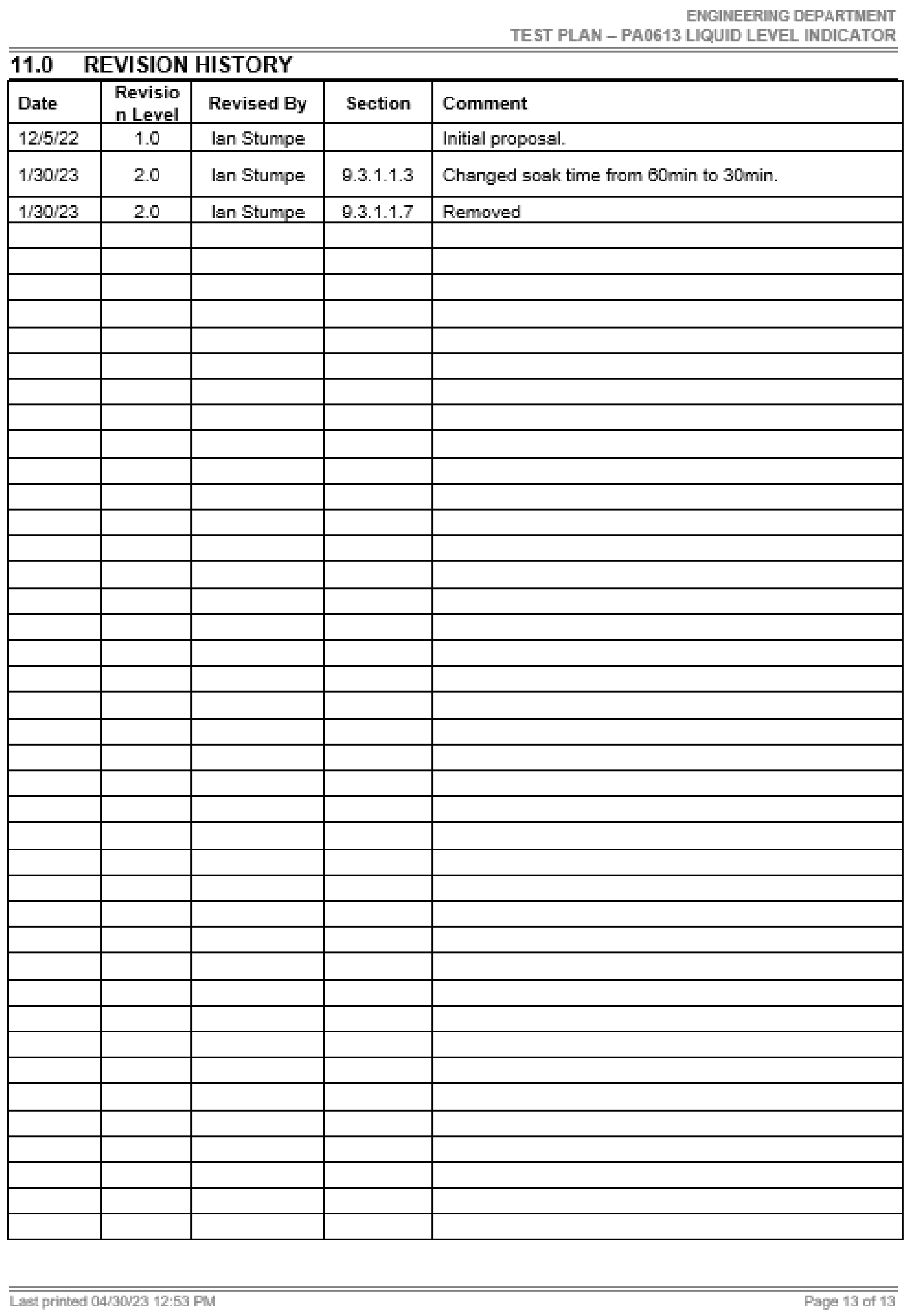
| | | |
|---|---|---|
| UNLESS OTHERWISE SPECIFIED: | DRAWN BY | BGB |
| DIMENSIONS ARE IN INCHES | DATE DRAWN | 11/3/2020 |
| TOLERANCES: ANGULAR: 1° TWO PLACE DECIMAL ±.01 THREE PLACE DECIMAL ±.005 | CUSTOMER P/N | REV |
| SURFACE FINISH: 125 µin MAX. | | |
| EDGE BREAK: .005 MAX. | INTERPRET GEOMETRIC TOLERANCING PER: ASME Y14.5-2009 | |
| MATERIAL | | |
| FINISH | | |
| ALL DIMENSIONS ARE POST PLATE | | |

PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF TLX TECHNOLOGIES. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF TLX TECHNOLOGIES IS PROHIBITED.

TLX Technologies

TITLE: EXTERNAL FLOAT ASSEMBLY

| SIZE | DWG. NO. | REV |
|---|---|---|
| B | SA1307 | .02 |
| SCALE: 2:1 | DO NOT SCALE DRAWING | SHEET 1 OF 1 |



## 11.0 REVISION HISTORY

| Date | Revision Level | Revised By | Section | Comment |
|---|---|---|---|---|
| 12/5/22 | 1.0 | Ian Stumpe | | Initial proposal. |
| 1/30/23 | 2.0 | Ian Stumpe | 9.3.1.1.3 | Changed soak time from 60min to 30min. |
| 1/30/23 | 2.0 | Ian Stumpe | 9.3.1.1.7 | Removed |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

## Appendix D: LLI Firmware Rev 2.0

```
/*
  Reading distance from the laser-based ST Microelectronics VL53L1X sensor and
  temperature from the TI LM35 temperature sensor.
  By: Ian Stumpe
  MSOE MSE Capstone Project
  Date: December 10, 2022
  License: This code is public domain but you buy me a beer if you use this and we meet
  someday (Beerware license).
*/
#include <SparkFun_VL53L1X.h> //SparkFun ST VL53L1A shield Library:
http://librarymanager/All#SparkFun_VL53L1X
#include <Wire.h> //Allows Arduino boards to communicate with I2C/TWI Devices.
#include <Adafruit_RGBLCDShield.h>  //Adafruit RGB LCD Shield Library:
https://learn.adafruit.com/rgb-lcd-shield
#include <avr/sleep.h> //this AVR library contains the methods that control the sleep
modes
//Copyright (c) 2019 Luis Llamas
#include <InterpolationLib.h> //This library contains the methods that interpolate agent
density based on temperature
// The Adafruit RGBLCD Shield uses the I2C SCL and SDA pins.
Adafruit_RGBLCDShield lcd = Adafruit_RGBLCDShield();
```

```
//Optional interrupt and shutdown pins.

#define SHUTDOWN_PIN 2

#define INTERRUPT_PIN 3

// These #defines set the backlight color of the Adafruit RGB LCD Display

#define OFF 0x0

#define RED 0x1

#define YELLOW 0x3

#define GREEN 0x2

#define TEAL 0x6

#define BLUE 0x4

#define VIOLET 0x5

#define WHITE 0x7

#define interruptPin 2 //Pin we are going to use to wake up the Arduino

SFEVL53L1X distanceSensor; //Initializes ST VL53L1X distance sensor.

//Uncomment the following line to use the optional shutdown and interrupt pins.

//SFEVL53L1X distanceSensor(Wire, SHUTDOWN_PIN, INTERRUPT_PIN); //Not
currently used

//Variables

int val; //Raw ADC temperature value

int tempPin = A0; //Analog pin connected to TI temperature sensor

//float olddistance = 100; //Variable for tracking old distance measurement

/*

//FM-200 agent density (kg/m^3) values based on temperature (C)
```

```
const int numValues = 24;
double xValues[24] = { -15, -10, -5, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70,
75, 80, 85, 90, 95, 100}; // Temperature values
double yValues[24] = { 1539.7, 1522.1, 1504.2, 1486.0, 1467.3, 1448.2, 1428.6, 1408.4,
1387.7, 1366.2, 1344.0, 1320.9, 1296.7, 1271.4, 1244.8, 1216.5, 1186.2, 1153.6, 1117.9,
1078.2, 1032.8, 978.6, 907.8, 786.8}; //Density values
*/
//Water density (g/cm^3) values based on temperature (C)
const int numValues = 15;
double xValues[24] = { 0.00, 4.00, 4.40, 10.00, 15.60, 21.00, 26.70, 32.20, 37.80, 48.90,
60.00, 71.10, 82.20, 93.30, 100.00}; // Temperature values
double yValues[24] = { 0.99987, 1.00, 0.99999, 0.99975, 0.99907, 0.99802, 0.99669,
0.9951, 0.99318, 0.9887, 0.98338, 0.97729, 0.97056, 0.96333, 0.95865}; //Density values
(g/cm^3)
//Tank size and shape cylinder (5gal pail
float radius = 136.5;    //mm 136.5 142.875
float thickness = 0.0;  //mm
float height = 398.0;    //mm 406.4 402 346
void setup(void)
{
  Wire.begin(); //Initialize I2C BUS
  lcd.begin(16, 2); //Initialize LCD Display, sets LCD's number of columns and rows.
  lcd.setBacklight(WHITE); //Set LCD backlight color
  Serial.begin(115200); //Initialize Serial BUS for troubleshooting.
  /* //Used for putting the LCD in sleep mode and waking it up.
  pinMode(LED_BUILTIN,OUTPUT);
  pinMode(LED_BUILTIN,HIGH);
  pinMode(interruptPin, INPUT_PULLUP);
  */
  //Verifies ST VL53L1 sensor is wired correctly. Begin returns 0 on a good init
  if (distanceSensor.begin() != 0)
  {
    Serial.println("Sensor failed to begin. Please check wiring. Freezing...");
    while (1);
  }
  Serial.println("Sensor online!");
  distanceSensor.setROI(4, 4, 230); //Sets the VL53L1's Range of Interest (receiver grid
size)
   //Sets the VL53L1 timing budget which is the amount of time (ms) over which a
measurement is taken
   distanceSensor.setTimingBudgetInMs(200);
   // Sets the period in between measurements. Must be greater than or equal to the timing
budget. Default is 100 ms.
   //distanceSensor.setIntermeasurementPeriod(4000);
   //Sets the VL53L1's distance measurement mode Short = 1.3m Long = 4m
   distanceSensor.setDistanceModeShort();
```

```
}
void loop(void)
{
  float distance = 0; //Variable to hold distance measurement data
  distanceSensor.startRanging(); //Write configuration bytes to initiate measurement
  for (int i=0; i < 1000; i++) //Loop to take 10 measurements to get the average
measurement
  {
    while (!distanceSensor.checkForDataReady()) //Checks if a measurement is ready
    {
      delay(1);
    }
    distance = distance + distanceSensor.getDistance(); //Get the result of the measurement
in mm from the sensor
    //Serial.println(distanceSensor.getSignalPerSpad());
  }
  distanceSensor.clearInterrupt(); //Clears interrupt caused by the .getDistance command.
  distanceSensor.stopRanging(); //Stop taking measurements
  Serial.print("SUM: ");
  Serial.print(distance);
  distance = distance/1000; //Calculate the average
  Serial.print(" AVG: ");
  Serial.print(distance);
  //calculate volume in mm
  float volume = 3.14 * (sq(radius - thickness)) * (height - distance); //mm^3
  Serial.print(" Volume (mm3): ");
  Serial.print(volume);
  /*
  //FM-200
  //Variable to hold converted distance measurement (mm to meters)
  float volumeM = (volume / 1000000000); //Convert mm^3 to m^3
  Serial.print(" Volume (m): ");
  Serial.print(volumeM,5);
*/
  //Water
  //Variable to hold converted distance measurement (mm to cm)
  float volumeM = (volume / 1000); //Convert mm^3 to cm^3
  Serial.print(" Volume (cm3): ");
  Serial.print(volumeM,5);
  val = analogRead(tempPin); //Get temperature from TI sensor
  //Uncomment the next two lines if Black POC is being used. Black POC uses LM35
temp sensor which converts voltage straight to C
  float mv = (val/1023.0)*5000; //Convert raw ADC data to voltage value
  float cel = mv/10; //Convert voltage to degree Celsius, Used for Black POC
  // Uncomment the next two lines if RED POC is being used. Red POC uses LM355
which converts voltage to kelvin requiring an extra conversion to C
```

```
  //float kelvin = mv/10.0; //Convert voltage to kelvin
  //float cel = kelvin - 273.0; //Convert kelvin to C, used for Red POC
  double density = Interpolation::Linear(xValues, yValues, numValues, cel, true); //prints
agent density
  //Serial.print(" Density (kg/m^3): ");
  //Serial.print(" Density (g/mm^3): ");
  Serial.print(" Density (g/cm^3): ");
  Serial.print(density);
  //calculate mass of agent
  float mass = density * volumeM;
  //float mass = density * volume;
  //Serial.print(" Mass (kg): ");
  Serial.print(" Mass (g): ");
  Serial.println(mass,2);
  //print ambient temp
  lcd.setCursor(0, 0); //Set display cursor
  lcd.print("TEMP (C): ");
  lcd.print(cel,1);
  //print agent mass/weight
  lcd.setCursor(0, 1); //Set display cursor
  //lcd.print("Mass (kg): ");
  lcd.print("Mass (g): ");
  lcd.print(mass,2);  }
```

