Software Formalization

Year: 2020 **Semester:** Fall **Team:** 6 **Project:** Snow-weAR Goggles
Creation Date: 27 September 2020 **Last Modified:** 5 October 2020
Author: Kennedy Monaco **Email:** monacok@purdue.edu

Assignment Evaluation:

Item	Score (0-5)	Weigh t	Point s	Notes
Assignment-Specific Items				
Third Party Software		x2		
Description of Components		X3		
Testing Plan		x3		
Software Component Diagram		x4		
Writing-Specific Items				
Spelling and Grammar		x2		
Formatting and Citations		x1		
Figures and Graphs		x2		
Technical Writing Style		x3		
Total Score				

5: Excellent 4: Good 3: Acceptable 2: Poor 1: Very Poor 0: Not attempted

General Comments:
Relevant overall comments about the paper will be included here

1.0 Utilization of Third Party Software

Our team may reference open source software to become familiar with sensor interfacing; however, we do not intend to utilize third party software for the implementation of our project except for header files and libraries provided by STM32 and those included with our chosen sensors.
The STM32 and sensor libraries being used are as follows:

Name	License	Description	Use
HAL Library	STMicroelectronics	"STM32 abstraction layer embedded software ensuring maximized portability across the STM32 portfolio" [1]	Primary Hardware Abstraction Layer for STM32L4 Microcontroller to more easily work with peripherals
SSD1306	MIT	"STM32 library for working with OLEDs based on SSD1306, SH1106, SH1107 and SSD1309, supports I2C and 4-wire SPI.." [2]	Reference library for TOLED display functionality
LoRa Library	HopeRF	"RFDK_RFM95_V2.1-"[4]	Primary Hardware Abstraction Layer for HopeRF RFM95W LoRa Radio
IMU Library	Adafruit	"Unified sensor driver for the Adafruit BNO055 orientation sensor breakout" [5]	Primary Hardware Abstraction Layer for BNO055

2.0 Description of Software Components

The Snow-weAR Goggles require three main software components to organize user sessions and interface with the primary electrical components central to the operation of the system. A Session structure will store the immediate and summary metrics of the user and provide functions to begin, end, and process a session, including calculating relevant user metrics based on information gathered by the GPS, IMU, and temperature sensors. The remaining two main software components include functions to communicate to the OLED Display and transmit and receive radio communication. Other software components include a function for monitoring battery level, functions to read and write to I2C devices, functions to read and write to the UART device, and general I/O functions for user input via push buttons.

2.1 Session Structure: a new session structure will be created when the user indicates that they wish to begin recording their performance. This structure will store user information in the following variables:

- *float start_pos[3]*: This is a three element array that will store the Latitude, Longitude, and Altitude data values collected from the GPS at the start of the session which will then be used for future metric calculations including total distance traveled.
    - init = <given by GPS>
- *float curr_pos[3]*: This is a three element array that will store the Latitude, Longitude, and Altitude data values collected from the GPS at the current collection time stamp.
    - init = start_pos
- *float prev_pos[3]*: This is a three element array that will store the Latitude, Longitude, and Altitude data values collected from the GPS at the last collection time stamp. The variable is initialized to start_pos at the start of the session.
    - init = start_pos
- *int time_start*: The start time of the session relative to system start-up.
    - init = <current time>
- *int radio_ID*: Radio ID of the user. To be used when the radio option is enabled.
    - init = <Hard-coded Goggle ID>
- *char* RFM95data:* Holds received or transmitted data from Radio Transceiver
    - init = <empty string>
- *float accel[3]*: This is a three element array that will store the acceleration data values in the x, y, and z-axis collected from the IMU.
    - init = <given by IMU>
- *float temp*: The current temperature as measured by the STM32L476VGT Temperature Sensor.
    - init = <given by Micro>
- *float velocity[3]*: The current velocity in the x, y, and z-axis as calculated by the get_metrics() function.
    - init = 0
- *float avg_temp*: The average temperature for the session.
    - init = 0
- *float avg_velocity[3]*: The average velocity in the x, y, and z-axis for the session.
    - init = 0
- *float total_distance*: The total distance covered in the session estimated by the summation of euclidean distances between all values of curr_pos collected during the session.
    - init = 0
- *float degreeInclineXZ:* The degree, incline being positive, from XZ being at equilibrium
    - init = <given by IMU>

The Session Structure also utilizes the following functions:

- *int session_start(self)*: This function will first initialize all session variables to their respective init values. If any of the peripherals do not respond, the function will return an appropriate error value. The function will then have the OLED display the initial metrics, with any unresponsive values dashed out.

