University of Nevada, Reno

SkyWarden: Aerial Drone Notification System

A thesis submitted in partial fulfillment
of the requirements for the degree of

Bachelor of Science in Computer Science and Engineering

by

Jia Li, Bryan Kline, Robert Watkins, and Rony Calderon

Dr. Kostas Alexis, Thesis Advisor

May, 2018

UNIVERSITY OF NEVADA RENO

THE HONORS PROGRAM

We recommend that the thesis
prepared under our supervision by

JIA LI, BRYAN KLINE, ROBERT WATKINS, and RONY CALDERON

entitled

SkyWarden: Aerial Drone Notification System

be accepted in partial fulfillment of the
requirements for the degree of

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

_____

Kostas Alexis, Ph.D., Thesis Advisor

_____

Tamara Valentine, Ph.D., Director, Honors Program

May, 2018

Abstract

SkyWarden is an alert and notification system for aerial drones which issues real-time alerts to the drone operators in the Autonomous Robots Laboratory at UNR. Drone battery voltage levels, as well as drone proximity to external obstacles, are monitored by way of sensors on-board the drone. The alert and notification system issues visual and audible warnings to the operator whenever thresholds for each monitored attribute are exceeded. The alerts are issued through both hardware a graphical user interface, and the data is streamed over the network through ROS. Additionally, the system allows for up to eight generic alerts to be added by the operator. The objective of this document is to detail the use cases and functional requirements which have been implemented upon completion of the project, to describe the few which have not been implemented, and to provide a summary of the software modules which comprise the various subsystems of ADNS. The full documentation of the three major subsystems of ADNS is also included.

Table of Contents

Abstract i

List of Figures iv

Project Description 1

Significance 2

Functionality Implemented 3

Use Cases 4

Functional Requirements 7

Functionality Not Implemented 10

Software Overview 11

Updated Design 17

Data Structures 18

Updated Hardware Design 19

Drone Hardware Component Descriptions 20

Physical Alert Device Components 24

Physical Alert Device Hardware Component Descriptions 25

Updated User Interface Design: Drone 27

Updated User Interface Design: Physical Alert Device 29

Updated User Interface Design: GUI Application 30

Contributions 33

List of Figures

Figure 1: Context Model for the Aerial Drone Notification System. 18

Figure 2: High-level design diagram of the system and its hardware components, both the hardware on board the drone and the hardware comprising the physical alert device. 20

Figure 3: DJI Matrice 100 Aerial Drone. 21

Figure 4: Teensy 3.2 Microcontroller. 21

Figure 5: VL53L0X Time-of-Flight Sensor. 22

Figure 6: INA219 Current & Voltage Sensor. 23

Figure 7: nRF24L01 RF Wireless Transceiver. 23

Figure 8: High-level design diagram of the components that make up the physical alert device and their interactions. 25

Figure 9: Modularity aspect for the sensor modules is critical since the pin configurations for different manufacturers selling the same type of sensor may vary. 28

Figure 10: Initial output of each sensor's proximity reading to an object. Each value is represented in millimeters since distance precision is an important variable feature. Output format is subject to change. 29

Figure 11: Early box concept featuring seven-segment displays for tracking voltage and current, visual led alerts, speaker for audible alerts, volume control, alert resets, and input/output interfaces. 30

Project Description

Aerial drone systems often lack functionality which provide continuous feedback to the user about drone performance and safety. The aerial drone systems in use in the Autonomous Robots Lab, ARL, at UNR similarly lack the ability to report to researchers' various metrics associated with their experiments such as drone battery voltage and current levels and proximity to obstacles. The team is working with ARL to develop a system which can be easily added to their existing aerial drones which will continuously monitor drone battery levels, proximity readings, as well as an additional, user-defined sensor, and report the readings to the researchers in real time. At present, the drone operators in ARL must interrupt their experiments to manually monitor drone battery levels and there are no additional checks on their odometry data during drone flight. The Aerial Drone Notification System, ADNS, will allow the researchers to optimize their work flow by dispensing with the need to pause experiments and manually meter drone battery levels. The system will also provide an additional check on the spatial orientation and proximity of the drone to improve flight safety. ADNS will also provide the drone operators with an additional line of defense against safety risks by providing a kill switch that will send a signal to immediately ground the drone. While the project is developed specifically for the project sponsors in the ARL, the system could also be used by other research facilities which utilize aerial drones in their work or by manufacturers of consumer drone systems. ADNS is composed of four major subsystems, some hardware based, others entirely software based. The two subsystems which are hardware based include the components on-board the drone, the current, voltage, and proximity sensors, a Teensy microcontroller, and an RF transmitter, as well as the components which make up the physical alert device that issues the alerts, a Raspberry Pi, seven segment displays, LEDs, switches, a speaker, and an RF receiver. The drivers for the hardware components

on the drone are written in ANSI C, while the software controlling the hardware in the physical alert device is written in Python. The subsystems which are entirely software based include ROS node components for communicating with the ARL ROS master node which controls the drone, and a GUI application to be run on the Raspberry Pi which acts as another interface for the user to interact with the system in addition to the physical alert device. These components are developed in both C++ and Python. The system will be made dependable and reliable by making each subsystem modular and each one independent from other subsystems downstream from it. If, for example, the GUI application were to malfunction, the physical alert device and the components on-board the drone would be unaffected.

Significance

The ADNS system is worthwhile to pursue since it will allow the researchers utilizing the system to maximize their research efforts. This is due to the battery alert system deployed by the ADNS which helps to maximize in-flight test runs and helps in the preservation and integrity of the battery. Additionally, the proximity alert feature helps researchers in aiding the aerial drone to navigate near stationary objects or even in close-quarters such as tunnels. The ADNS system helps the development team learn how hardware and software components can be merged to create a cohesive and responsive system to accomplish predefined tasks. Moreover, the ADNS development team will gain the critical knowledge of modularization within projects and systems that help extend their longevity and adaptive usage. The ADNS project is a relatively new concept involving aerial drone autonomous testing. Although battery sensors are already integrated with some battery packs, the alerts are not clearly visible to researchers and operators as the warning indicators may be covered by equipment or simply obscured by the drone chassis. The ADNS solves this issue and integrates the proximity sensors mounted to each arm of the drone to further

assist in collision avoidance. The ADNS project is a relatively new concept built from merging existing projects located in the references section. The first is a collision avoidance technique using Leddar solid-state LiDAR technology. The ADNS project uses a similar LiDAR scheme but at a much lower cost and does not use expensive computer vision processing technology. The second is a battery monitor unit for aerial drones that is mounted on the drone but cannot convey the readings wirelessly back to the user on the ground. The ADNS project makes the jump into wireless transmission of the voltage value pulled from an on-board voltage sensor. The last similar project that ADNS is built upon is a battery monitoring system. This system uses an on-board camera system and the values are projected to the user via a video feed. The issue with this scheme is that it does not monitor the voltage levels of the battery in real-time whereas ADNS does convey battery voltage levels in real-time. The business and market potential for ADNS caters to researchers and hobbyists who want to maximize test flights. The ADNS scheme relieves the user of the burden of battery and proximity monitoring for a fraction of the price and the sensors are all interchangeable to accommodate several dozen sensors or just a few. The open-source community can also benefit from the ADNS project as improvements to sensor readings or wireless transmission can be encrypted to ensure privacy of data. The ADNS project can be easily expanded by adding additional sensors or warnings to any of the components. For example, an altimeter sensor can be integrated using protocols already being utilized such as the I2C or an independent altimeter sensor can be added to unused pins on the microcontroller.

Functionality Implemented

Upon completion and delivery of ADNS, all major features requested by the project sponsors have been implemented. The use cases and functional requirements detailed in the project's requirements specification documents which have been implemented are described

below. First, each use case which has been implemented is listed along with a brief description of all the features of the system which pertain to that use case. Next, all the functional requirements which are satisfied are listed followed by a description of the components of the system which fulfill each requirement.