## Appendix E: LLI Firmware with Interrupt Rev 1.0

```
/*
Reading distance from the laser-based ST Microelectronics VL53L1X sensor and
temperature from the TI LM35 temperature sensor.
By: Ian Stumpe
MSOE MSE Capstone Project
Date: December 10, 2022
License: This code is public domain, but you buy me a beer if you use this and we meet
someday (Beerware license).
*/
#include <Wire.h>
#include "SparkFun_VL53L1X.h" //Click here to get the library:
http://librarymanager/All#SparkFun_VL53L1X
#include <Adafruit_RGBLCDShield.h>
#include <utility/Adafruit_MCP23017.h>
#include <avr/sleep.h> //this AVR library contains the methods that control the sleep
modes
// The shield uses the I2C SCL and SDA pins. On classic Arduinos
// this is Analog 4 and 5 so you can't use those for analogRead() anymore
// However, you can connect other I2C sensors to the I2C bus and share
// the I2C bus.
Adafruit_RGBLCDShield lcd = Adafruit_RGBLCDShield();
//Optional interrupt and shutdown pins.
#define SHUTDOWN_PIN 2//not used
#define INTERRUPT_PIN 3
// These #defines make it easy to set the backlight color
#define OFF 0x0
#define RED 0x1
#define YELLOW 0x3
#define GREEN 0x2
#define TEAL 0x6
```

```
#define BLUE 0x4
#define VIOLET 0x5
#define WHITE 0x7
#define interruptPin 2 //Pin we are going to use to wake up the Arduino
SFEVL53L1X distanceSensor;
//Uncomment the following line to use the optional shutdown and interrupt pins.
//SFEVL53L1X distanceSensor(Wire, SHUTDOWN_PIN, INTERRUPT_PIN);
int val;
int tempPin = A0;
void setup(void)
{
  Wire.begin();
  lcd.begin(16, 2);
  lcd.setBacklight(OFF);
  Serial.begin(115200);
  pinMode(LED_BUILTIN,OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  pinMode(LED_BUILTIN,HIGH);
  // set up the LCD's number of columns and rows:
  if (distanceSensor.begin() != 0) //Begin returns 0 on a good init
  {
    Serial.println("Sensor failed to begin. Please check wiring. Freezing...");
    while (1);
  }
  Serial.println("Sensor online!");
   distanceSensor.setROI(4, 4, 71);
   distanceSensor.setTimingBudgetInMs(500);
   //distanceSensor.setIntermeasurementPeriod(2000);
   distanceSensor.setDistanceModeShort();
}
void loop(void)
```

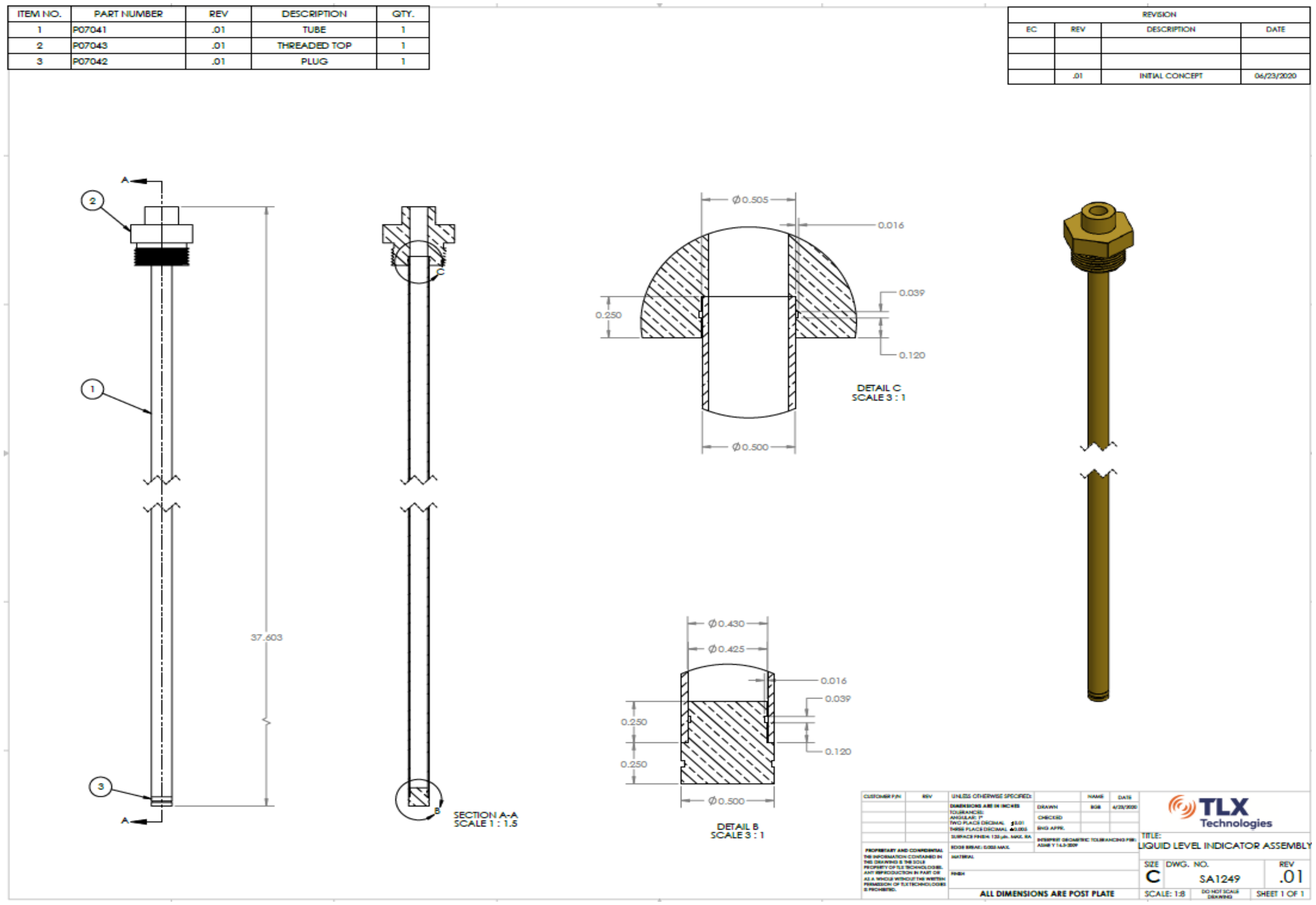
```
{
  delay(5000);  //wait 5 seconds before going to sleep
  Going_To_Sleep();
}
void Going_To_Sleep()
{
  sleep_enable();
  attachInterrupt(digitalPinToInterrupt(interruptPin), wakeUp, LOW);
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);
  digitalWrite(LED_BUILTIN,LOW);
  lcd.clear();
  lcd.setBacklight(OFF);
  delay(1000);
  sleep_cpu();
  digitalWrite(LED_BUILTIN,HIGH);
  distanceSensor.startRanging(); //Write configuration bytes to initiate measurement
  delay(2000);
  while (!distanceSensor.checkForDataReady())
  {
    delay(1);
  }
  int distance = distanceSensor.getDistance(); //Get the result of the measurement from
the sensor
  distanceSensor.clearInterrupt();
  distanceSensor.stopRanging();
  //float distanceT = map(distance,0,914.4,914.4,0);
  float distanceInches = (distance * 0.0393701);
  float distanceFeet = distanceInches / 12.0;

  val = analogRead(tempPin);
  float mv = (val/1023.0)*5000;
  float cel = mv/10;

  lcd.setBacklight(WHITE);
  lcd.setCursor(0, 0);
  // print the number of seconds since reset
  lcd.print("DIST(in): ");
  lcd.print(distanceInches, 3);

  lcd.setCursor(0, 1);
  // print the number of seconds since reset:
  lcd.print("TEMP (C): ");
  lcd.print(cel);
}

void wakeUp()
{
  sleep_disable();
  detachInterrupt(digitalPinToInterrupt(interruptPin));
 }
```

# Appendix F: ROI Center-Point Calibration Data

Table F- 1: ROI Center-Point Calibration Measurement Data.