- *int get_metrics(self)*: This function will contact the GPS, IMU, and temperature sensors, collect current data, and store the values in the appropriate structure variables. Velocity will be calculated by integrating the acceleration obtained from the IMU over the time period since the last data collection with assumed initial session velocity of zero. Velocity will also be calculated as the euclidean distance of prev_pos and curr_pos divided by the time since the last data collection. The user's velocity will be a pre-defined weighted average of the two velocity measurements based on estimation certainty (α, β).

$$v_{imu} = \int_{t_c}^{t_{c+1}} accel_{imu} dt \qquad v_{gps} = \frac{\sqrt{\sum_{i=0}^{2} (curr\_pos[i] - prev\_pos[i])^2}}{t_{c+1} - t_c}$$

$$v = \frac{\alpha v_{imu} + \beta v_{gps}}{\alpha + \beta}$$

  The function will also use the current values to update the variables storing summary metrics. Finally, the function will update the OLED display to show the new metric values. If any of the peripherals do not respond, the function will return an appropriate error value and any unresponsive values will be dashed out on the display.
- *void end_session(self)*: This function will display the summary statistics of the session on the OLED display and end communication with peripherals.

2.2 OLED: The following functions are necessary to update the OLED display:
- *void write_string(char* string, int x, int y)*: Writes a string at position x, y
- *void write_char(char c, int x, int y)*: Writes a character at position x, y
- *void clear_display(void)*: clears the display
- *void show_homescreen(void)*: displays the Snow-weAR homescreen
- *void zero_stats()*: Displays stats zeroed out

2.3 LoRa: The following functions are necessary to enable radio operations:
- *void rx_init(void):* checks that the rx state is stable, sets the fifo address pointer to the rx base address, determines the packet size (read from the transceiver chip), and clears any interrupts.
- *void tx_init(void):* Sets payload length, clears interrupts, and opens the TxDone interrupt. It also sets the fifo address pointer to the tx base address, and clears the rx status flag.
- *int rx_packet(void):* Returns 0 for failure and 1 for success. If packet is successfully read, global variable RxData is set to packet value.
- *int tx_packet(void):* Returns 1 for successful transmission and 0 for failure. It sends a timing interval and writes the data from the global variable *RFM95Data* to the address of the tx register for the chip. It then sets tx mode.

2.4 Battery Monitor: A global variable will store the current battery level as calculated by the Battery Babysitter module. If the battery level is below 20%, the OLED will display a battery low warning.

2.5 I2C: The following functions are necessary to communicate with I2C devices:

- *char* i2c_read(int slave_id)*: Read data from I2C slave.
- *void i2c_write(int slave_id, char* data)*: Write data to I2C slave.

2.6 UART: The following functions are necessary to communicate with the UART device:
- *char* uart_read()*: Read data from UART device.
- *parse_uart(char* data)*: Parse UART device data.

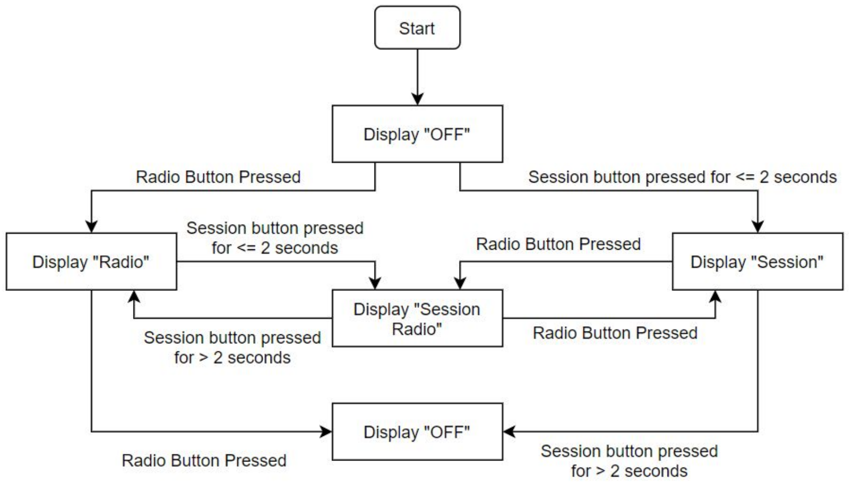
2.7 General I/O: Includes functions to process user input via push buttons.