Use Cases

To make the design and specification aspects of the project clear and concise, the team provided a list of use cases documented below:

1. SystemStart: The wiring harness and ToF sensors, on-board the drone, are secured to the drone arms by way of custom made sensor mounts. The subsystem on-board the drone also allows for hot swapping of sensors and indicates whether the subsystem is operational or not by way of an RGB LED.
2. GetBusVoltage: The INA219 Voltage/Current sensor module integrated into the on-board subsystem obtains the bus voltage aligned in-series with the battery.
3. ReadRangeContinuousMillimeters: The 14 VL53L0X ToF sensors integrated into the on-board subsystem provide the proximity between the sensor being polled and any stationary or moving objects that the drone may collide with. The distance value returned is in millimeters (mm).
4. SendData: The nRF24L01 transceiver module integrated into the on-board subsystem sends a packaged uint8_t array to the nRF24L01 transceiver module integrated into the ground base unit. The array sent contains the pertinent measurement data.
5. ProvideVoltage: The subsystem on-board the drone wirelessly sends voltage data to the ground unit subsystem.

6. ProvidePosition: The subsystem on-board the drone wirelessly sends proximity data from each of the 14 ToF sensors to the ground unit subsystem.
7. ConfigureQuaternion: The ground unit opens a configuration file containing the x, y, and z translation and pitch, roll, and yaw rotation of each sensor mount on-board the drone, uses the data in the file to build an XML launch file, and publishes it as a static transform through ROS. The GUI application has a drop-down menu which allows for the Set Quaternion Data window to be opened and for x, y, z, pitch, roll, and yaw values to be entered for each sensor on the drone. Each sensor offset is saved to a configuration file which is used to build the quaternion launch file upon system startup if the save button is pressed.
8. ReadVoltage: The ground unit subsystem publishes voltage data as a ROS topic named “voltagePublish”. The ground unit subsystem also displays the voltage data in real-time on a seven segment. Additionally, the ground unit subsystem sends the voltage data to the GUI application by way of a logical pipe. The GUI application receives voltage data through the pipe from and displays it in a text field, which is continually updated in a separate thread.
9. ReadPosition: The ground unit subsystem publishes proximity data as a ROS topic name “proximityPublish”. The ground unit subsystem also displays the proximity threshold on a seven segment.
10. SetThresholds: The ground unit subsystem receives thresholds for both voltage and proximity from the GUI application by way of a logical pipe, defaulting to predefined values if none are sent.
11. VisualAlert, AudioAlert: The ground unit subsystem compares values as they are

streamed in from the drone with the thresholds and issues alerts for either or both drone voltage and/or proximity by activating LEDs and the speaker. The speaker volume is modulated by way of a potentiometer. The GUI application issues a visual alert for the voltage value by changing the color of the main GUI window to red.

12. ResetAlert: The individual alerts may be deactivated by hitting a reset button for either or both drone voltage and/or proximity, and the speaker volume is modulated by way of a potentiometer.
13. ProvideOther: The ground unit subsystem subscribes to eight ROS topics which represent generic sensors, and which activate eight corresponding LEDs on the ground unit.
14. PublishOther: The ground unit subsystem publishes four ROS topics which represent generic sensors, and which correspond to four switches on the ground unit.
15. OpenGUI: The GUI application immediately opens upon start-up of the ground base unit subsystem.
16. SetThresholds: The GUI application has a drop-down menu which allows for the Set Thresholds window to be opened and for voltage and proximity thresholds to be entered text entry boxes which sends them through a logical pipe to the ground unit with the click of their corresponding buttons.
17. SaveValues: The GUI application allows the voltage value coming in from the ground unit subsystem to be saved to a file along with a timestamp if the user clicks the save button, which allows the user to then inspect the values over time.

Functional Requirements

Below is list of functional requirements that team had engineered and implemented:

1. *FR01. [1] ADNS shall gather voltage levels from the drone battery and send them to a physical device by WiFi.* The subsystem on-board the drone meters the drone battery by way of an INA219 voltage sensor and sends the voltage value wirelessly to the ground base unit by way of a pair of nRF24L01 RF transceivers.
2. *FR02. [1] ADNS shall gather proximity data from the sensors and send them to a physical device by WiFi.* The subsystem on-board the drone meters the drone battery by way of 14 VL53L0X ToF sensors and sends the values wirelessly to the ground base unit by way of a pair of nRF24L01 RF transceivers.
3. *FR03. [1] ADNS shall send voltage data from the physical device to a ROS node.* The ground base unit subsystem receives the voltage data from the RF transceiver over the serial port, parses, formats, and checks the data for errors and sends the data to the module which acts as the ROS interface and publishes the voltage as a string to a ROS node called “voltagePublish”.
4. *FR04. [1] ADNS shall send proximity data from the physical device to a ROS node.* The ground base unit subsystem receives the proximity data from the RF transceiver over the serial port, parses, formats, and checks the data for errors and sends the data to the module which acts as the ROS interface and publishes the proximity data as a Marker message to a ROS node called “proximityPublish”.
5. *FR05. [1] ADNS shall issue visual and audible warnings if the voltage is outside of an acceptable threshold range.*
6. *FR06. [1] ADNS shall issue visual and audible warnings if the proximity is outside*

of an acceptable range. The ground base unit subsystem, while reading in both the voltage and proximity data streaming in from the drone, continually checks each value against the thresholds for each (which are set to defaults upon startup of the system and thereafter configurable by the user through the GUI), and issues alerts if the data are outside of their acceptable ranges. The ground unit activates a separate LEDs if either voltage or proximity are below threshold. The GUI subsystem also issues visual alerts if the voltage goes below threshold by way of changing the color of the GUI windows to red.

7. *FR08. [1] The user shall be able to monitor pertinent drone data.* The ground base unit subsystem displays both the voltage values and the proximity threshold on seven segments, thereby allowing the user to monitor their values. The proximity threshold is displayed to the user, rather than the proximity values themselves, because there are too many sensors which change their values very rapidly making such a display of little utility to the user.
8. *FR09. [1] The user shall be able to set customizable thresholds.* The user can open the Set Thresholds window from the main window in the GUI and set both the voltage and the proximity thresholds, which are then set back into the ground base unit subsystem through the logical pipe where they are compared with the values coming from the drone.
9. *FR10. [1] The user shall be able to reset all alerts.* The ground base unit has a button for both the voltage alert and the proximity alert which, if and alert is activated, allows the user to reset or silence the alert even while the values remain outside of their acceptable ranges.

10. *FR11. [1] The user shall be able to enter quaternion data for each sensor.* The user can open the Set Quaternion Data window in the GUI and configure the x, y, z, pitch, roll, and yaw for each sensor on board the drone and then save the changes to the configuration file. This file is used to build the XML launch file so that upon startup of the system the quaternion data for the drone will be different, which allows the user to add different ToF sensor or to move the sensor mounts in the future.
11. *FR12. [1] ADNS shall initialize thresholds with defaults.* Before the user specifies the thresholds for the voltage and proximity in the Set Thresholds window of the GUI, the system sets the thresholds to default values upon startup.
12. *FR14. [2] ADNS should publish test topics via button presses and switch toggles.* The ground base unit subsystem has four switches which are tied to ROS nodes which publish generic flags so that the user can physically publish to ROS through the system hardware.
13. *FR15. [2] ADNS should subscribe to auxiliary topics in addition to main topics.*
14. *FR16. [2] The user should be able to visually see activity on auxiliary topics via LEDs.*
15. *FR18. [2] ADNS might support additional sensors, such as a Geiger counter, barometer, gyroscope, etc.* The ground base unit has in addition to LEDs for the voltage and proximity eight LEDs of another color (green instead of red) which can be tied to generic alerts by simply publishing a 1 to any of eight ROS topic named “genericLEDNode_x” where x is a number 0 through 7. This allows for the user to add more sensors to the drone in the future and then, on another machine which can

read those values, publish to the corresponding topic to issue the alert.