| Distance = 352 | | | | Distance = 301 ±1 | | | | Distance = 252 ±1 | | | | Distance = 202 ±1 | | | | Distance = 152 ±1 | | | | Distance = 102 ±1 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 230cp | 204cp | 228cp | 196cp | 230cp | 204cp | 228cp | 196cp | 230cp | 204cp | 228cp | 196cp | 230cp | 204cp | 228cp | 196cp | 230cp | 204cp | 228cp | 196cp | 230cp | 204cp | 228cp | 196cp |
| 350.21 | 351 | 351.45 | 349 | 300.22 | 299.89 | 301.22 | 301.11 | 250.34 | 250.44 | 251.22 | 252.55 | 201.34 | 201.78 | 201.89 | 202.22 | 151.11 | 154.55 | 153.33 | 153.55 | 102 | 99.22 | 99.9 | 99.22 |
| 350.22 | 351 | 351.34 | 348.89 | 300.22 | 299.56 | 301.22 | 301.11 | 250.56 | 250.45 | 250.79 | 252.78 | 202.46 | 201.56 | 201.99 | 201.89 | 151.32 | 154 | 153.22 | 153.56 | 102.11 | 99.33 | 99.89 | 99.33 |
| 350.44 | 350.89 | 351.55 | 349.11 | 300.11 | 300.22 | 301.22 | 301.11 | 250.66 | 250.44 | 250.9 | 252.77 | 201.89 | 201.23 | 201.55 | 201.67 | 151.9 | 154.56 | 153.11 | 153.23 | 101.99 | 99.11 | 99.78 | 99.79 |
| 350.44 | 351.11 | 351.45 | 348.66 | 300.45 | 300 | 301 | 300.89 | 250.45 | 250.34 | 251.11 | 252.78 | 201.68 | 201.99 | 201.77 | 201.33 | 151.79 | 154.66 | 153.33 | 153.56 | 102 | 99.22 | 99.78 | 99.33 |
| 350.44 | 350.78 | 351.34 | 348.89 | 300.33 | 300.11 | 301.22 | 301.11 | 250.66 | 250.33 | 251.33 | 253 | 200.77 | 201.67 | 201.44 | 201.34 | 152 | 154.34 | 153.11 | 153.56 | 101.89 | 99.11 | 99.67 | 99.55 |
| 350.34 | 350.78 | 351.45 | 348.89 | 300.1 | 300 | 301.22 | 301.22 | 250.55 | 250.34 | 250.89 | 252.67 | 201 | 201.77 | 201.55 | 201.44 | 151.77 | 154.44 | 152.89 | 153.44 | 101.89 | 99.1 | 99.78 | 99.66 |
| 350.55 | 350.89 | 351.56 | 348.89 | 300.55 | 299.89 | 301 | 301.34 | 250.56 | 250.45 | 250.89 | 252.89 | 201.34 | 201.89 | 202 | 201.56 | 151.33 | 154.66 | 153.34 | 153.22 | 102 | 99.45 | 99.77 | 99.77 |
| 350.34 | 351.01 | 351.22 | 348.78 | 300.34 | 300 | 301.22 | 301.22 | 250.55 | 250.33 | 251 | 252.77 | 201.67 | 201.55 | 201.77 | 201.67 | 151.37 | 154.66 | 153.33 | 153.78 | 102 | 99.45 | 99.55 | 99.89 |
| 350.34 | 350.9 | 351.45 | 348.99 | 300.22 | 299.66 | 301.45 | 301.1 | 250.89 | 250.34 | 251.11 | 252.79 | 202.2 | 201.44 | 202 | 201.45 | 150.69 | 154.22 | 153.11 | 153.77 | 102.11 | 99 | 99.66 | 99.55 |
| 350.56 | 350.78 | 351.45 | 348.89 | 300.22 | 299.9 | 301.11 | 300.89 | 250.44 | 250.45 | 250.89 | 252.89 | 201.42 | 201.67 | 201.77 | 201.66 | 152 | 154.34 | 153.11 | 153.44 | 101.89 | 99 | 99.78 | 100.01 |
| 350.56 | 350.78 | 351.55 | 348.89 | 300.34 | 299.78 | 301.22 | 301.11 | 250.55 | 250.45 | 250.78 | 252.66 | 201.57 | 201.9 | 201.66 | 201.34 | 151.77 | 154.34 | 152.89 | 153.56 | 101.89 | 99.44 | 99.78 | 99.77 |
| 350.44 | 350.78 | 351.45 | 348.89 | 300.11 | 299.66 | 301.34 | 301 | 251 | 250.34 | 251.11 | 252.89 | 201.46 | 201.78 | 202.21 | 201.77 | 151.44 | 154.34 | 153.45 | 153.66 | 101.89 | 99.66 | 99.34 | 99.99 |
| 350.33 | 351.11 | 351.55 | 349.33 | 300.45 | 299.78 | 301.55 | 301.22 | 250.89 | 250.44 | 251.11 | 252.55 | 200.98 | 201.78 | 202.22 | 201.79 | 151.67 | 154.45 | 152.89 | 153.45 | 102 | 99.22 | 100 | 99.9 |
| 350.22 | 350.89 | 351.66 | 349.22 | 300 | 299.78 | 301.22 | 301.11 | 250.79 | 250.45 | 250.78 | 252.79 | 201.46 | 201.67 | 201.89 | 202.11 | 151.44 | 154.66 | 153.11 | 153.56 | 102 | 99.34 | 99.89 | 100 |
| 350.33 | 350.89 | 351.44 | 348.89 | 300.45 | 299.78 | 301.11 | 301.1 | 250.55 | 250.45 | 250.66 | 253 | 201 | 202.11 | 202 | 202.11 | 150.99 | 154.45 | 153.22 | 153.45 | 102 | 99.34 | 100.12 | 99.78 |
| 350.67 | 351.11 | 351.44 | 348.89 | 300.22 | 299.89 | 301.34 | 301.34 | 250.55 | 250.22 | 251 | 253.01 | 201.57 | 201.78 | 201.89 | 202.01 | 150.88 | 154.66 | 153.11 | 153.45 | 102 | 99.34 | 100 | 100.1 |
| 350.11 | 350.89 | 351.45 | 348.78 | 300.11 | 299.78 | 301 | 301.34 | 250.88 | 250.56 | 251.01 | 252.66 | 200.89 | 201.78 | 201.89 | 202.34 | 150.78 | 154.66 | 153.22 | 153.66 | 102 | 99.67 | 99.78 | 100.22 |
| 350.11 | 350.9 | 351.33 | 348.56 | 300.34 | 300 | 301.34 | 301.1 | 251 | 250.34 | 250.77 | 252.76 | 201.34 | 201.56 | 201.89 | 202.56 | 151.77 | 154.44 | 152.89 | 153.45 | 101.9 | 99.55 | 99.67 | 99.9 |
| 350.34 | 350.89 | 351.45 | 349 | 300.66 | 299.89 | 301.23 | 301.11 | 251.01 | 250.33 | 250.78 | 252.55 | 201.65 | 201.56 | 201.89 | 202.33 | 151.23 | 154.44 | 153 | 153.33 | 102 | 99.66 | 100 | 99.78 |
| 350.22 | 351 | 351.33 | 349 | 300.34 | 299.89 | 301.34 | 301.45 | 250.89 | 250.45 | 250.79 | 252.56 | 201.12 | 201.45 | 202 | 202.67 | 151.89 | 154.56 | 152.78 | 153.55 | 102 | 99.22 | 99.9 | 100.11 |
| 350.34 | 350.78 | 351.56 | 348.89 | 300.45 | 299.89 | 301.22 | 301 | 251 | 250.22 | 250.77 | 252.45 | 200.9 | 202 | 201.66 | 202.45 | 151.78 | 154.67 | 152.89 | 153.34 | 101.78 | 99.33 | 100 | 100.11 |
| 350.78 | 350.89 | 351.45 | 348.9 | 300.11 | 299.66 | 301.22 | 301 | 250.9 | 250.45 | 250.89 | 252.89 | 200.44 | 201.89 | 202 | 202.45 | 151.48 | 154.66 | 152.99 | 153.55 | 102 | 99.45 | 99.89 | 100 |
| 350.67 | 350.78 | 351.34 | 349 | 300.11 | 299.89 | 300.89 | 300.9 | 251.11 | 250.34 | 250.89 | 252.89 | 200.45 | 202.1 | 201.89 | 202.55 | 150.92 | 154.45 | 152.89 | 153.34 | 102 | 99.22 | 100.11 | 99.78 |
| 350.56 | 350.67 | 351.56 | 348.89 | 300.44 | 299.56 | 301.22 | 301 | 251 | 250.45 | 250.89 | 252.77 | 201.79 | 202.01 | 202 | 202.23 | 151 | 154.66 | 153.11 | 153.45 | 101.89 | 99.11 | 99.89 | 99.56 |
| 350.44 | 350.89 | 351.56 | 349 | 300.45 | 299.78 | 301.11 | 301.11 | 251.11 | 250.11 | 250.78 | 252.67 | 201.22 | 201.78 | 201.89 | 201.89 | 151.67 | 154.66 | 153.22 | 153.45 | 101.89 | 99.44 | 99.9 | 99.99 |
| 350.22 | 350.45 | 351.66 | 349.1 | 300.22 | 299.89 | 301.11 | 301.45 | 251.01 | 250.34 | 250.78 | 252.9 | 201.46 | 201.89 | 201.89 | 202.23 | 151.33 | 154.78 | 152.89 | 153.44 | 101.89 | 99.22 | 100.11 | 99.78 |
| 350.22 | 350.45 | 351.44 | 349.11 | 300.44 | 299.55 | 301.22 | 301.11 | 250.78 | 250.22 | 250.79 | 252.99 | 201.45 | 201.66 | 202 | 201.89 | 151.22 | 154.89 | 152.89 | 153.23 | 101.89 | 99.34 | 99.79 | 99.9 |
| 350.55 | 350.55 | 351.34 | 348.89 | 300.44 | 300 | 301.34 | 301.11 | 250.89 | 250.45 | 251.22 | 252.88 | 201 | 201.77 | 202 | 201.56 | 151.21 | 154.55 | 152.78 | 153.34 | 102 | 99.45 | 99.67 | 100 |
| 350.66 | 350.89 | 351.44 | 348.89 | 300.44 | 299.78 | 301.33 | 301 | 250.9 | 250.45 | 250.79 | 252.89 | 201.12 | 201.66 | 202.22 | 201.78 | 151.78 | 154.89 | 152.89 | 153.55 | 102 | 99.22 | 99.56 | 100.22 |
| 350.11 | 350.44 | 351.34 | 349.22 | 300.33 | 300 | 301.11 | 301.11 | 250.78 | 250 | 251 | 252.89 | 200.77 | 202 | 201.66 | 200.99 | 151.55 | 154.66 | 152.89 | 153.1 | 101.89 | 99.11 | 99.34 | 100.11 |

# Appendix G: LLI Tube Assembly Drawing



| ITEM NO. | PART NUMBER | REV | DESCRIPTION | QTY. |
|---|---|---|---|---|
| 1 | P07041 | .01 | TUBE | 1 |
| 2 | P07043 | .01 | THREADED TOP | 1 |
| 3 | P07042 | .01 | PLUG | 1 |

| REVISION | | | |
|---|---|---|---|
| EC | REV | DESCRIPTION | DATE |
| | | | |
| | .01 | INITIAL CONCEPT | 06/23/2020 |

| CUSTOMER P/N | REV | UNLESS OTHERWISE SPECIFIED: | | NAME | DATE |
|---|---|---|---|---|---|
| | | DIMENSIONS ARE IN INCHES TOLERANCES: ANGULAR: 1° TWO PLACE DECIMAL ±0.01 THREE PLACE DECIMAL ±0.005 | DRAWN | BGB | 4/23/2020 |
| | | | CHECKED | | |
| | | | ENG APPR. | | |
| | | SURFACE FINISH: 125 µin. MAX. RA | INTERPRET GEOMETRIC TOLERANCING PER: ASME Y 14.5-2009 | | |
| PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF TLX TECHNOLOGIES. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF TLX TECHNOLOGIES IS PROHIBITED. | | EDGE BREAK: 0.005 MAX. | | | |
| | | MATERIAL | | | |
| | | FINISH | | | |
| | | ALL DIMENSIONS ARE POST PLATE | | | |

TLX Technologies

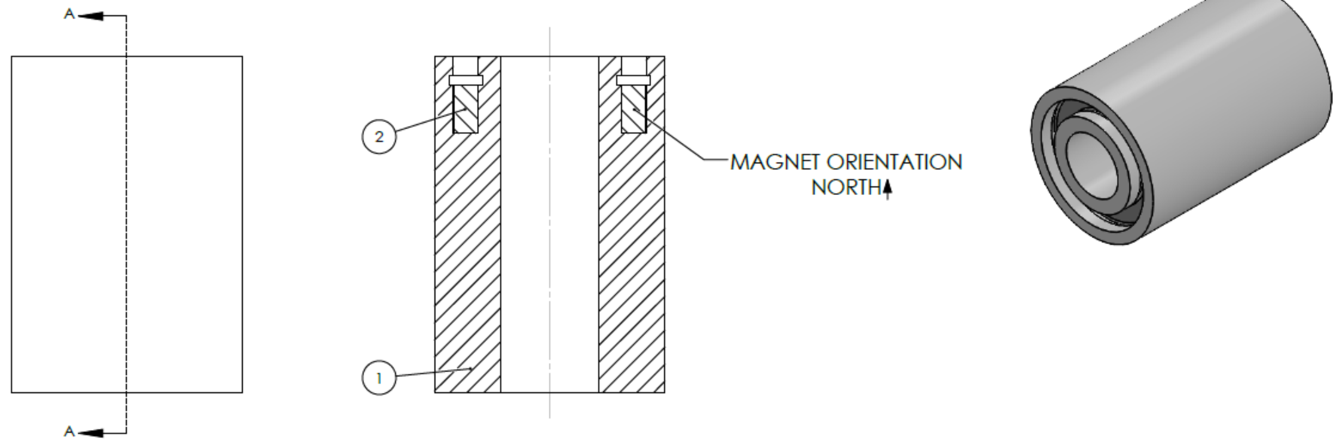
TITLE:
LIQUID LEVEL INDICATOR ASSEMBLY

| SIZE | DWG. NO. | REV |
|---|---|---|
| C | SA1249 | .01 |
| SCALE: 1:8 | DO NOT SCALE DRAWING | SHEET 1 OF 1 |

# Appendix H: LLI External Float Assembly

| ITEM NO. | PART NUMBER | REV | DESCRIPTION | QTY. |
|---|---|---|---|---|
| 1 | P07383 | .01 | EXTERNAL FLOAT | 1 |
| 2 | P07382 | .01 | RING MAGNET | 1 |

| REVISION | | | |
|---|---|---|---|
| EC | REV | DESCRIPTION | DATE |
| | | | |
| | .01 | INITIAL CONCEPT | 11/3/2020 |
| | .02 | REMOVING P07379 & ADDING P07829 | 05/19/2021 |



SECTION A-A

| | | | |
|---|---|---|---|
| UNLESS OTHERWISE SPECIFIED: | DRAWN BY | BGB | |
| DIMENSIONS ARE IN INCHES | DATE DRAWN | 11/3/2020 | |
| TOLERANCES: ANGULAR: 1° TWO PLACE DECIMAL ±.01 THREE PLACE DECIMAL ±.005 | CUSTOMER P/N | REV | |
| SURFACE FINISH: 125 µin MAX. | | | |
| EDGE BREAK: .005 MAX. | INTERPRET GEOMETRIC TOLERANCING PER: ASME Y14.5-2009 | | |
| MATERIAL | | | |
| FINISH | | | |
| ALL DIMENSIONS ARE POST PLATE | | | |

**PROPRIETARY AND CONFIDENTIAL**
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF TLX TECHNOLOGIES. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF TLX TECHNOLOGIES IS PROHIBITED.