The code structure is further explained in the Software Component Diagram in Appendix 1.

3.0 Testing Plan

3.1 OLED: Successful use of the display will greatly improve future debugging processes. The following test will ensure proper operation of the OLED display:
1. Display "Snow-weAR" at the top of the OLED display
2. Clear OLED
3. Display "0123456789" in the middle of the OLED display
4. Clear OLED
5. Display three filled squares at the bottom of the OLED display
6. Clear OLED

3.2 General I/O: Successful operation of General I/O will also be useful for future debugging processes. To ensure proper operation of General I/O, the following logic flowchart must hold:



3.3 I2C: Functional I2C communication is necessary for interfacing with the IMU and Battery Monitor sensors. The following test will ensure proper operation of the I2C interface:
1. Request and read packet from IMU when stationary.
2. Display returned values on OLED and check for expected output.

3. Request and read packet from IMU with linear acceleration in any one axis.
4. Display returned values on OLED and check for expected output.

3.4 UART: Functional UART communication is necessary for interfacing with the GPS sensor. The following test will ensure proper operation of the UART interface:

1. Read UART packet from GPS.
2. Parse data and display values on OLED display.
3. Check for expected output.

3.5 Metric Calculation: Velocity is one of the primary metrics displayed to the user. Proper velocity calculation is necessary for the success of the project. The following test will ensure proper calculation of velocity:

1. Obtain IMU and GPS data when stationary.
2. Calculate v_imu, v_gps, and their weighted average and display the values on the OLED.
3. Check for expected output.
4. Repeat 1-3 at 2 m/s and 5m/s.

3.6 Radio: Radio data transmission is another key factor in the success of the Snow-weAR Goggles. The following test will ensure proper operation of LoRa Radio communication:

1. System 1 requests GPS data from System 2.
2. System 2 sends relevant data to System 1.
3. System 1 displays the message on OLED.
4. Check for expected output.

3.7 Battery Monitor: The system should be able to monitor battery level to display to the user. The following test will ensure proper operation of the Battery Monitor:

1. Read Battery Level when fully charged.
2. Display recorded battery level on OLED.
3. Check for expected output with 10% tolerance.
4. Repeat steps 1-3 at ~50% battery level and ~20% battery level.

4.0 Sources Cited:

[1] S. T. Microelectronics, “Description of STM32F4 HAL and low-layer drivers,” *STMicroelectronics*, 2020. [Online]. Available: https://www.st.com/resource/en/user_manual/dm00105879-description-of-stm32f4-hal-and-ll-drivers-stmicroelectronics.pdf. [Accessed: 27-Sep-2020].

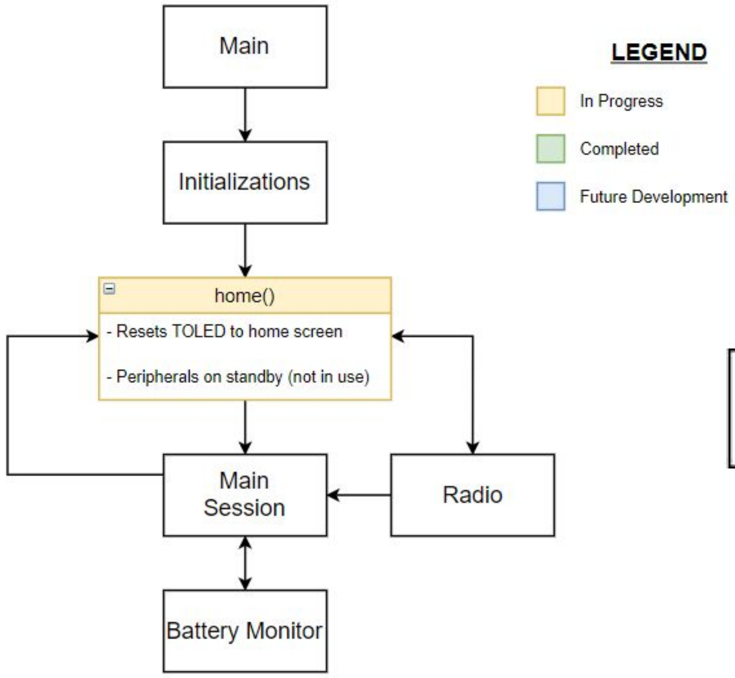
[2] A. Alekseev, “stm32-ssd1306,” *GitHub*, 11-Aug-2020. [Online]. Available: https://github.com/afiskon/stm32-ssd1306 [Accessed: 27-Sep-2020].