16. *FR17. [2] The user should be able to adjust alert volume.* The ground base unit has a potentiometer which acts as a volume control for the speaker so that when an alert is issued, and the speaker is activated the user can turn the volume up or down.
17. *FR20. [3] ADNS might represent quaternion readings in the GUI.* The contents of the sensor offset configuration file are read into the text fields in the Set Quaternion Data window so that they can be altered if the position of the sensor mounts is changed on the drone.

Functionality Not Implemented

In addition to the functionality that was implemented, there remains one use case which is left unimplemented at this time. The following use case, PlotValues and its corresponding functional requirement, were not implemented as the use case was a stretch goal of low importance and there was not sufficient time to develop it:

1. *PlotValues FR13. [2] The user should be able to save and plot data.* The main window of the GUI does not allow the user to start a dynamic plot of the voltage values streaming in from the ground base unit subsystem which was to display each voltage data point as a function of time. The purpose of the plot was so that the voltage curve could be visualized to give the user a sense of the slope as the decay in voltage is not linear and drops off precipitously. This feature was not implemented because no good means of dynamically plotting the voltage in the window was discovered in time to build the feature. A dynamic plot would need both a canvas or other GUI window object in the GTK+3 framework which would provide a venue in which to display the plot, as well as some means of drawing the

plot at fixed intervals. Many tools exist in Python for producing static plots, such as those found in the matplotlib library, however no suitable tools were found for dynamically drawn and scaled plots. Further, the project sponsors did not explicitly request this feature, instead the team decided to attempt to add it to the system and so this omission does not adversely affect the project deliverables from the perspective of the sponsors.

Software Overview

On-board Subsystem

The Aerial Drone Notification System (ADNS) on-board subsystem is comprised of a Teensy 3.2 ARM microcontroller, 14 Time of Flight (ToF) sensors, an INA219 current/voltage sensor, and a nRF24L01 2.4 GHz wireless transceiver unit. The Aerial Drone Notification System is composed of two separate systems that communicate via radio frequency (RF) wireless using the NRF24L01 transceiver modules. The two systems considered are the on-board unit mounted to an aerial drone and a ground base unit that is used to capture the data from the on-board system and process the data accordingly. The system is designed to accommodate up to 20+ VL53L0X ToF sensors using independent addressing through the I2C bus. The ADNS system currently uses 14 ToF sensors and an INA219 voltage/current sensor to grab voltage levels from the aerial drone's battery pack. The sensors are sequentially polled and transmitted wirelessly from the on-board system to the ground base system in a stream of data that is, unfortunately, unreliable (think UDP). To combat the unreliable datagrams, the ADNS team chose to send each data stream independently. This is done to reduce the amount of bad or corrupted datagrams as well as increase the throughput of the wireless system. Features for the ADNS system include independent I2C addressing and high-speed polling rates of the ToF sensors. Currently and collectively, the ToF

sensors poll at about 56 Hz. The wireless data rate has been set to the lowest setting possible at 250 kbps to extend the possible ranges of the NRF24L01 transceivers and due to the bottleneck of the system which are the ToF sensors. Additionally, the on-board system has a software reset feature that will attempt to restart the system if a sensor has timed out. This presents a good robust and recovery method but also introduces the possibility of a single sensor triggering a constant reboot of the on-board system. The estimated reboot time is around 500ms (half a second). Another feature implemented is the auto restart of the system in the event of system initialization failure. The ADNS system begins by trying to poll a set amount of readings for the system upon boot and will trigger a red LED alarm if even one sensor fails to initialize properly. The set amount of readings or polls can be set by the user and default is 250 readings. Additionally, if the ADNS system successfully initializes, a green LED is turned on constant state to indicate successful initialization. Moreover, if a sensor is not seated properly, the user will see the triggered alarm and can reseat the faulty sensor to re-initialize the system.

Ground Base Unit Subsystem

The ground base unit subsystem is composed of a main driver and seven classes, all divided into separate Python files. The main driver first instantiates a QuaternionManager object, a Parser object, a ROSNodeManager object, a GenericLEDs object, ShiftSevenSegment and PinSevenSegment objects, and several objects from the GPIOZero library which control the hardware on the ground unit. The QuaternionManager class opens the sensor offsets configuration file which contains the x, y, z, pitch, roll, and yaw values for all 14 ToF sensors on-board the drone and builds a launch XML file which publishes the offsets as a static transform. The Parser class creates a SerialPort object which it uses to read in data over the serial port which it then parses and formats. The data it brings into the main driver is then passed off to the ROSNodeManager object

which it publishes over ROS. Additionally, the ROSNodeManager object continually listens for ROS topics named “genericLEDNode_0” through “genericLEDNode_7” and the GenericLEDs object drives the LEDs tied to those topics. The ROSNodeManager object also continually publishes the state of four generic switches as ROS topics. The ShiftSevenSegment class objects display the values coming into the ground unit on the seven segments. The ground unit also starts a process which runs the GUI and creates a logical pipe through which to pass data back and forth from the GUI, include the voltage value and the thresholds for both the voltage and proximity. The main driver also creates several other processes and threads which control the hardware on the ground unit, and its main loop continually reads data from the drone, publishes it through ROS, exchanges data with the GUI, and controls the hardware to issues alerts whenever the data falls outside acceptable ranges. Included in the software package is a lightweight, headless version of the system which omits the GUI and the ground unit hardware and only forwards the data coming in from the drone through ROS. This allows the core functionality of the system to be moved or customized for other uses. Below is a brief description of each class used by the main driver described above:

- QuaternionManager
  - Opens the sensor offset configuration file, reads in its values, and builds a ROS launch XML file from the values.
  - Launches the XML which publishes the sensor offsets as a static transform.
- Parser
  - Instantiates a SerialPort class object.
  - Reads in data from the SerialPort object, parses, formats, and returns it to the main driver.

- SerialPort
  - Searches all open serial ports for the one which is connected to the RF transceiver.
  - Sets the permissions for the serial port as well as the baud rate and opens a connection.
  - Reads in data from the serial port and returns it to the Parser class.
- ROSNodeManager
  - Creates several ROS nodes, include those to publish the voltage and proximity data, those to publish the switch flags, and those to subscribe to the generic alerts.
  - Creates std_msgs data types (String, Marker, Int8) which are used by the nodes to send and receive data.
  - Converts the topics coming in from the eight generic alerts into a hexadecimal number which it passes to the GenericLEDs class to drive the generic LEDs.
- GenericLEDs
  - Sets up the addresses for the I2C bus which drives the generic LEDs.
  - Continually tries for new addresses and catches exceptions if the address is momentarily lost.
  - Takes a hexadecimal value and controls eight pins which are tied to the generic LEDs.
- ShiftSevenSegment
  - Sets the pins on the Raspberry Pi to control the seven segments.

        - Uses the 74LS138, 74LS164, and 74LS373 chips to shift in the values to be displayed from the shift register into the D-latch, and to select the appropriate digit on the display so that only two data lines are needed for both voltage and proximity.
        - Loops through the digits in the display and lights up different segments to show the four-digit number.
    - PinSevenSegment
        - Sets the pins on the Raspberry Pi to control the seven segments.
        - Loops through the digits in the display and lights up different segments to show the four-digit number.
        - Is intended to act as a backup for seven segment controls if any of the chips needed by the ShiftSevenSegment class are faulty.