TLX Technologies

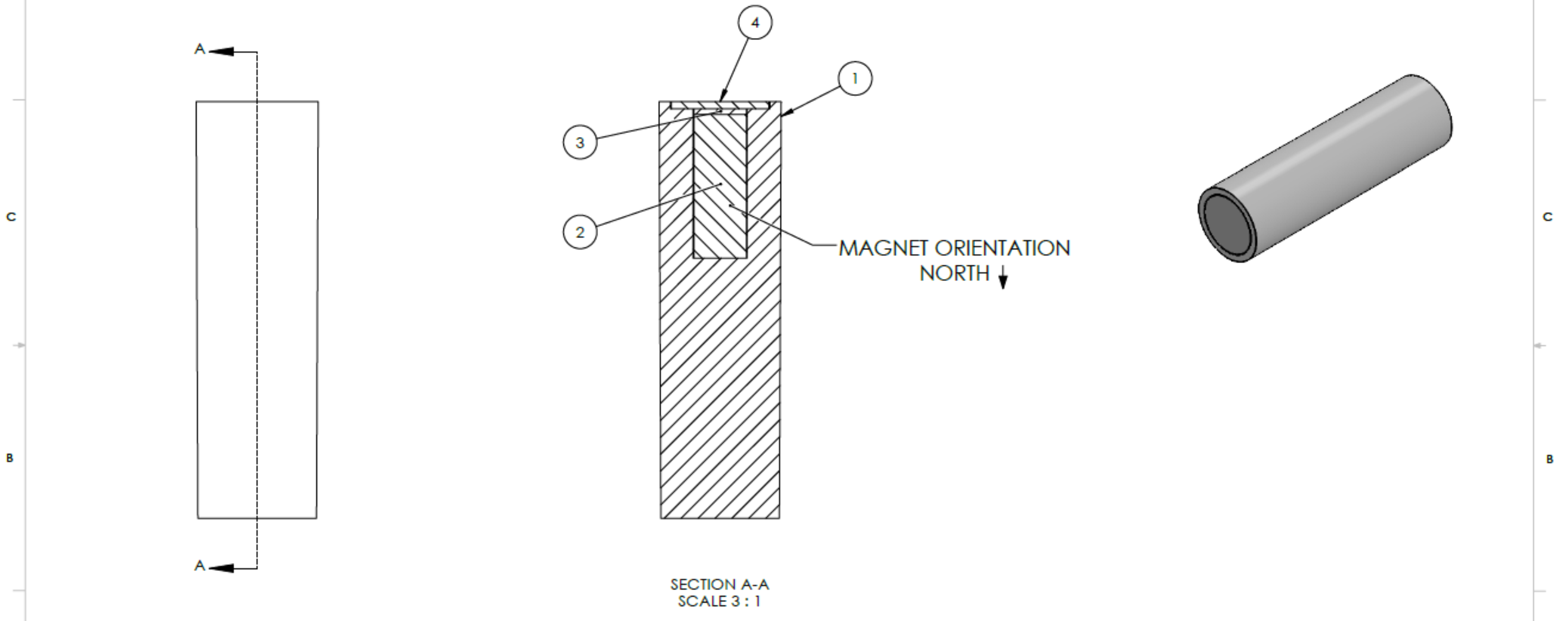
TITLE:
EXTERNAL FLOAT ASSEMBLY

| SIZE | DWG. NO. | REV |
|---|---|---|
| B | SA1307 | .02 |
| SCALE: 2:1 | DO NOT SCALE DRAWING | SHEET 1 OF 1 |

# Appendix I: LLI Internal Float Assembly

| ITEM NO. | PART NUMBER | REV | DESCRIPTION | QTY. |
|---|---|---|---|---|
| 1 | P07381 | .03 | INTERNAL FLOAT | 1 |
| 2 | P07380 | .01 | MAGNET | 1 |
| 3 | P07821 | .01 | ADHESIVE | 1 |
| 4 | P07378 | .03 | WASHER | 1 |

| REVISION | | | |
|---|---|---|---|
| EC | REV | DESCRIPTION | DATE |
| | .01 | INITIAL CONCEPT | 11/2/2020 |
| | .02 | P07821 WAS P07389 | 5/19/2021 |
| | .03 | P07381 & P07378 TO REV .03 | 06/03/2021 |



NOTES:

1. MAGNET IS PRESS FIT INTO FLOAT

**PROPRIETARY AND CONFIDENTIAL**
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF TLX TECHNOLOGIES. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF TLX TECHNOLOGIES IS PROHIBITED.

| | | |
|---|---|---|
| UNLESS OTHERWISE SPECIFIED: | DRAWN BY | BGB |
| DIMENSIONS ARE IN INCHES | DATE DRAWN | 11/2/2020 |
| TOLERANCES: ANGULAR: 1° TWO PLACE DECIMAL ±.01 THREE PLACE DECIMAL ±.005 | CUSTOMER P/N | REV |
| SURFACE FINISH: 125 µin MAX. | | |
| EDGE BREAK: .005 MAX. | INTERPRET GEOMETRIC TOLERANCING PER: ASME Y14.5-2009 | |
| MATERIAL | | |
| FINISH | | |
| ALL DIMENSIONS ARE POST PLATE | | |