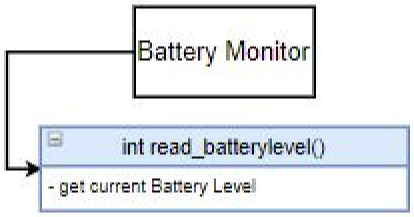
[3] R. Sellens, “Adafruit_GPS,” *GitHub*, 16-Sept-2020. [Online]. Available: https://github.com/adafruit/Adafruit_GPS. [Accessed: 27-Sep-2020].

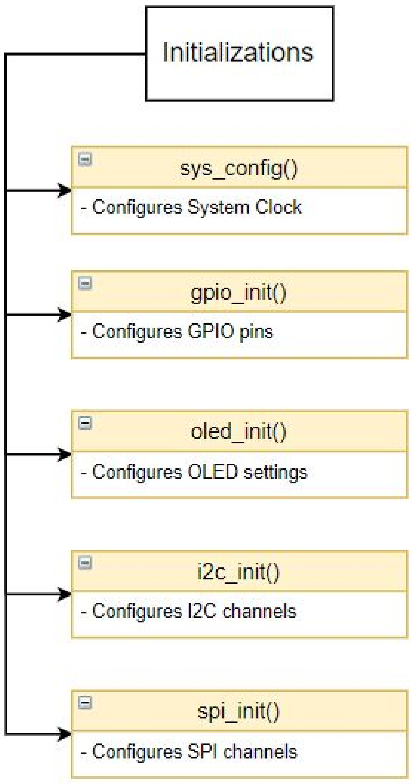
[4] HopeRF, “RFM95 ,” *HopeRF*, 21-Jul-2020. [Online]. Available: https://www.hoperf.com/modules/lora/RFM95.html. [Accessed: 6-Oct-2020].

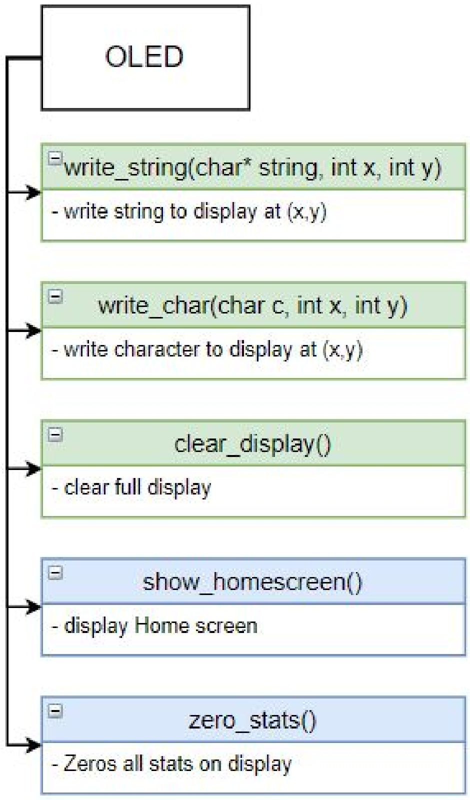
[5] Adafruit, “Adafruit_BNO055 ,” *GitHub*, 21-Jul-2020. [Online]. Available: https://github.com/adafruit/Adafruit_BNO055. [Accessed: 6-Oct-2020].