GUI Application Subsystem

The GUI application is designed to be simple and effective when conveying data values or setting the thresholds of the system. In like manner, it should be a simple design that allows users of the system to view data in real-time. Additionally, the GUI features a reset capability if physical resets are not functioning properly. Moreover, the GUI application subsystem includes the sensor offset feature within the QuaternionWindow module that allows the users of the system to set different offsets for each individual sensor. This feature is necessary as the user may choose to view a representation of the position of the drone, as well as showing sensors as vectors, in ROS. To correctly position the sensor vectors on the drone, the sensor positions must be read into ROS as a set of quaternion values, and this option allows the user to either manually enter offsets or load them from saved profiles. The Aerial Drone Notification System allows the users to save the

thresholds for each test flight. The user can also save these values to be included in the file system for later use. The Aerial Drone Notification System includes a ThresholdWindow module that displays an alert not only on the portable alert box but also on the GUI. Users of the system will see the GUI flash red when a threshold has been met or exceeded. The main window MyWindow will issue the alert when the event occurs. The alert must be vibrant to be effective and so any alert may cause the window to flash between alert and non-alert states. Below is a brief description of each class used by the GUI main driver described above:

- MyWindow
  - Displays voltage and proximity data coming from the Python Pipe.
  - Interface with the Python Pipe via multi-threading.
  - Instantiate the ThresholdWindow and QuaternionWindow modules. ○ Save voltage and proximity data to log files.
- ThresholdWindow
  - Allows user to input upper bound voltage and proximity thresholds.
  - Allows user to input lower bound voltage and proximity thresholds.
  - Checks the thresholds against incoming voltage and proximity data. ○ Issues visual alert if the thresholds are exceeded.
  - Saves voltage and proximity thresholds to log files.
- QuaternionWindow
  - Display quaternion data regarding ToF sensors' rotational and transformational information
  - Utilize regular expression parsing to read quaternion data from configuration file into text-fields.

- Write quaternion data into configuration file from manual inputs at the text-fields.

Updated Design

Summary of Changes in Project Design

The high-level view of ADNS has not changed, as shown in the context diagram shown in Figure 1. There have been some slight changes at the subsystem level, however. Originally, the sensor harness aboard the drone was meant to me modular, allowing it to fit several different drone form factors and sensor configurations. Since ARL experiments almost exclusively with the Matrice 100 currently, it was determined that a standard harness to fit that specific drone was sufficient. As mentioned earlier, support for current tracking has been removed so any classes or methods concerning current have been removed. A serial class has been added to automate initialization of drone to ground communication, so user intervention isn't required with each system boot. Lastly, the pipes data structure has been added to the GUI to allow for simultaneous send and receive communication between the Raspberry Pi and the ROS master node.

Updated High-Level and Medium-Level Design

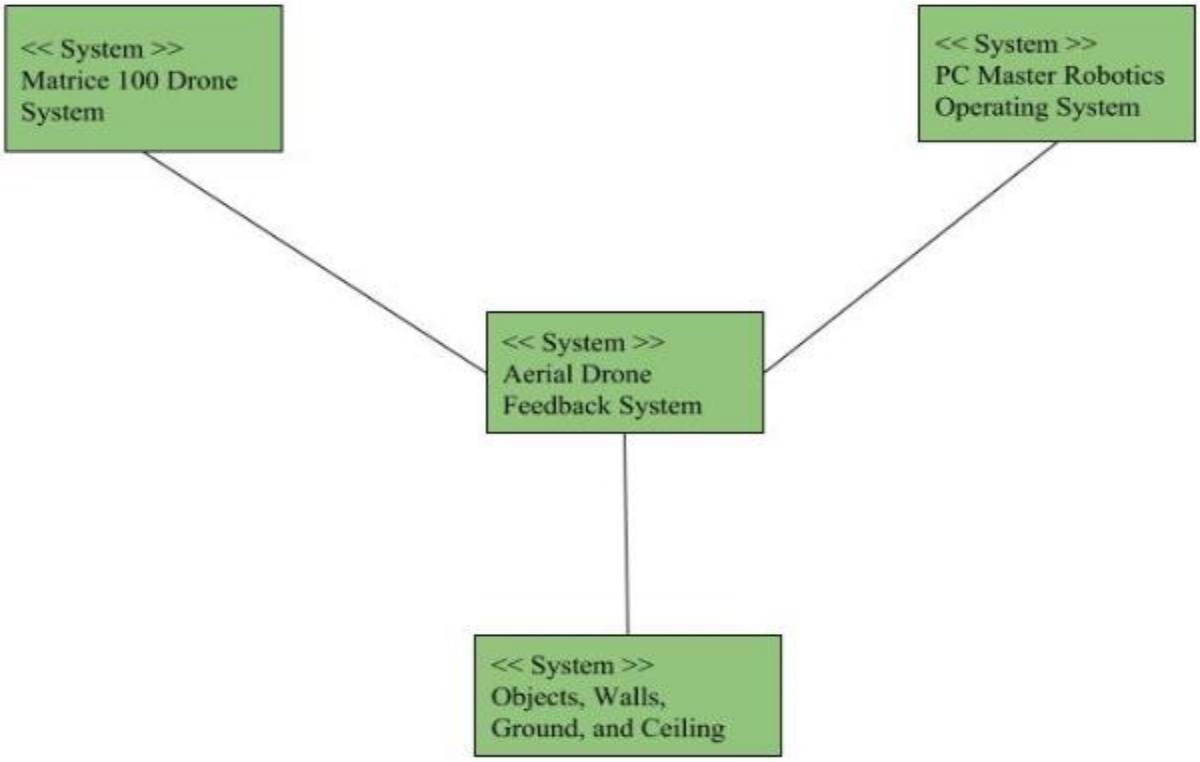

Figure 1: *Context Model for the Aerial Drone Notification System.*

Data Structures

The various classes that make up the subsystems must be able to communicate with each other for effective notification to the user. This will primarily be done using several data structures. The interface between the physical alert device and the GUI application consists of two sets of pipes, which act as logical connections between the module controlling the hardware on the Pi and the GUI. The pipes act to carry both the voltage and proximity data to GUI text fields, and to send threshold values in GUI thresholds into the hardware module. Another important data structure used by the system is a list, which will hold the values recorded in the plot corresponding to the voltage values. If the user chooses to save a given plot, then the values plotted will be written to a file. The data structures are summarized below.

Pipe (voltage and proximity values)

The pipe is a multiprocessing construct, which connects different processes and sends data between them; these pipes send voltage and proximity data from the hardware module on the Pi to the GUI application.

Pipe (voltage and proximity thresholds)

The pipe is a multiprocessing construct, which connects different processes and sends data between them; these pipes send voltage and proximity thresholds from the GUI application to the hardware module on the Pi to allow for dynamic threshold modifications.

List

If the user chooses to plot voltage values, then those values will be fed into the appropriate list. The contents of the list will be written to a file if the user choses to save the plotted values.

Updated Hardware Design

The subsystems that are hardware-based, the drone and the physical alert device, along with all the components, which make them up, are described in the following sections. Both component diagrams and pictures of the hardware components which make up the subsystem on the drone are shown. Each component is also described in detail.

ADNS uses several intercommunicating components to function but is also modular in the event of a component failure. Figure 2 shows how each component interacts with its counterparts to complete the system.