TLX Technologies

TITLE: INTERNAL FLOAT ASSEMBLY

| SIZE | DWG. NO. | REV |
|---|---|---|
| B | SA1306 | .03 |
| SCALE: 2:1 | DO NOT SCALE DRAWING | SHEET 1 OF 1 |

## Appendix J:  Serial_Comms_Rev_5

```
#include<EEPROM.h>

//User menu variables

int Agent = 0; //stored in EEPROM address 0

int Units = 0; //stored in EEPROM address 4

float radius = 0; //stored in EEPROM address 8

float thickness = 0; //stored in EEPROM address 12

float height = 0; //stored in EEPROM address 16

int Display = 0; //stored in EEPROM address 20

int Interval = 0; //stored in EEPROM address 24

int Recording = 0; //stored in EEPROM address 28

int Year = 0;     //stored in EEPROM address 32

int Month = 0;    //stored in EEPROM address 33

int Day = 0;      //stored in EEPROM address 34

double Time = 0;  //stored in EEPROM address 35

int eeAddress = 0;

//User menu function

void devicemenu()

{

  //User menu

  Serial.println("1. Agent Type");

  Serial.println("2. Measurement Units");

  Serial.println("3. Storage Cylinder Dimensions");
  Serial.println("4. Display Data");
  Serial.println("5. Measurement Interval");
  Serial.println("6. Data Recording");
  Serial.println("7. Set Date");
  Serial.println("8. Set Time");
  Serial.println("Enter Menu Number? ");
  while (Serial.available() == 0) //Waiting for user input
  {
  }
  //Reading user input from serial stream
  int menuChoice = Serial.parseInt(SKIP_WHITESPACE,'\r');
  switch (menuChoice) //Displays submenu option based on user selected menu option
  {
    case 1:
    {
      // Agent Type Submenu
      bailoutA:
      Serial.println("Agent Type:");
      Serial.println("1. FM200");
      Serial.println("2. Novec 1230");
      Serial.println("3. Water");
      while (Serial.available() == 0);  //Waiting for user input
      {
   }
   //Reading user input from serial stream
   Agent = Serial.parseInt(SKIP_WHITESPACE,'\r');
   //Verifies user input is a valid menu option.
   //If it is, the variable is set.
   //If the menu option is invalid error message is sent to
   //serial terminal and submenu is redisplayed.
   if (Agent >= 1 && Agent <= 3)
   {
     Serial.print("Agent Type: ");
     Serial.println(Agent);
     eeAddress = 0;
     EEPROM.put(eeAddress,Agent);
     break;
   }
   else
   {
     Serial.println("Error setting must be 1, 2 or 3");
     Agent = 0;
     goto bailoutA;
   }
  }
  case 2:
```

```
{
 // Measurement Units Submenu
 bailoutB:
 Serial.println("Measurement Units:");
 Serial.println("1. Metric");
 Serial.println("2. US Imperial");
 while (Serial.available() == 0);  //Waiting for user input
 {
 }
 //Reading user input from serial stream
 Units = Serial.parseInt(SKIP_WHITESPACE,'\r');
 //Verifies user input is a valid menu option.
 //If it is, the variable is set.
 //If the menu option is invalid error message is sent
 //to serial terminal and submenu is redisplayed.
 if (Units >= 1 && Units <= 2)
 {
   Serial.print("Units Set To: ");
   Serial.println(Units);
   eeAddress = 4;
   EEPROM.put(eeAddress,Units);
   break;
 }
```

```
  else
  {
    Serial.println("Error setting must be 1 or 2");
    Units = 0;
    goto bailoutB;
  }
}
case 3:
{
  // Cylinder Dimensions Submenu
  // Verifies that the measurement units have been set.
  // If they have not, function redirects user to the Measurement Units submenu
  if (Units == 1)
  {
    Serial.println("Cylinder Radius (mm): RR");
    while (Serial.available() == 0);  //Waiting for user input
    {
    }
    //Reading user input from serial stream
    radius = Serial.parseFloat(SKIP_WHITESPACE,'\r');
    Serial.print("Cylinder Radius is: ");
    Serial.println(radius);
    eeAddress = 8;
    EEPROM.put(eeAddress,radius);
    Serial.println("Cylinder Thickness (mm): TT");
    while (Serial.available() == 0);  //Waiting for user input
    {
    }
    //Reading user input from serial stream
    thickness = Serial.parseFloat(SKIP_WHITESPACE,'\r');
    Serial.print("Cylinder Thickness is: ");
    Serial.println(thickness);
    eeAddress = 12;
    EEPROM.put(eeAddress,thickness);
    Serial.println("Cylinder Height (mm): HH");
    while (Serial.available() == 0);  //Waiting for user input
    {
    }
    //Reading user input from serial stream
    height = Serial.parseFloat(SKIP_WHITESPACE,'\r');
    Serial.print("Cylinder Height is: ");
    Serial.println(height);
    eeAddress = 16;
    EEPROM.put(eeAddress,height);
    break;
  }
```

```
else if (Units == 2)
{
  Serial.println("Cylinder Radius (in): RR.R");
  while (Serial.available() == 0);  //Waiting for user input
  {
  }
  //Reading user input from serial stream
  radius = Serial.parseFloat(SKIP_WHITESPACE,'\r');
  Serial.print("Cylinder Radius is: ");
  Serial.println(radius);
  radius = radius * 25.4; //convert inch to mm
  eeAddress = 8;
  EEPROM.put(eeAddress,radius);

  Serial.println("Cylinder Thickness (in): TT.T");
  while (Serial.available() == 0);  //Waiting for user input
  {
  }
  //Reading user input from serial stream
  thickness = Serial.parseFloat(SKIP_WHITESPACE,'\r');
  Serial.print("Cylinder Thickness is: ");
  Serial.println(thickness);
  thickness = thickness * 25.4; //convert inch to mm
```

```
    eeAddress = 12;
    EEPROM.put(eeAddress,thickness);
    Serial.println("Cylinder Height (in): HH.H");
    while (Serial.available() == 0);  //Waiting for user input
    {
    }
    //Reading user input from serial stream
    height = Serial.parseFloat(SKIP_WHITESPACE,'\r');
    Serial.print("Cylinder Height is: ");
    Serial.println(height);
    height = height * 25.4; //convert inch to mm
    eeAddress = 16;
    EEPROM.put(eeAddress,height);
    break;
  }
  else
  {
    Serial.println("Measurement Units are not set");
    goto bailoutB;
  }
}
case 4:
  {
```

```
//  Display Refresh Rate
bailoutC:
Serial.println("Display Refresh Rate:");
Serial.println("1. Continuous");
Serial.println("2. Single");
while (Serial.available() == 0);  //Waiting for user input
{
}
//Reading user input from serial stream
Display = Serial.parseInt(SKIP_WHITESPACE,'\r');
//Verifies user input is a valid menu option.
//If it is, the variable is set.
//If the menu option is invalid error message is sent to serial
//terminal and submenu is redisplayed.
if (Display >= 1 && Display <= 2)
{
  Serial.print("Display Refresh Rate Set To: ");
  Serial.println(Display);
  eeAddress = 20;
  EEPROM.put(eeAddress,Display);
  break;
}
else
```

```
   {
     Serial.println("Error setting must be 1 or 2");
     Display = 0;
     goto bailoutC;
   }
 }
case 5:
{
   //  Measurement Interval
   bailoutD:
   Serial.println("Measurement Interval:");
   Serial.println("1. 1s");
   Serial.println("2. 15min");
   Serial.println("3. 30min");
   Serial.println("4. 1hr");
   Serial.println("5. 3hrs");
   Serial.println("6. 6hrs");
   Serial.println("7. 12hrs");
   Serial.println("8. 24hrs");
   while (Serial.available() == 0);  //Waiting for user input
   {
   }
   //Reading user input from serial stream
   Interval = Serial.parseInt(SKIP_WHITESPACE,'\r');
   //Verifies user input is a valid menu option.
   //If it is, the variable is set.
   //If the menu option is invalid error message is sent to
   //serial terminal and submenu is redisplayed.
   if (Interval >= 1 && Interval <= 8)
   {
     Serial.print("Measurement Interval Set To: ");
     Serial.println(Interval);
     eeAddress = 24;
     EEPROM.put(eeAddress,Interval);
     break;
   }
   else
   {
     Serial.println("Error setting must be 1 thru 8");
     Interval = 0;
     goto bailoutD;
   }
 }
 case 6:
  {
   //  Data Recording
```

```
bailoutE:
Serial.println("Data Recording:");
Serial.println("1. Off");
Serial.println("2. On");
while (Serial.available() == 0);  //Waiting for user input
{
}
//Reading user input from serial stream
Recording = Serial.parseInt(SKIP_WHITESPACE,'\r');
//Verifies user input is a valid menu option.
//If it is, variable is set.
//If the menu option is invalid error message is sent to
//serial terminal and submenu is redisplayed.
if (Recording >= 1 && Recording <= 2)
{
  Serial.print("Data Recording Is: ");
  Serial.println(Recording);
  eeAddress = 28;
  EEPROM.put(eeAddress, Recording);
  break;
}
else
{
      Serial.println("Error setting must be 1 or 2");
      Recording = 0;
      goto bailoutE;
    }
  }
case 7:
{
   //  Set Date
   bailoutF:
   Serial.println("Enter Year: yyyy");
   while (Serial.available() == 0);  //Waiting for user input
   {
   }
   //Reading user input from serial stream
   Year = Serial.parseInt(SKIP_WHITESPACE,'\r');
   eeAddress = 32;
   EEPROM.put(eeAddress, Year);
   delay(10);
   Serial.println("Enter Month: mm");
   while (Serial.available() == 0);  //Waiting for user input
   {
   }
   //Reading user input from serial stream
```

```
Month = Serial.parseInt(SKIP_WHITESPACE,'\r');
eeAddress = 34;
EEPROM.put(eeAddress, Month);
Serial.println("Enter Day: dd");
while (Serial.available() == 0);  //Waiting for user input
{
}
//Reading user input from serial stream
Day = Serial.parseInt(SKIP_WHITESPACE,'\r');
eeAddress = 36;
EEPROM.put(eeAddress, Day);
bailoutG:
Serial.print(F("You entered: "));
Serial.print(Year);
Serial.print(F(" : "));
Serial.print(Month);
Serial.print(F(" : "));
Serial.println(Day);
Serial.println(F("Is that correct? Yes(1) No(2)"));
while (Serial.available() == 0);  //Waiting for user input
{
}
//Reading user input from serial stream
```

```
   int answer = Serial.parseInt(SKIP_WHITESPACE,'\r');
   if (answer == 1)
   {
     break;
   }
   else if (answer == 2)
   {
     goto bailoutF;
   }
   else
   {
     Serial.println("Error entry must be 1 or 2");
     answer = 0;
     goto bailoutG;
   }
}
 case 8:
 {
   //  Set Time
   Serial.println("Enter Time (24hr): hhmmss");
   while (Serial.available() == 0);  //Waiting for user input
   {
   }
```

```
        //Reading user input from serial stream
        Time = Serial.parseFloat(SKIP_WHITESPACE,'\r');
        Serial.print("Time Set To: ");
        Serial.println(Time, 0);
        eeAddress = 38;
        EEPROM.put(eeAddress, Time);
        break;
      }
     default:
     {
        Serial.println("Please choose a valid selection");
        break;
     }
  }
}
void setup()
{
  Serial.begin(9600);
  Serial.println(F("Clear EEPROM Yes(1) or No(2)"));
  while (Serial.available() == 0) //Waiting for user input
  {
  }
  int Clear = Serial.parseInt(SKIP_WHITESPACE,'\r');
```

```
switch (Clear)  //Clear the EEPROM
{
 case 1:
 {
   for (int i = 0 ; i < EEPROM.length() ; i++)
   {
     EEPROM.write(i, 0);
   }
   Serial.println(F("Clear Complete"));
   break;
 }
 case 2:
 {
   Serial.println(F("Clear Skipped"));
   break;
 }
}
eeAddress = 0;
EEPROM.get(eeAddress,Agent);
eeAddress = 4;
EEPROM.get(eeAddress,Units);
eeAddress = 8;
EEPROM.get(eeAddress,radius);
```

```
  eeAddress = 12;
  EEPROM.get(eeAddress,thickness);
  eeAddress = 16;
  EEPROM.get(eeAddress,height);
  eeAddress = 20;
  EEPROM.get(eeAddress,Display);
  eeAddress = 24;
  EEPROM.get(eeAddress,Interval);
  eeAddress = 28;
  EEPROM.get(eeAddress,Recording);
  eeAddress = 32;
  EEPROM.get(eeAddress,Year);
  eeAddress = 34;
  EEPROM.get(eeAddress,Month);
  eeAddress = 36;
  EEPROM.get(eeAddress,Day);
  eeAddress = 38;
  EEPROM.get(eeAddress,Time);
}
void loop()
{
  //Verifying that device settings have been entered.  If settings are not set, run device
menu function
```

```
  while (Agent == 0 || Units == 0 || radius == 0 || thickness == 0 || height == 0 || Display
== 0  || Interval == 0 || Recording == 0 || Year == 0 || Month == 0 || Day == 0 || Time ==
0)
  {
    devicemenu();
  }
  Serial.println("Device setup complete");
  Serial.println("Run Program");
  int Af = 0;
  eeAddress = 0;
  EEPROM.get(eeAddress,Af);
  Serial.println(Af);
  int Uf = 0;
  eeAddress = 4;
  EEPROM.get(eeAddress,Uf);
  Serial.println(Uf);
  float rf = 0;
  eeAddress = 8;
  EEPROM.get(eeAddress,rf);
  Serial.println(rf);
  float tf = 0;
  eeAddress = 12;
  EEPROM.get(eeAddress,tf);
```

```
Serial.println(tf);

float hf = 0;

eeAddress = 16;

EEPROM.get(eeAddress,hf);

Serial.println(hf);

int Df = 0;

eeAddress = 20;

EEPROM.get(eeAddress,Df);

Serial.println(Df);

int If = 0;

eeAddress = 24;

EEPROM.get(eeAddress,If);

Serial.println(If);

int Rf = 0;

eeAddress = 28;

EEPROM.get(eeAddress,Rf);

Serial.println(Rf);

int Yf = 0;

eeAddress = 32;

EEPROM.get(eeAddress,Yf);

Serial.println(Yf);

int Mf = 0;

eeAddress = 34;
```

```
EEPROM.get(eeAddress,Mf);
Serial.println(Mf);
int DDf = 0;
eeAddress = 36;
EEPROM.get(eeAddress,DDf);
Serial.println(DDf);
double Tf = 0;
eeAddress = 38;
EEPROM.get(eeAddress, Tf);
Serial.println(Tf);
Serial.println(Agent);
Serial.println(Units);
Serial.println(radius);
Serial.println(thickness);
Serial.println(height);
Serial.println(Display);
Serial.println(Interval);
Serial.println(Recording);
Serial.println(Year);
Serial.println(Month);
Serial.println(Day);
Serial.println(Time);
delay(20000);
```

```
}
```