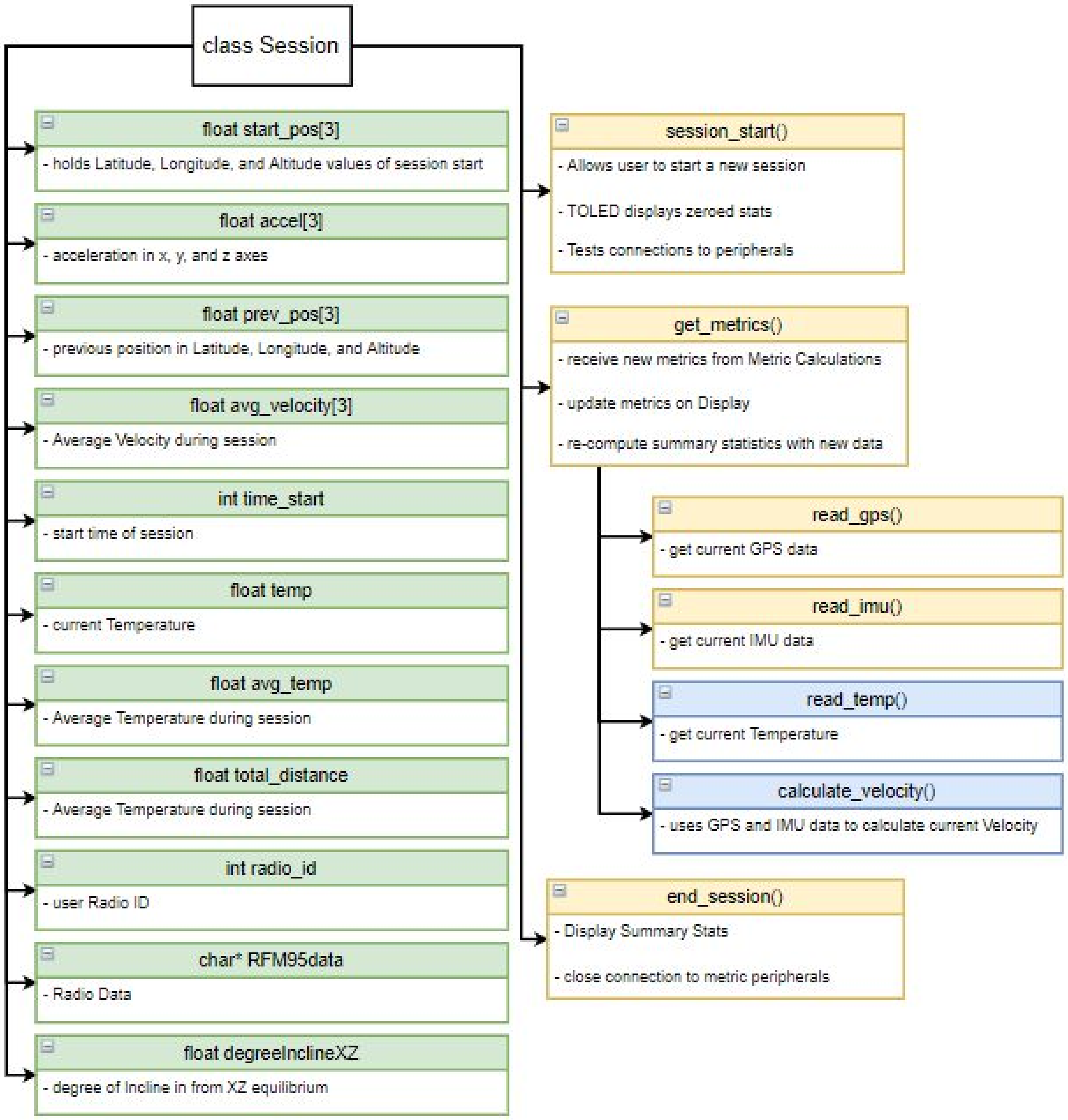
Appendix 1: Software Component Diagram

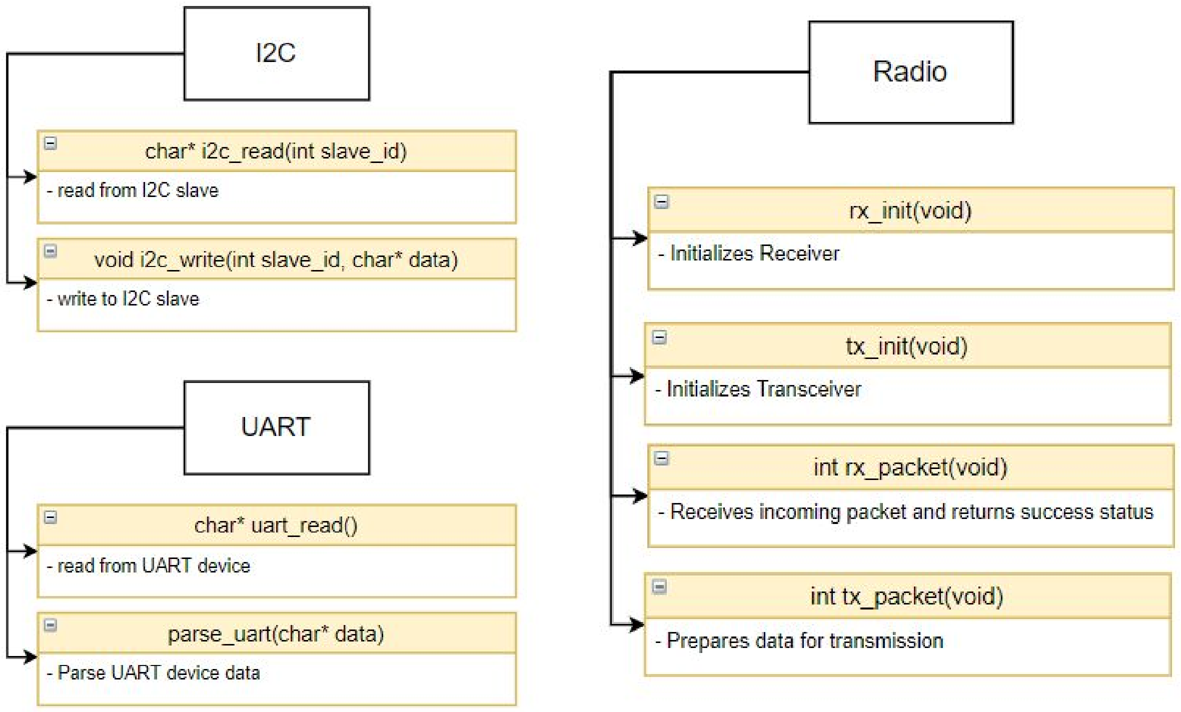





Initializations
sys_config()
- Configures System Clock
gpio_init()
- Configures GPIO pins
oled_init()
- Configures OLED settings
i2c_init()
- Configures I2C channels
spi_init()
- Configures SPI channels


OLED
write_string(char* string, int x, int y)
- write string to display at (x,y)
write_char(char c, int x, int y)
- write character to display at (x,y)
clear_display()
- clear full display
show_homescreen()
- display Home screen
zero_stats()
- Zeros all stats on display


class Session
float start_pos[3]
- holds Latitude, Longitude, and Altitude values of session start
float accel[3]
- acceleration in x, y, and z axes
float prev_pos[3]
- previous position in Latitude, Longitude, and Altitude
float avg_velocity[3]
- Average Velocity during session
int time_start
- start time of session
float temp
- current Temperature
float avg_temp
- Average Temperature during session
float total_distance
- Average Temperature during session
int radio_id
- user Radio ID
char* RFM95data
- Radio Data
float degreeInclineXZ
- degree of Incline in from XZ equilibrium
session_start()
- Allows user to start a new session
- TOLED displays zeroed stats
- Tests connections to peripherals
get_metrics()
- receive new metrics from Metric Calculations
- update metrics on Display
- re-compute summary statistics with new data
read_gps()
- get current GPS data
read_imu()
- get current IMU data
read_temp()
- get current Temperature
calculate_velocity()
- uses GPS and IMU data to calculate current Velocity
end_session()
- Display Summary Stats
- close connection to metric peripherals


I2C
char* i2c_read(int slave_id)
- read from I2C slave
void i2c_write(int slave_id, char* data)
- write to I2C slave
UART
char* uart_read()
- read from UART device
parse_uart(char* data)
- Parse UART device data
Radio
rx_init(void)
- Initializes Receiver
tx_init(void)
- Initializes Transceiver
int rx_packet(void)
- Receives incoming packet and returns success status
int tx_packet(void)
- Prepares data for transmission