High-Level Hardware Design Diagram

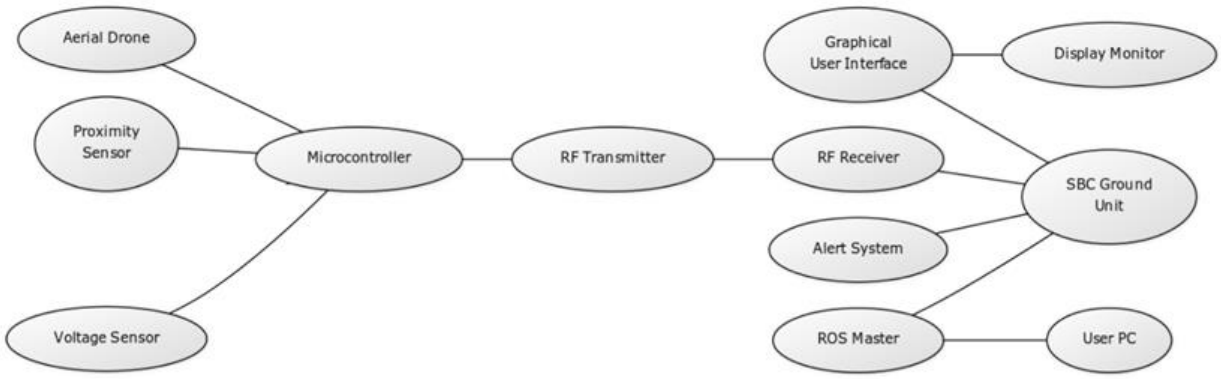

Figure 2: *High-level design diagram of the system and its hardware components, both the hardware on board the drone and the hardware comprising the physical alert device.*

Drone Hardware Component Descriptions

DJI Matrice 100

Figure 3 shows the Matrice 100, an aerial drone that the sensor system will be mounted to for testing of the alert and notification system and its components. It is a quadcopter aerial drone that is fully customizable and programmable flight platform that lets you to turn your ideas and dreams into reality. Equipped with DJI's easy-to-fly technology and optimized for easy programming through the DJI SDK, the Matrice 100 will carry the various sensors and additional devices that researchers want to put in the sky.

Figure 3: *DJI Matrice 100 Aerial Drone.*

Teensy 3.2 Microcontroller

Figure 4 shows the Teensy 3.2 microcontroller which features a 32-bit Advance RISC Machine, ARM, development board. Features include multiple channels for Direct Memory Access (DMA), high-resolution Analog-to-Digital Converter (ADC), and I²C compatibility. The Teensy 3.2 has 34 digital I/O pins to ensure that the notification system has room for future extensions including adding or removing additional sensors.

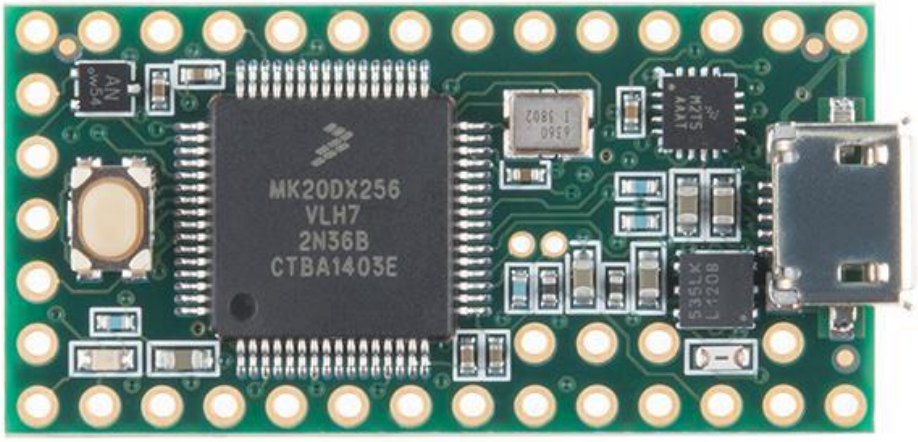

Figure 4: *Teensy 3.2 Microcontroller.*

VL53L0X Time of Flight (ToF) sensor

Figure 5 shows the VL53L0X ToF sensor, which measures the range to a target object up to 2 meters away. The VL53L0X uses time-of-flight measurements of infrared pulses for ranging, allowing it to give accurate results independent of the target's color and surface. Distance measurements can be read through a digital I²C interface and can be configured to poll continuously at an increased rate of 50Hz. The proximity measurements for the notification system will use several VL53L0X sensors to ensure drone proximity warnings are triggered when necessary.

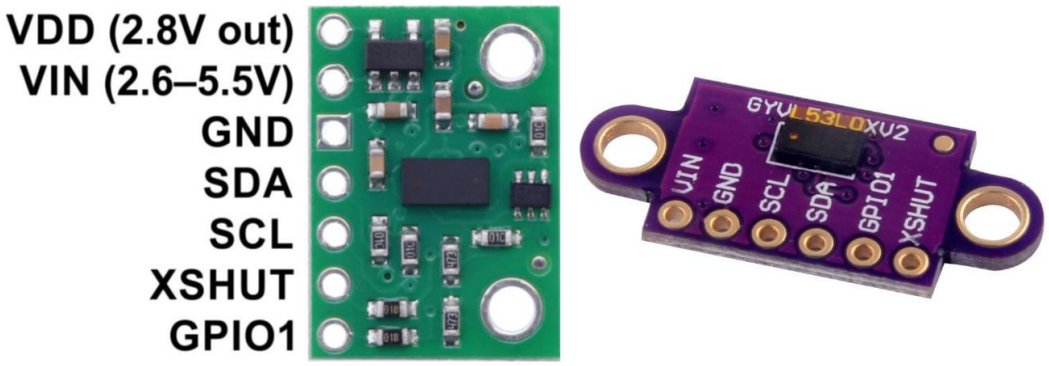

Figure 5: *VL53L0X Time-of-Flight Sensor.*

INA219 DC Current &Voltage Sensor

Figure 6 shows the INA219 voltage and current sensor module that is to be used to poll real-time battery DC voltages from a connected device. It is a precise bidirectional, high-side device that is powered by a load source that can range from 3V to 25V DC readings.

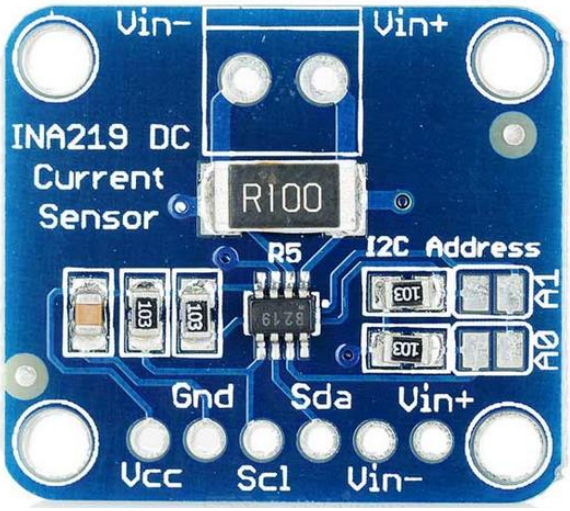

Figure 6: *INA219 Current & Voltage Sensor.*

Nordic nRF24L01 RF Transceiver Module

Figure 7 shows the nRF24L01 transceiver module that utilizes the ISM 2.4 GHz wireless frequencies to transmit the pertinent data gathered by the on-board system to the ground base system. The transceiver module is capable of transmission speeds up to 2Mbps and has an extended range of 200 meters.

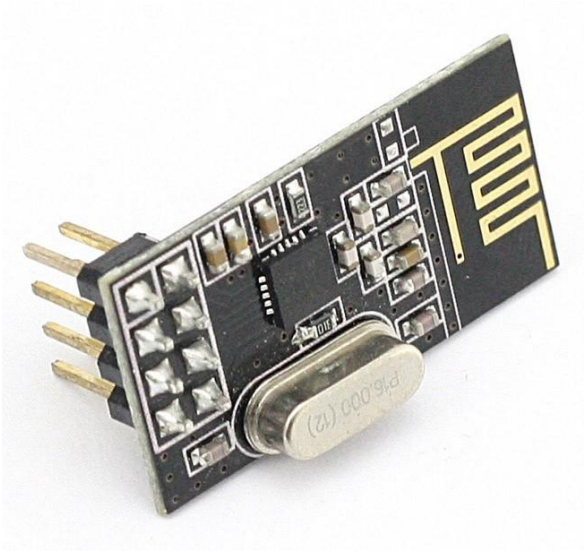

Figure 7: *nRF24L01 RF Wireless Transceiver.*

Physical Alert Device Components

In addition to the drone-based subsystem and its components, the other hardware-based subsystem is the physical alert device, which acts as the ground base unit that both manages the data that comes in from the drone and issues alerts. The device also runs the GUI application in Ubuntu Mate. Figure 8 below shows the physical alert device component diagram, and following are descriptions of the components that make up the physical alert device.