## Appendix K: LLI Firmware with User Interface Rev 8

```
/*
  Reading distance from the laser-based ST Microelectronics VL53L1X sensor and
temperature from the TI LM35 temperature sensor.
  By: Ian Stumpe
  MSOE MSE Capstone Project
  Date: May 5, 2023
  License: This code is public domain, but you buy me a beer if you use this and we meet
someday (Beerware license).
*/
#include <SparkFun_VL53L1X.h> //SparkFun ST VL53L1X shelied Library:
http://librarymanager/All#SparkFun_VL53L1X
#include <Wire.h> //Allows Arduino boards to communicate with I2C/TWI Devices.
#include <Adafruit_RGBLCDShield.h>  //Adafruit RGB LCD Shield Library:
https://learn.adafruit.com/rgb-lcd-shield
#include <InterpolationLib.h> //This library contains the methods that interpolate agent
density based on temperature
#include<EEPROM.h>
#define BacklightOFF 0x0   //Turns the LCD backlight off
#define WHITE 0x7 //Define the backlight color of the Adafruit RGB LCD Display
#define RED 0x1   //Define the backlight color of the Adafruit RGB LCD Display
#define interruptPin 2 //Pin we are going to use to trigger interrupt to show data on the
Display
```

```
// The Adafruit RGBLCD Shield uses the I2C SCL and SDA pins.
Adafruit_RGBLCDShield lcd = Adafruit_RGBLCDShield();
SFEVL53L1X distanceSensor; //Initializes ST VL53L1X distance sensor.
//Global Variables
int tempPin = A0; //Analog pin connected to TI temperature sensor
//float olddistance = 100; //Variable for tracking old distance measurement
int Agent = 0; //User Menu variable stored in EEPROM address 0
int Units = 0; //User Menu variable stored in EEPROM address 4
float radius = 0; //User Menu variable stored in EEPROM address 8
float thickness = 0; //User Menu variable stored in EEPROM address 12
float height = 0; //User Menu variable stored in EEPROM address 16
int Display = 0; //User Menu variable stored in EEPROM address 20
int Interval = 0; //User Menu variable stored in EEPROM address 24
int Recording = 0; //User Menu variable stored in EEPROM address 28
int Year = 0;     //stored in EEPROM address 32
int Month = 0;    //stored in EEPROM address 34
int Day = 0;      //stored in EEPROM address 36
double Time = 0;  //stored in EEPROM address 38
int eeAddress = 0; //EEPROM address location variable, EEPROM address 44 and up for
datalogger
int EEaddress = 44; //EEPROM address location variable used for writing data to datalog
float temperature = 0;  //holds temperature data
float mass = 0; //holds mass data
float oldMass = 0;  //holds the first mass data point after power on for 5% limit check
float oldmass = 0;  //used for data recording EEPROM addressing
volatile long int seconds = 0;
long int sampleRate = 0;
int record = 0;
void setup(void)
{
  Wire.begin(); //Initialize I2C BUS
  lcd.begin(16, 2); //Initialize LCD Display, sets LCD's number of columns and rows.
  lcd.setBacklight(WHITE); //Set LCD backlight color
  pinMode(interruptPin, INPUT_PULLUP);
  Serial.begin(9600); //Initialize Serial BUS for troubleshooting.
  Serial.println(F("Clear EEPROM Yes(1) or No(2)"));
  while (Serial.available() == 0) //Waiting for user input
  {
  }
  int Clear = Serial.parseInt(SKIP_WHITESPACE,'\r');
  switch (Clear)  //Clear the EEPROM
  {
    case 1:
    {
      for (int i = 0 ; i < EEPROM.length() ; i++)
      {
```

```
      EEPROM.write(i, 0);
     }
     Serial.println(F("Clear Complete"));
     break;
   }
   case 2:
   {
     Serial.println(F("Clear Skipped"));
     break;
   }
 }
//Check EEPROM for user setting
eeAddress = 0;
EEPROM.get(eeAddress,Agent);
eeAddress = 4;
EEPROM.get(eeAddress,Units);
eeAddress = 8;
EEPROM.get(eeAddress,radius);
eeAddress = 12;
EEPROM.get(eeAddress,thickness);
eeAddress = 16;
EEPROM.get(eeAddress,height);
eeAddress = 20;
```

```
  EEPROM.get(eeAddress,Display);
  eeAddress = 24;
  EEPROM.get(eeAddress,Interval);
  eeAddress = 28;
  EEPROM.get(eeAddress,Recording);
  eeAddress = 32;
  EEPROM.get(eeAddress,Year);
  eeAddress = 34;
  EEPROM.get(eeAddress,Month);
  eeAddress = 36;
  EEPROM.get(eeAddress,Day);
  eeAddress = 38;
  EEPROM.get(eeAddress,Time);
  dataRecording();  //setting up data recording based on user setting
  //Verifies ST VL53L1 sensor is wired correctly. Begin returns 0 on a good init
  if (distanceSensor.begin() != 0)
  {
    Serial.println(F("Sensor failed to begin. Please check wiring. Freezing..."));
    while (1);
  }
  Serial.println(F("Sensor online!"));
  distanceSensor.setROI(4, 4, 196); //Sets the VL53L1's Range of Interest (receiver grid
size)
```

```
   //Sets the VL53L1 timing budget which is the amount of time (ms) over which a
measurement is taken
  distanceSensor.setTimingBudgetInMs(200);
   //Sets the VL53L1's distance measurement mode Short = 1.3m Long = 4m
  distanceSensor.setDistanceModeShort();
}
void loop(void)
{
   //Verifing that device settings have been entered.  If settings are not set run device
menu function
  while (Agent == 0 || Units == 0 || radius == 0 || thickness == 0 || height == 0 || Display
== 0 ||
         Interval == 0 || Recording == 0 || Year == 0 || Month == 0 || Day == 0 || Time == 0)
  {
    devicemenu();
  }
  float distance = getdistance();
  float volume = getvolume(distance, radius, thickness, height);
  temperature = gettemp();
  mass = getmass(volume);
  displayRefresh();
  checkLimits();
  recordData();
```

```
}
void devicemenu()
{
  //User menu
  Serial.println(F("1. Agent Type"));
  Serial.println(F("2. Measurement Units"));
  Serial.println(F("3. Storage Cylinder Dimensions"));
  Serial.println(F("4. Display Data"));
  Serial.println(F("5. Measurement Interval"));
  Serial.println(F("6. Data Recording"));
  Serial.println(F("7. Set Date"));
  Serial.println(F("8. Set Time"));
  Serial.println(F("Enter Menu Number? "));
  while (Serial.available() == 0) //Waiting for user input
  {
  }
  int menuChoice = 0;
  menuChoice = Serial.parseInt(SKIP_WHITESPACE,'\r');   //Reading user input from
serial stream
  switch (menuChoice) //Displays submenu option based on user selected menu option
  {
    case 1:
    {
```

```
// Agent Type Submenu
bailoutA:
Serial.println(F("Agent Type:"));
Serial.println(F("1. FM200"));
Serial.println(F("2. Novec 1230"));
Serial.println(F("3. Water"));
while (Serial.available() == 0);  //Waiting for user input
{
}
Agent = Serial.parseInt(SKIP_WHITESPACE,'\r');  //Reading user input from serial
stream
//Verifies user input is a valid menu option.
//If it is, the variable is set.
//If the menu option is invalid error message is sent to serial terminal and submenu is
redisplayed.
if (Agent >= 1 && Agent <= 3)
{
  Serial.print(F("Agent Type: "));
  Serial.println(Agent);
  eeAddress = 0;
  EEPROM.put(eeAddress,Agent);
  break;
}
```

```
      else
      {
        Serial.println("Error setting must be 1, 2 or 3");
        Agent = 0;
        goto bailoutA;
      }
    }
    case 2:
    {
      // Measurement Units Submenu
      bailoutB:
      Serial.println(F("Measurement Units:"));
      Serial.println(F("1. Metric"));
      Serial.println(F("2. US Imperial"));
      while (Serial.available() == 0);  //Waiting for user input
      {
      }
      Units = Serial.parseInt(SKIP_WHITESPACE,'\r');        //Reading user input from
serial stream
      //Verifies user input is a valid menu option.
      //If it is the variable is set.
      //If the menu option is invalid error message is sent to serial terminal and submenu is
redisplayed.
```

```
  if (Units >= 1 && Units <= 2)
  {
    Serial.print(F("Units Set To: "));
    Serial.println(Units);
    eeAddress = 4;
    EEPROM.put(eeAddress,Units);
    break;
  }
  else
  {
    Serial.println(F("Error setting must be 1 or 2"));
    Units = 0;
    goto bailoutB;
  }
}
case 3:
{
  // Cylinder Dimensions Submenu
  // Verifies that the measurement units have been set.
  // If they have not, function redirects user to the Measurement Units submenu
  if (Units == 1)
  {
    Serial.println(F("Cylinder Radius (mm): RR"));
```

```
        while (Serial.available() == 0);  //Waiting for user input
        {
        }
        radius = Serial.parseFloat(SKIP_WHITESPACE,'\r');     //Reading user input from
serial stream
        Serial.print(F("Cylinder Radius is: "));
        Serial.println(radius);
        eeAddress = 8;
        EEPROM.put(eeAddress,radius);
        Serial.println(F("Cylinder Thickness (mm): TT"));
        while (Serial.available() == 0);  //Waiting for user input
        {
        }
        thickness = Serial.parseFloat(SKIP_WHITESPACE,'\r');  //Reading user input from
serial stream
        Serial.print(F("Cylinder Thickness is: "));
        Serial.println(thickness);
        eeAddress = 12;
        EEPROM.put(eeAddress,thickness);
        Serial.println(F("Cylinder Height (mm): HH"));
        while (Serial.available() == 0);  //Waiting for user input
        {
        }
```

```
    height = Serial.parseFloat(SKIP_WHITESPACE,'\r');     //Reading user input from
serial stream
    Serial.print(F("Cylinder Height is: "));
    Serial.println(height);
    eeAddress = 16;
    EEPROM.put(eeAddress,height);
    break;
   }
   else if (Units == 2)
   {
    Serial.println(F("Cylinder Radius (in): RR.R"));
    while (Serial.available() == 0);  //Waiting for user input
    {
    }
    radius = Serial.parseFloat(SKIP_WHITESPACE,'\r');     //Reading user input from
serial stream
    Serial.print(F("Cylinder Radius is: "));
    Serial.println(radius);
    radius = radius * 25.4; //convert inch to mm
    eeAddress = 8;
    EEPROM.put(eeAddress,radius);
    Serial.println(F("Cylinder Thickness (in): TT.T"));
    while (Serial.available() == 0);  //Waiting for user input
```

```
      {
      }
      thickness = Serial.parseFloat(SKIP_WHITESPACE,'\r');  //Reading user input from
serial stream
      Serial.print(F("Cylinder Thickness is: "));
      Serial.println(thickness);
      thickness = thickness * 25.4; //convert inch to mm
      eeAddress = 12;
      EEPROM.put(eeAddress,thickness);
      Serial.println(F("Cylinder Height (in): HH.H"));
      while (Serial.available() == 0);  //Waiting for user input
      {
      }
      height = Serial.parseFloat(SKIP_WHITESPACE,'\r');     //Reading user input from
serial stream
      Serial.print(F("Cylinder Height is: "));
      Serial.println(height);
      height = height * 25.4; //convert inch to mm
      eeAddress = 16;
      EEPROM.put(eeAddress,height);
      break;
    }
    else
```

```
      {
        Serial.println(F("Measurement Units are not set"));
        goto bailoutB;
      }
    }
    case 4:
    {
      //  Display Refresh Rate
      bailoutC:
      Serial.println(F("Display Refresh Rate:"));
      Serial.println(F("1. Continues"));
      Serial.println(F("2. Single"));
      while (Serial.available() == 0);  //Waiting for user input
      {
      }
      Display  = Serial.parseInt(SKIP_WHITESPACE,'\r');      //Reading user input from
serial stream
      //Verifies user input is a valid menu option.
      //If it is, the variable is set.
      //If the menu option is invalid error message is sent to serial terminal and submenu is
redisplayed.
      if (Display >= 1 && Display <= 2)
      {
```

```
    Serial.print(F("Display Refresh Rate Set To: "));
    Serial.println(Display);
    eeAddress = 20;
    EEPROM.put(eeAddress,Display);
    break;
  }
  else
  {
    Serial.println(F("Error setting must be 1 or 2"));
    Display = 0;
    goto bailoutC;
  }
}
 case 5:
 {
   // Measurement Interval
  bailoutD:
  Serial.println(F("Measurement Interval:"));
  Serial.println(F("1. 60s"));
  Serial.println(F("2. 15min"));
  Serial.println(F("3. 30min"));
  Serial.println(F("4. 1hr"));
  Serial.println(F("5. 3hrs"));
```

```
    Serial.println(F("6. 6hrs"));
    Serial.println(F("7. 12hrs"));
    Serial.println(F("8. 24hrs"));
    while (Serial.available() == 0);  //Waiting for user input
    {
    }
    Interval = Serial.parseInt(SKIP_WHITESPACE,'\r');   //Reading user input from
serial stream
    //Verifies user input is a valid menu option.
    //If it is, the variable is set.
    //If the menu option is invalid error message is sent to serial terminal and submenu is
redisplayed.
    if (Interval >= 1 && Interval <= 8)
    {
      Serial.print(F("Measurement Interval Set To: "));
      Serial.println(Interval);
      eeAddress = 24;
      EEPROM.put(eeAddress,Interval);
      break;
    }
    else
    {
      Serial.println(F("Error setting must be 1 thru 8"));
```

```
        Interval = 0;
        goto bailoutD;
      }
    }
    case 6:
    {
     //  Data Recording
     bailoutE:
     Serial.println(F("Data Recording:"));
     Serial.println(F("1. Off"));
     Serial.println(F("2. On"));
     while (Serial.available() == 0);  //Waiting for user input
     {
     }
     Recording = Serial.parseInt(SKIP_WHITESPACE,'\r');    //Reading user input from
serial stream
     //Verifies user input is a valid menu option.
     //If it is, variable is set.
     //If the menu option is invalid error message is sent to serial terminal and submenu is
redisplayed.
     if (Recording >= 1 && Recording <= 2)
     {
       Serial.print(F("Data Recording Is: "));
```

```
  Serial.println(Recording);
  eeAddress = 28;
  EEPROM.put(eeAddress, Recording);
  break;
 }
 else
 {
  Serial.println(F("Error setting must be 1 or 2"));
  Recording = 0;
  goto bailoutE;
 }
}
case 7:
{
  //  Set Date
  bailoutF:
  Serial.println(F("Enter Year: yyyy"));
  while (Serial.available() == 0);  //Waiting for user input
  {
  }
  Year = Serial.parseInt(SKIP_WHITESPACE,'\r');       //Reading user input from
serial stream
  eeAddress = 32;
```

```
        EEPROM.put(eeAddress, Year);
        delay(10);
        Serial.println(F("Enter Month: mm"));
        while (Serial.available() == 0);  //Waiting for user input
        {
        }
        Month = Serial.parseInt(SKIP_WHITESPACE,'\r');       //Reading user input from
serial stream
        eeAddress = 34;
        EEPROM.put(eeAddress, Month);
        Serial.println(F("Enter Day: dd"));
        while (Serial.available() == 0);  //Waiting for user input
        {
        }
        Day = Serial.parseInt(SKIP_WHITESPACE,'\r');       //Reading user input from
serial stream
        eeAddress = 36;
        EEPROM.put(eeAddress, Day);
        bailoutG:
        Serial.print(F("You entered: "));
        Serial.print(Year);
        Serial.print(F(" : "));
        Serial.print(Month);
```

```
    Serial.print(F(" : "));
    Serial.println(Day);
    Serial.println(F("Is that correct? Yes(1) No(2)"));
    while (Serial.available() == 0);  //Waiting for user input
    {
    }
    int answer = Serial.parseInt(SKIP_WHITESPACE,'\r');       //Reading user input
from serial stream
    if (answer == 1)
    {
      break;
    }
    else if (answer == 2)
    {
      goto bailoutF;
    }
    else
    {
      Serial.println(F("Error entry must be 1 or 2"));
      answer = 0;
      goto bailoutG;
    }
  }
```

```
    case 8:
    {
     //  Set Time
     Serial.println(F("Enter Time (24hr): hhmmss"));
     while (Serial.available() == 0);  //Waiting for user input
     {
     }
     Time = Serial.parseFloat(SKIP_WHITESPACE,'\r');       //Reading user input from
serial stream
     Serial.print(F("Time Set To: "));
     Serial.println(Time);
     eeAddress = 38;
     EEPROM.put(eeAddress, Time);
     break;
    }
    default:
    {
     Serial.println(F("Please choose a valid selection"));
     break;
    }
  }
}
float getdistance()
```

```
{
   float distance = 0; //Variable to hold distance measurement data
  distanceSensor.startRanging(); //Write configuration bytes to initiate measurement
  for (int i=0; i < 1000; i++) //Loop to take 10 measurements to get the average
measurement
  {
    while (!distanceSensor.checkForDataReady()) //Checks if a measurement is ready
    {
      delay(1);
    }
    distance = distance + distanceSensor.getDistance(); //Get the result of the measurement
in mm from the sensor
    //Serial.println(distanceSensor.getSignalPerSpad());
  }
  distanceSensor.clearInterrupt(); //Clears interrupt caused by the .getDistance command.
  distanceSensor.stopRanging(); //Stop taking measurements
  Serial.print(F("SUM (mm): "));
  Serial.print(distance);
  distance = distance/1000; //Calculate the average
  Serial.print(F(" AVG (mm): "));
  Serial.print(distance);
  return distance;
}
```

```
float getvolume(float distance, float radius, float thickness, float height)
{
   //calculate volume in mm
  float volume = 3.14 * (sq(radius - thickness)) * (height - distance); //mm^3
  Serial.print(F(" Volume (mm3): "));
  Serial.print(volume);
  //Water
  return volume;
}
float gettemp()
{
  int val = analogRead(tempPin); //Get temperature from TI sensor
  float mv = (val/1023.0)*5000; //Convert raw ADC data to voltage value
  float cel = mv/10; //Convert voltage to degree Celsius
  Serial.print(F(" Temperature (C): "));
  Serial.println(cel);
  return cel;
}
float getmass(float vol)
{
  switch (Agent)
  {
    case 1: //FM-200
  {
   vol = (vol / 1000000000); //Convert mm^3 to m^3
   Serial.print(F(" Volume (m^3): "));
   Serial.print(vol,5);
   //FM-200 agent density (kg/m^3) values based on temperature (C)
   const int numValues = 21;
   double xValues[24] = { 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80,
                            85, 90, 95, 100}; // Temperature values
   double yValues[24] = { 1486.0, 1467.3, 1448.2, 1428.6, 1408.4, 1387.7, 1366.2,
1344.0, 1320.9, 1296.7, 1271.4, 1244.8, 1216.5, 1186.2, 1153.6, 1117.9, 1078.2,
                           1032.8, 978.6, 907.8, 786.8}; //Density values
   double density = Interpolation::Linear(xValues, yValues, numValues, temperature,
true); //prints agent density
   Serial.print(F(" Density (kg/m^3): "));
   Serial.print(density);
   //calculate mass of agent
   float mass = density * vol;
   Serial.print(F(" Mass (kg): "));
   Serial.println(mass,2);
   if (oldMass == 0)   //saves first mass calculation to variable oldMass
   {
     oldMass = mass;
   }
```

```
      return mass;
    }
    case 2: //Novec 1230
    {
      vol = (vol / 1000000000); //Convert mm^3 to m^3
      Serial.print(F(" Volume (m^3): "));
      Serial.print(vol,2);
      //Novec 1230 agent density (kg/m^3) 10% concentration values based on temperature
(C)
      const int numValues = 21;
      double xValues[24] = { 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80,
                              85, 90, 95, 100}; // Temperature values
      double yValues[24] = { 1.67, 1.64, 1.61, 1.58, 1.55, 1.52, 1.49, 1.46, 1.44, 1.41,
                             1.39, 1.36, 1.34, 1.32, 1.30, 1.28, 1.26, 1.24, 1.22, 1.20, 1.18};
//Density values
      double density = Interpolation::Linear(xValues, yValues, numValues, temperature,
true); //prints agent density
      Serial.print(F(" Density (kg/m^3): "));
      Serial.print(density);
      //calculate mass of agent
      float mass = density * vol;
      Serial.print(F(" Mass (kg): "));
      Serial.println(mass,2);
      if (oldMass == 0) //saves first mass calculation to variable oldMass
      {
        oldMass = mass;
      }
      return mass;
    }
    case 3: //Water
    {
      //Variable to hold converted distance measurement (mm to cm)
      vol = (vol / 1000); //Convert mm^3 to cm^3
      Serial.print(F(" Volume (cm^3): "));
      Serial.print(vol,5);
      //Water density (g/cm^3) values based on temperature (C)
      const int numValues = 15;
      double xValues[15] = { 0.00, 4.00, 4.40, 10.00, 15.60, 21.00, 26.70, 32.20, 37.80,
48.90, 60.00, 71.10, 82.20, 93.30, 100.00}; // Temperature values
      double yValues[15] = { 0.99987, 1.00, 0.99999, 0.99975, 0.99907, 0.99802, 0.99669,
0.9951, 0.99318, 0.9887, 0.98338, 0.97729,
                             0.97056, 0.96333, 0.95865}; //Density values (g/cm^3)
      double density = Interpolation::Linear(xValues, yValues, numValues, temperature,
true); //prints agent density
      //Serial.print(" Density (g/mm^3): ");
      Serial.print(F(" Density (g/cm^3): "));
```

```
      Serial.print(density);
      //calculate mass of agent
      float mass = density * vol;
      mass = (mass / 1000); //Convert g to kg
      Serial.print(F(" Mass (kg): "));
      //Serial.print(F(" Mass (g): "));
      Serial.println(mass,2);
      if (oldMass == 0)   //saves first mass calculation to variable oldMass
      {
        oldMass = mass;
      }
      return mass;
    }
  }
}
void displayRefresh()
{
  switch (Display)
  {
    case 1:
    {
      displayData();
      return;
    }
    case 2:
    {
      attachInterrupt(digitalPinToInterrupt(interruptPin), wakeUp, LOW);
      lcd.clear();
      lcd.setBacklight(BacklightOFF);
      return;
    }
    case 3:
    {
      lcd.clear();
      lcd.setBacklight(WHITE);
      displayData();
      Display = 2;
      delay(2000);
      return;
    }
  }
}
void displayData()
{
  switch (Units)
  {
```

```
case 1: //Display temperature and mass data in SI units on LCD
{
  //print ambient temp
  lcd.setCursor(0, 0); //Set display cursor
  lcd.print("TEMP (C): ");
  lcd.print(temperature,1);
  //print agent mass/weight
  lcd.setCursor(0, 1); //Set display cursor
  lcd.print("Mass (kg): ");
  //lcd.print("Mass (g): ");
  lcd.print(mass,2);
  return;
}
case 2: //Display temperature and mass data in US Imperaial units on LCD
{
  temperature = (temperature * 1.8) + 32; //convert C to F
  mass = (mass * 2.20462); //convert kg to lbs
  //print ambient temp
  lcd.setCursor(0, 0); //Set display cursor
  lcd.print("TEMP (F): ");
  lcd.print(temperature,1);
  //print agent mass/weight
  lcd.setCursor(0, 1); //Set display cursor
```

```
      lcd.print("Mass (lbs): ");
      //lcd.print("Mass (g): ");
      lcd.print(mass,2);
      return;
    }
  }
}
void wakeUp()
{
  Display = 3;
  detachInterrupt(digitalPinToInterrupt(interruptPin));
}
void dataRecording()
{
  switch(Recording)
  {
    case 1: //Data recording off
    {
      return;
    }
    case 2: //Data recording on
    {
      int frequency = 1; // in hz
      //Interrupt Service Routine and timer setup
      noInterrupts();// kill interrupts until everybody is set up
      //We use Timer 1 b/c it's the only 16 bit timer
      TCCR1A = B00000000;//Register A all 0's since we're not toggling any pins
        // TCCR1B clock prescalers
        // 0 0 1 clkI/O /1 (No prescaling)
        // 0 1 0 clkI/O /8 (From prescaler)
        // 0 1 1 clkI/O /64 (From prescaler)
        // 1 0 0 clkI/O /256 (From prescaler)
        // 1 0 1 clkI/O /1024 (From prescaler)
      TCCR1B = B00001100;//bit 3 set for CTC mode, will call interrupt on counter match,
bit 2 set to divide clock by 256, so 16MHz/256=62.5KHz
      TIMSK1 = B00000010;//bit 1 set to call the interrupt on an OCR1A match
      OCR1A  = (unsigned long)((62500UL / frequency) - 1UL);//our clock runs at
62.5kHz, which is 1/62.5kHz = 16us
      recordingRate();
      interrupts();//restart interrupts
      Serial.println(F("Recording Settings On"));
      return;
    }
  }
}
void recordingRate()
```

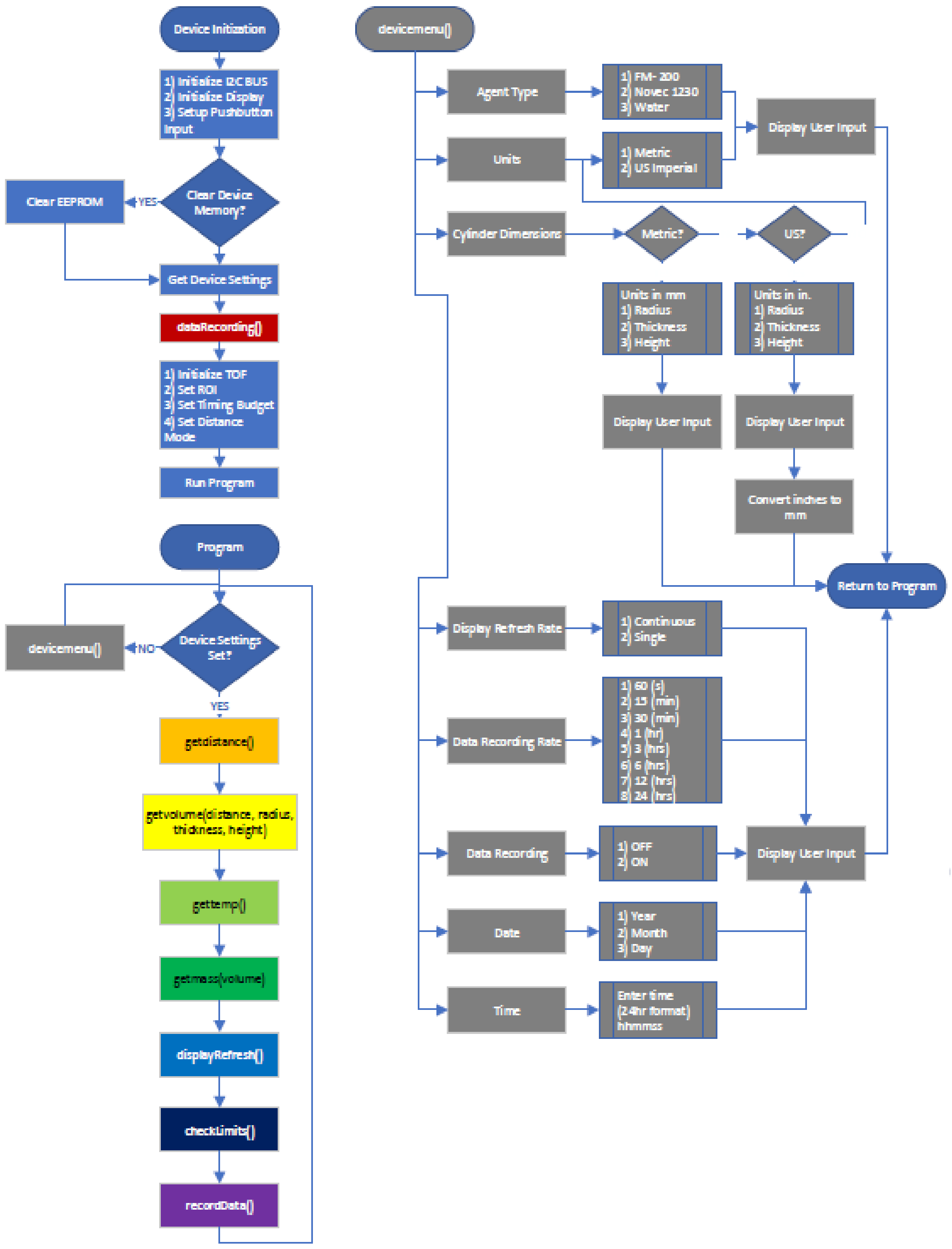
```
{
 switch(Interval)
 {
  case 1: //recording rate equals 60s
  {
   sampleRate = 60;
   return;
  }
  case 2: //recording rate equals 900s (15min)
  {
   sampleRate = 900;
   return;
  }
  case 3: //recording rate equals 1800s (30min)
  {
   sampleRate = 1800;
   return;
  }
  case 4: //recording rate equals 3600s (1hr)
  {
   sampleRate = 3600;
   return;
  }
```