Physical Alert Device High-Level Design Diagram

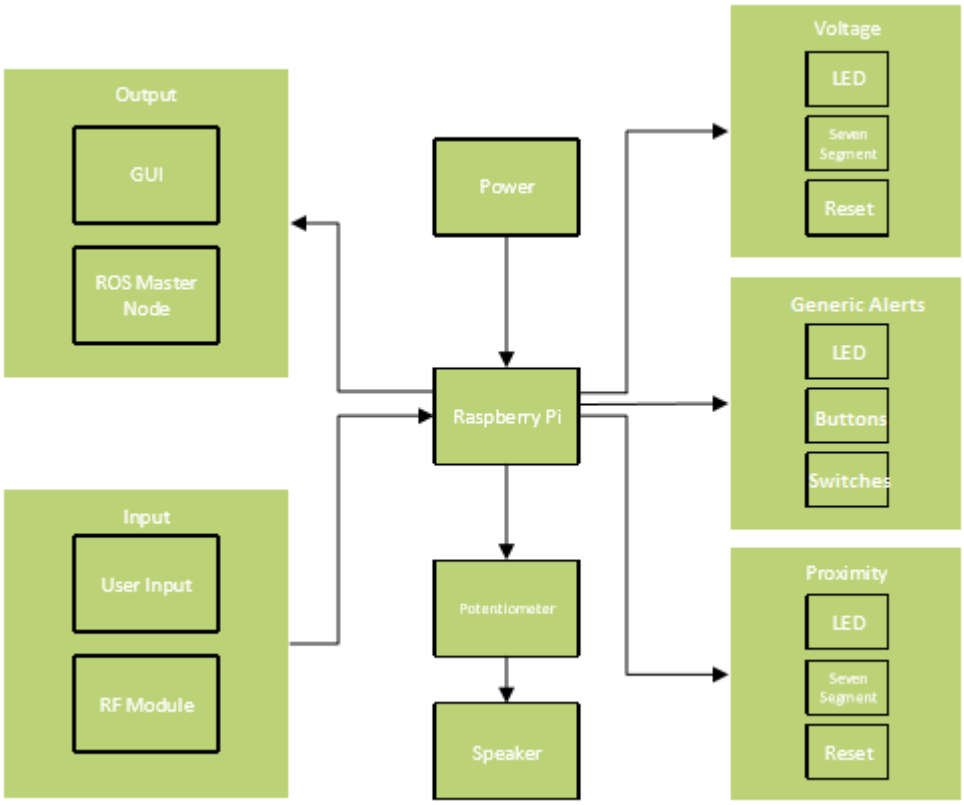

Figure 8: *High-level design diagram of the components that make up the physical alert device and their interactions.*

Physical Alert Device Hardware Component Descriptions

Raspberry Pi

The single board computer, SBC, which controls the physical device and runs Ubuntu Mate that provides the GUI application. Below is a description of hardware components:

- *Input:*
    - *User Input:* The Raspberry Pi has built-in USB and HDMI ports to which a mouse and a monitor can be connected to allow for user input for the GUI application.
    - *RF Module:* A radio frequency device which will receive the data streamed from

the microcontroller on board the drone and send them into the software module that processes the data.

- *Output:*
    - *GUI:* The data that is received by the device from the microcontroller on-board the drone will, after being processed, be sent out to the GUI application running in Ubuntu Mate on the device.
    - *ROS Master Node*: The data is also sent to the master ROS node controlling the drone by way of publishing the data as ROS topics within ROS on the device.

- *Voltage:*
    - *LED:* A light, which will be activated if a voltage alert is issued.
    - *Seven Segment:* Four seven segment digits, which will display the voltage data in volts.
    - *Reset:* A physical switch, which will allow the user to reset the voltage alert.

- *Generic Alerts:*
    - *LED:* Several LEDs will be tied to generic ROS nodes which, when signaled, will be activated.
    - *Buttons:* Several buttons will be tied to generic ROS nodes, which will publish a flag when pushed.
    - *Switches:* Several switches will be tied to generic ROS nodes, which will publish a flag when flipped.

- *Proximity:*
    - *LED:* A light, which will be activated if a proximity alert is issued.
    - *Seven Segment:* Four seven segment digits, which will display the proximity threshold data in millimeters.
    - *Reset:* A physical switch, which will allow the user to reset the proximity alert.

- *Power:*
    - The Raspberry Pi may be powered by either an AC power supply or through a USB cable from another device.

- *Potentiometer:*
    - A variable resistance dial, which will allow the user to vary the voltage supplied to the speaker, thereby acting as a volume control.

- *Speaker:*
    - A speaker, which will be activated if any alert is issued.

Updated User Interface Design: Drone

The primary ways in which the user will interact with the system are the initial installation and setup of the drone-based hardware, interaction with the physical alert device, and through the GUI application. Photographs of the drone-based hardware are shown first, as this is how the user will initially interact with the system while installing the sensor harness. Following the drone hardware image is a screenshot of the raw data from the sensors on the drone and a number of screenshots from the GUI application.

The hardware user interface implemented should model a modular system that allows for easy replacement of faulty components that are used by the Aerial Drone Notification System. The modularity of the wiring system allows for an easy snap-on/snap-off replacement feature that makes repairs easy, safe, and effective. Figure 9 shows this modularity feature when the user interacts with the sensor system. This modularity aspect is critical since different manufacturers for the ToF sensors use different pin configurations. The sensor system is designed to adapt to these types of changes.

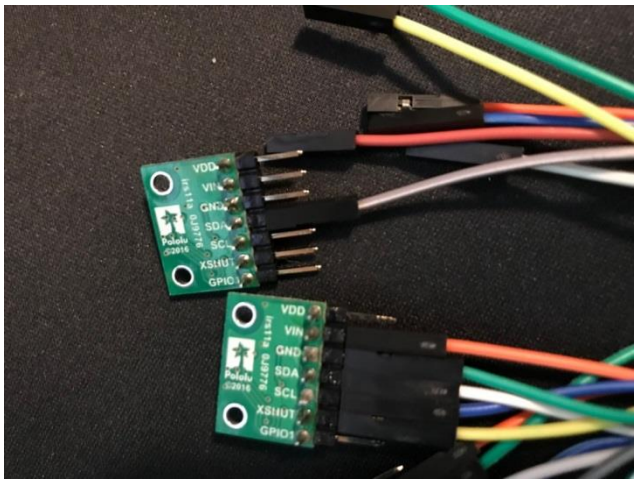

Figure 9: *Modularity aspect for the sensor modules is critical since the pin configurations for different manufacturers selling the same type of sensor may vary.*

Figure 10 shows a simple interface between the different sensor readings to be displayed to the user. The user will be able to easily spot sensors that are faulty or otherwise simply not working when the system is initialized or while in test flight autonomous mode.

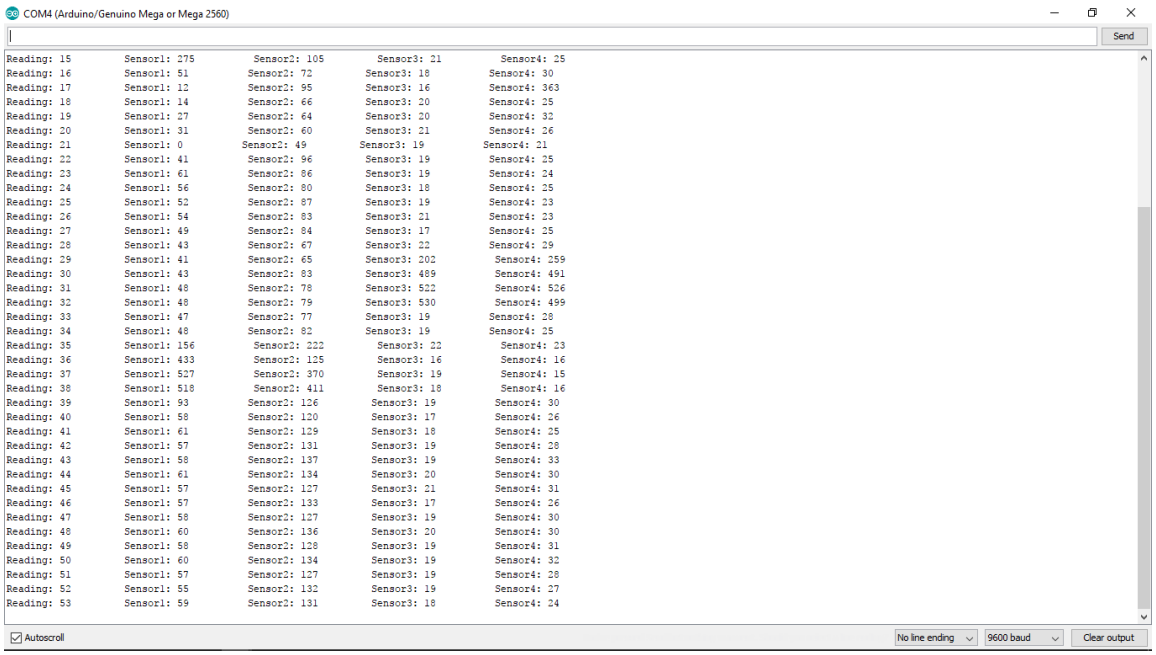

Figure 10: *Initial output of each sensor's proximity reading to an object. Each value is represented in millimeters since distance precision is an important variable feature. Output format is subject to change.*

Updated User Interface Design: Physical Alert Device

The physical alert box is designed to be simple and displays clear warnings to the operator. Figure 11 shows a model of the physical alert device, which has seven segment displays for each alert type, along with an LED and a reset switch. There are LEDs, switches, and buttons for several generic alerts tied to ROS nodes. There is also a speaker and a potentiometer, which acts as a volume control dial, as well as a USB port for a mouse and an HDMI port for a monitor.

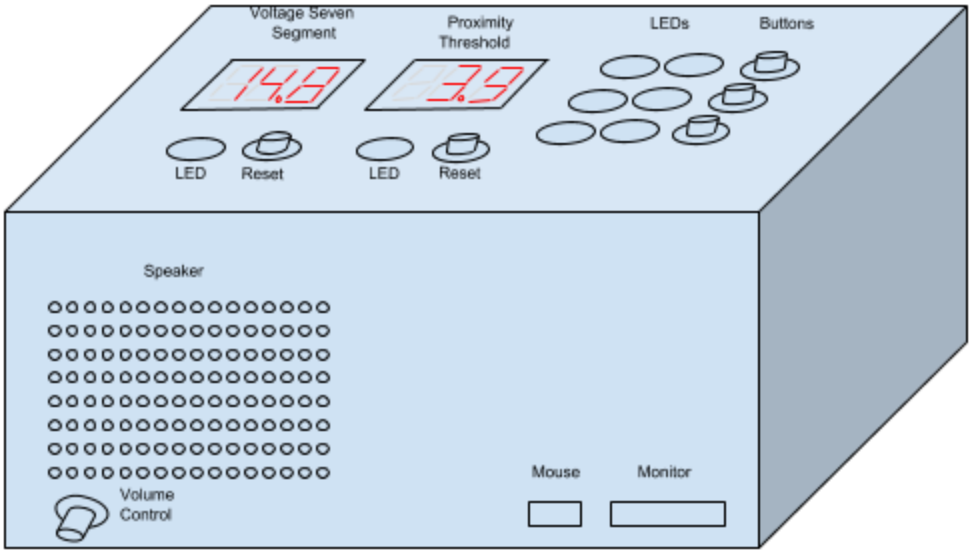

Figure 11: *Early box concept featuring seven-segment displays for tracking voltage and current, visual led alerts, speaker for audible alerts, volume control, alert resets, and input/output interfaces.*

Updated User Interface Design: GUI Application

The GUI application is also designed to be simple and effective when conveying data values or setting the thresholds of the system. Figure 12 shows a simple design that allows users of the system to view data in real-time. Additionally, the GUI will also feature a reset capability if physical resets are not functioning properly.

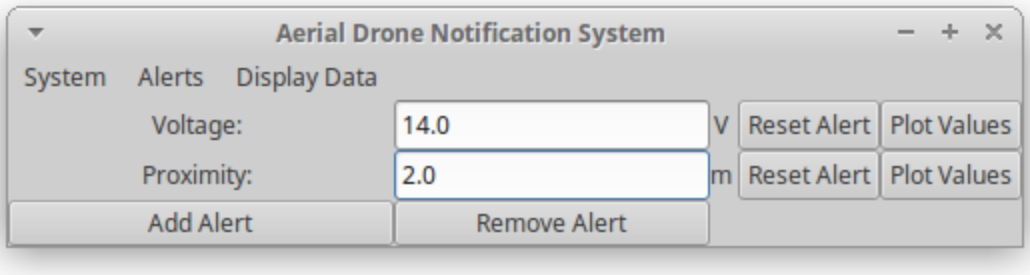

Figure 12: *The main window of the GUI application, showing the values for voltage and proximity*

and their units, buttons to reset their alarms, buttons to plot or display their values, and buttons to add or remove user-defined alerts.

Figure 13 shows the sensor offset feature that allows the users of the system to set different offsets for each individual sensor. This feature is necessary as the user may choose to view a representation of the position of the drone, as well as showing sensors as vectors, in ROS. In order to correctly position the sensor vectors on the drone, the sensor positions must be read into ROS as a set of quaternion values, and this option allows the user to either manually enter offsets or load them from saved profiles.

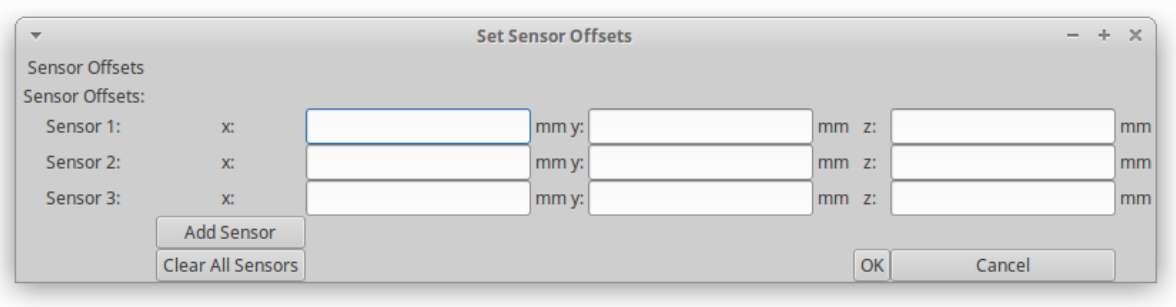

Figure 13: *The set sensor offsets window, which allows the user to specify the location of each proximity sensor on the drone as measured off from some reference point. Also shown are buttons to add more sensors or to clear all values.*

The Aerial Drone Notification System will allow the users to save the thresholds for each test flight. Figure 14 showcases how a user might save these values to be included in the file system for later use.