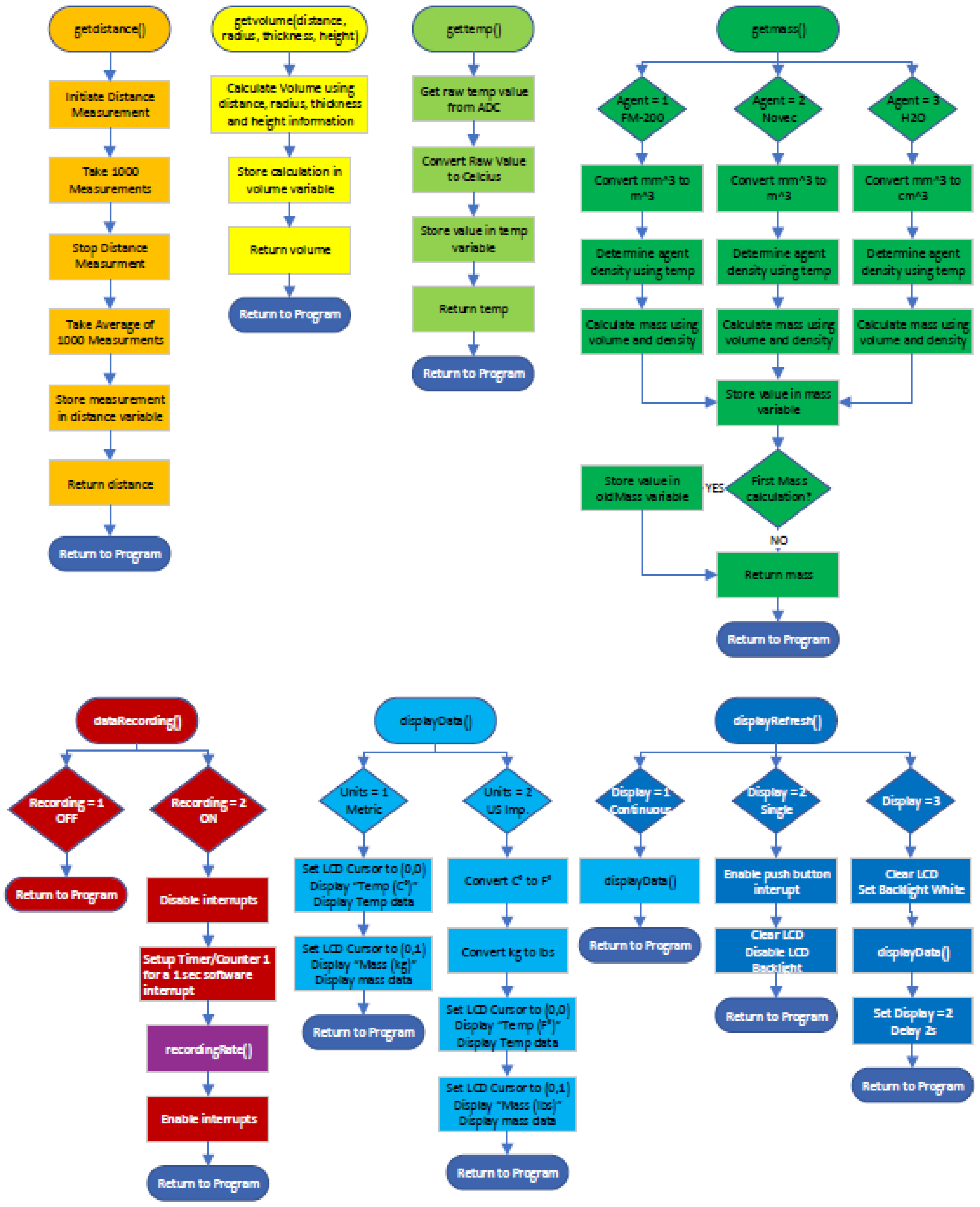
```
    case 5: //recording rate equals 10,800s (3hr)
    {
      sampleRate = 10800;
      return;
    }
    case 6: //recording rate equals 21,600s (6hr)
    {
      sampleRate = 21600;
      return;
    }
    case 7: //recording rate equals 43,200s (12hr)
    {
      sampleRate = 43200;
      return;
    }
    case 8: //recording rate equals 86,400s (24hr)
    {
      sampleRate = 86400;
      return;
    }
  }
}
void recordData()
```

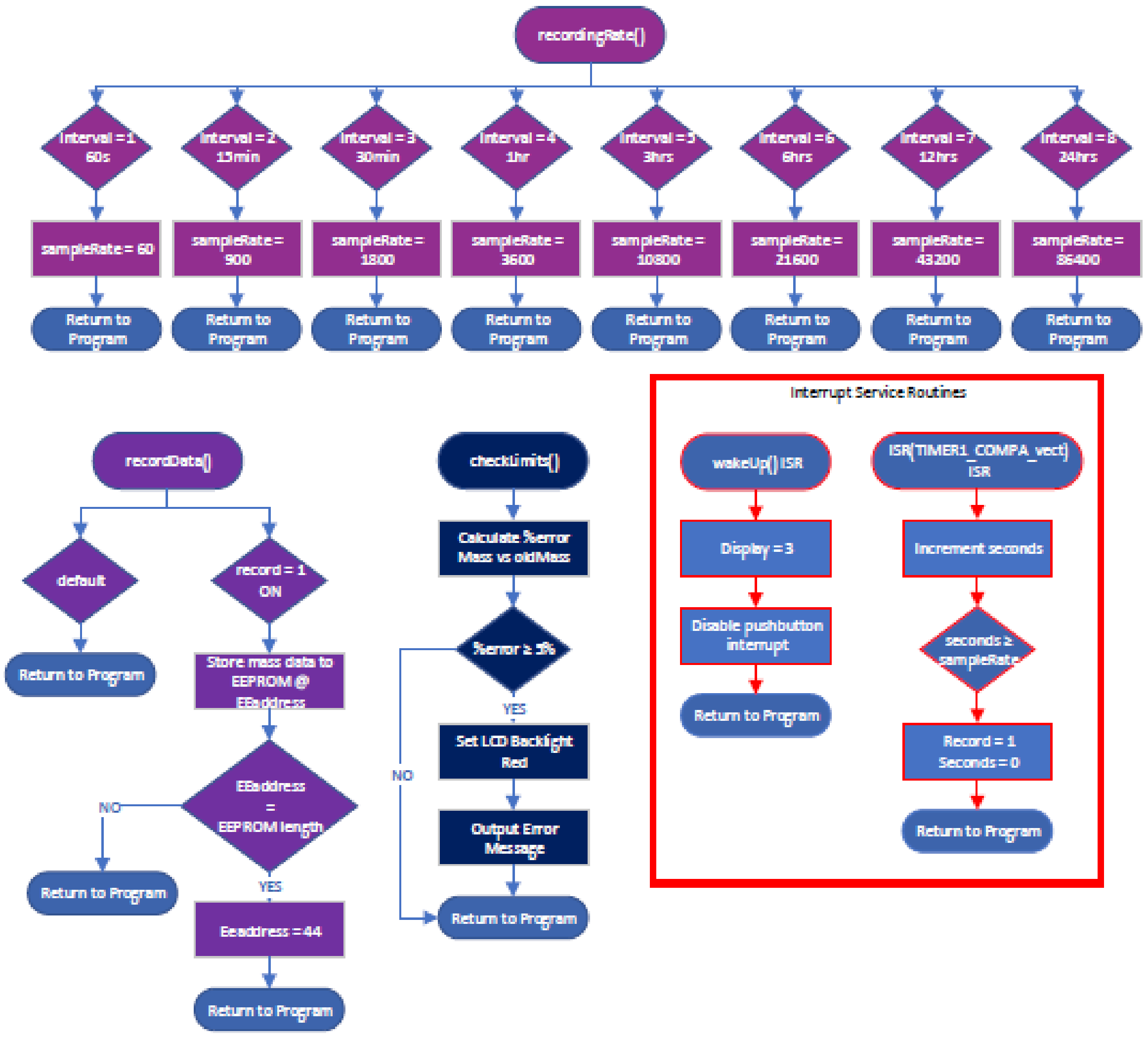
```
{
  switch (record)
  {
    case 1:
    {
      EEPROM.put(EEaddress, mass);
      Serial.println(F("Record Data to EEPROM"));
      oldmass = mass;
      record = 0;
      EEaddress = EEaddress + sizeof(oldmass);
      if (EEaddress == EEPROM.length())
      {
        EEaddress = 44;
      }
      return;
    }
    default:
    {
      return;
    }
  }
}
void checkLimits()
```

```
{
  float percentError = ((mass - oldMass)/(oldMass))*100;
  if (percentError >= 5 || percentError <= -5)
  {
    lcd.setBacklight(RED); //Set LCD backlight color
    Serial.println(F("ERROR Mass has change by more than 5%"));
    return;
  }
  else
  {
    return;
  }
}
ISR(TIMER1_COMPA_vect)
{ //Interrupt Service Routine, Timer/Counter1 Compare Match A
  seconds++;
  if(seconds >= sampleRate) //set to however many seconds you want
  {
    record = 1;             //Tells program to load mass and temperature data to EEPROM
    seconds = 0;            // after 'x' seconds
  }
}
```

## Appendix L: LLI Firmware with User Interface Rev 8 Flowchart




getdistance()
Initiate Distance Measurement
Take 1000 Measurements
Stop Distance Measurment
Take Average of 1000 Measurments
Store measurement in distance variable
Return distance
Return to Program
getvolume(distance, radius, thickness, height)
Calculate Volume using distance, radius, thickness and height information
Store calculation in volume variable
Return volume
Return to Program
gettemp()
Get raw temp value from ADC
Convert Raw Value to Celcius
Store value in temp variable
Return temp
Return to Program
getmass()
Agent = 1 FM-200
Agent = 2 Novec
Agent = 3 H2O
Convert mm^3 to m^3
Convert mm^3 to m^3
Convert mm^3 to cm^3
Determine agent density using temp
Determine agent density using temp
Determine agent density using temp
Calculate mass using volume and density
Calculate mass using volume and density
Calculate mass using volume and density
Store value in mass variable
First Mass calculation?
YES
Store value in oldMass variable
NO
Return mass
Return to Program
dataRecording()
Recording = 1 OFF
Recording = 2 ON
Return to Program
Disable interrupts
Setup Timer/Counter 1 for a 1 sec software interrupt
recordingRate()
Enable interrupts
Return to Program
displayData()
Units = 1 Metric
Units = 2 US Imp.
Set LCD Cursor to (0,0) Display "Temp (C°)" Display Temp data
Set LCD Cursor to (0,1) Display "Mass (kg)" Display mass data
Return to Program
Convert C° to F°
Convert kg to lbs
Set LCD Cursor to (0,0) Display "Temp (F°)" Display Temp data
Set LCD Cursor to (0,1) Display "Mass (lbs)" Display mass data
Return to Program
displayRefresh()
Display = 1 Continuous
Display = 2 Single
Display = 3
displayData()
Return to Program
Enable push button interrupt
Clear LCD Disable LCD Backlight
Return to Program
Clear LCD Set Backlight White
displayData()
Set Display = 2 Delay 2s
Return to Program


recordingRate()
Interval = 1 60s
Interval = 2 15min
Interval = 3 30min
Interval = 4 1hr
Interval = 5 3hrs
Interval = 6 6hrs
Interval = 7 12hrs
Interval = 8 24hrs
sampleRate = 60
sampleRate = 900
sampleRate = 1800
sampleRate = 3600
sampleRate = 10800
sampleRate = 21600
sampleRate = 43200
sampleRate = 86400
Return to Program
Interrupt Service Routines
recordData()
default
record = 1 ON
Return to Program
Store mass data to EEPROM @ EEaddress
EEaddress = EEPROM length
NO
YES
Eeaddress = 44
checkLimits()
Calculate %error Mass vs oldMass
%error ≥ 5%
YES
NO
Set LCD Backlight Red
Output Error Message
wakeUp() ISR
Display = 3
Disable pushbutton interrupt
ISR(TIMER1_COMPA_vect) ISR
Increment seconds
seconds ≥ sampleRate
Record = 1 Seconds = 0

**Engineering**

**Capstone Report Approval Form**

**Master of Science in Engineering – MSE**

**Milwaukee School of Engineering**

This capstone report, titled "Liquid Level Indicator (LLI) Utilizing an Optical Time-of-Flight Sensor for Clean Agent Fire Suppression System Cylinders," submitted by the student Ian Stumpe, has been approved by the following committee:

Faculty Advisor: ____________________ Date: ________________

Dr. Cory Prust, Ph.D.

Faculty Member: ____________________ Date: ________________

Dr. Subha Kumpaty, Ph.D.

Faculty Member: ____________________ Date: ________________

Professor Gary Shimek, M.L.I.S.