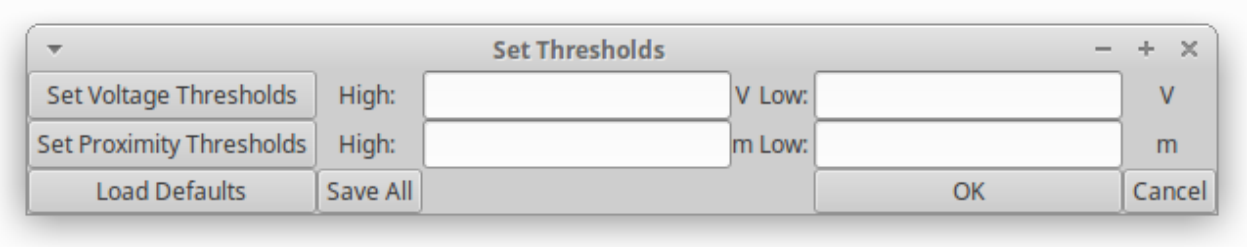

Figure 14: *The set thresholds window showing the ability to enter high and low values, which define the upper and lower thresholds and buttons to save each one, save them all, or load default values.*

The Aerial Drone Notification System will display an alert not only on the portable alert box but also on the GUI. Users of the system will see the GUI flash red when a threshold has been met or exceeded. Figures 15 and 16 show the main window and a warning window when an alert is issued. The alert must be vibrant to be effective and so any alert may cause the window to flash between alert and non-alert states.

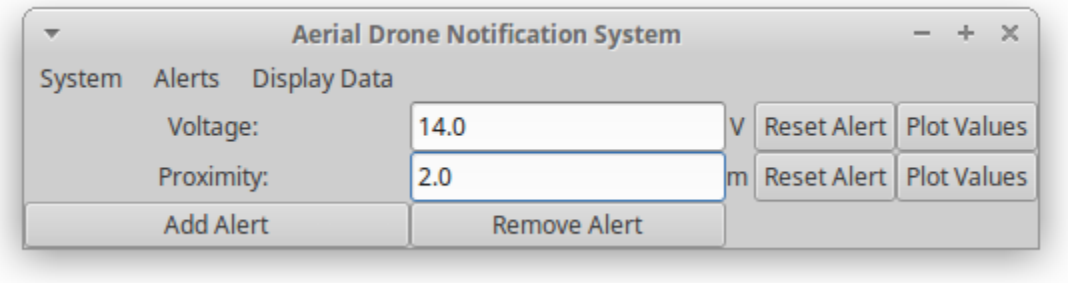

Figure 15: *The main window in the non-alert state before an alert state is triggered.*

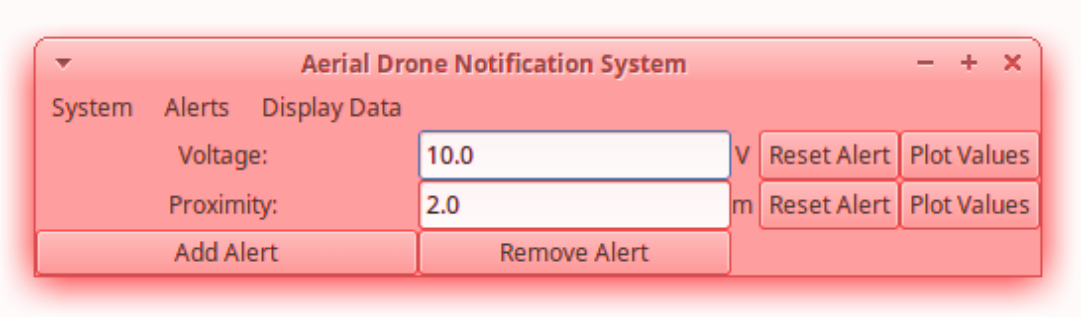

Figure 16: *The main window showing an alert state for voltage falling below the threshold.*

Contributions

Jia Li

~100 hours – ADNS GUI subsystem code: ~1500 lines

Jia's primary responsibilities regarding the GUI application unit subsystem on the ADNS project included:

- Development, optimization, and testing of the ADNS_Main_GUI subsystem.
- Development, optimization, and testing of the MyWindow main driver module, which shows incoming voltage and proximity data in real-time.
- Development, optimization, and testing of the ThresholdWindow module that allows user to dynamically adjust voltage and proximity threshold values.
- Development, optimization, and testing of the QuaternionWindow module that allows sensor offsets to be entered.
- Development, optimization, and testing of the warning feature that visually alerts the user by changing the rgba value for the window interface from green to red.
- Development and testing of log file creation for voltage and proximity data and thresholds. Primarily a testing feature to visualize voltage and proximity data.
- Integration of ADNS_Main_GUI subsystem into ADNS. ● ADNS GUI subsystem

documentation.

Rony Calderon

~115 hours – On-board unit code: 518 lines & Ground base unit transceiver code: 205 lines

Rony's primary responsibilities regarding the sensor network unit on the ADNS project included:

- Development, research, and optimization of the Teensy 3.2 ARM microcontroller for sequential polling and software reset capabilities.
- Development, research, optimization, and testing of the VL53L0X Time of Flight sensors that operate at maximum polling efficiency and longest-range detection possible.
- Development, optimization, and testing of the INA219 current/voltage sensor to obtain bus voltage from an external battery source.
- Development, research, optimization, and testing of the nRF24L01 on-board transceiver module to ensure fast and accurate transmission rates between the on-board subsystem and the ground base unit subsystem.
- Development, optimization, and testing of the nRF24L01 ground base unit transceiver module for fast and accurate reception of the raw data sent from the on-board subsystem.
- Construction of the on-board wiring harness to be mounted specifically for the Matrice 100.
- Design of the on-board subsystem schematic diagram.
- System integration testing and optimization between the ground base unit subsystem.

- On-board & ground base unit wireless communication testing and optimization to ensure distance coverage between the nRF24L01 transceivers.
- On-board subsystem documentation.

Bryan Kline

~153 hours – Ground Base Unit code: ~1500 lines, ROS Test Modules: ~450 lines, Headless Ground Base Unit ~200 lines, pair programmed with Robert Watkins (~1100 lines individually)

Bryan's primary responsibilities regarding the ground base unit on the ADNS project included:

- Developed and tested the ground base unit main driver.
- Developed and tested supporting classes: Parser, SerialPort, ROSNodeManager, ShiftSevenSegment, PinSevenSegment, GenericLEDs.
- Developed test GUI and test On-board modules to use to build and test the ground unit subsystem.
- Developed the ROS test modules to build and test ground unit subsystem.
- Developed the threads and logical pipes to enable the system to exchange data between processes.
- Researched, downloaded, installed, and configured packages, libraries, and other dependencies to enable the Raspberry Pi to read from the serial port, control the I2C bus, control GPIO pins, and publish and subscribe to ROS topics over the network.
- Designed, planned, constructed, and soldered hardware components including LEDs, buttons, switches, amplifier, potentiometer, speaker, I2C bus, and seven segment displays.

- Ground base unit physical planning, drawing, and construction.
- Ground base unit faceplate acrylic planning, drawing, and cutting.
- System level integration and testing.

Robert Watkins

~165 hours – Ground Base Unit code: ~1500 lines, ROS Test Modules: ~450 lines, Headless Ground Base Unit ~200 lines, pair programmed with Bryan Kline (~1100 lines individually)

Robert's primary responsibilities regarding the ground base unit on the ADNS project included:

- Developed and maintained the project website.
- Developed and tested the ground base unit main driver.
- Developed and tested supporting classes: Parser, SerialPort, ROSNodeManager, ShiftSevenSegment, PinSevenSegment, GenericLEDs.
- Developed test GUI and test On-board modules to use to build and test the ground unit subsystem.
- Researched, downloaded, installed, and configured packages, libraries, and other dependencies to enable the Raspberry Pi to read from the serial port, control the I2C bus, control GPIO pins, and publish and subscribe to ROS topics over the network.
- Designed, planned, constructed, and soldered hardware components including LEDs, buttons, switches, amplifier, potentiometer, speaker, I2C bus, and seven segment displays.
- Designed, printed, and installed drone arm sensor mounts.
- Designed, printed, and cut physical interface graphic.

- Ground base unit physical planning, drawing, and construction.
- Ground base unit finishing and painting.
- Ground base unit faceplate acrylic planning, drawing, and cutting.
- System level integration and testing.