LAB 5 – "ADDING FUNCTIONALITY" REPORT

Authors: Long Nguyen and Chase Arline
ECE/CSE 474, Embedded Systems
University of Washington – Dept. of Electrical and Computer Engineering

Date: 25th March 2021

TABLE OF CONTENTS

LIST OF FIGURES

1.0 ABSTRACT

The goal of this lab is to use an accelerometer in order to measure displacement, distance, and relative angle of a device. The microcontroller will have to set a global time base, draw a new screen on the display, and determine the displacement and angle of the device for the XYZ axes, as well as determine the total distance travelled. As it turns out, even with heavy filtering it does not seem possible to measure distance/displacement using the double integration method on accelerometer data. The angle is measured fine as long as it is the main acceleration is the static acceleration due to gravity.

2.0 INTRODUCTION

This stage in the project will set the global time base for the system in order to sample the accelerometer at a fast rate. Other tasks will run at a set interval, determined as a multiplier of the global time base. The display will have a new screen added that will display accelerometer measurements and processed values. The microcontroller will compute the displacement in the XYZ direction, as well as total distance in any direction, using the double integration method. The microcontroller will also calculate the angle with respect to gravity for each axis present.

3.0 DESIGN SPECIFICATION

The design of this portion of the system is defined by both the hardware and software architecture and requirements. The microcontroller will continue to interface with any I/O as it did in the previous revision, with modifications as follows. This includes the touchscreen, the simulated contactor, the HVIL, and new additions. The MMA7361 IC is the accelerometer used in this project. It is set to operate the +-1.5g range. Each XYZ acceleration value is read as an analog input through the ADC. The microcontroller will compute XYZ displacement, total distance, and angle relative to gravity for all axes. This will be displayed on a new screen for the display dedicated to the accelerometer data. Filtering methods will be implemented in order to more accurately relate accelerometer data with real data. The filtering methods used in this project are: second order the Simpson method of polynomial integration, circular/rolling buffer low pass filter, velocity stop for extended periods of near-zero acceleration, and a general deadband for drifting acceleration values. The scheduler will continue to be dynamic and executes on a 5ms time-base. This time-base is set by a hardware timer interrupt inside of the microcontroller. Certain sections of the code will be deemed critical, and precautions will be taken to ensure that interrupts do not interfere with these processes.

4.0 SOFTWARE IMPLEMENTATION

This project uses the Arduino Mega 2560 microcontroller to interface the TFT display and several GPIO pins. At this final phase of the project, this BMS system is also using a single accelerometer sensor (MMA7631) to measure the x,y and z acceleration with respect to gravity and calculate the x, y, z position and angle with respect to gravity. The accelerometer is supplied by a 5V power and the x, y, z accelerometer values are read via analog pin A12, A11 and A10 respectively. The high-level picture of this system is represented in **Figure 1** below.

The main architecture of the software revolves around the scheduler. The task queue's underlying data structure and implementation are the same with the previous lab. However, since the accelerometer task needs to be updated more frequently compared to other tasks, so the system's time base is changed to be 5ms, which is 200Hz. All the tasks will be added and removed dynamically from the task queue (which is a linked list) based on their execution rate. All tasks in the linked list will be added in the following order: accelerometerTask → measurementTask → alarmTask → socTask → touchScreenTask → contactorTask → data loggingTask → remoteTerminalTask. For simplicity, since the accelerometer is executed at the rate of the time base rate, then the accelerometerTask will not be removed from the task queue, while the other tasks will be added before its execution time and be removed right after finish executing. In order to create that time-based system, there will be a hardware timer interrupt service which sets a time base for the main loop to be 5ms. The implementation details of the Scheduler task, timerISR task and main loop will be represented in the Appendix A.

The new and big part of this final phase of the BMS project is the implementation of the accelerometer task. The accelerometer task is using a single accelerometer sensor (MMA7631) to get the x, y, z accelerations. The acceleration values are read in via three analog inputs described above. The acceleration values are used to calculate the x, y, z's displacements, total distance and x, y and z's angles with respect to the gravity. In order to read the acceleration values correctly, there is a dynamic calibration subtask that runs once time when the system starts up. The calibration subtask will read 500 samples for each x, y and z values and take the average errors as calibration values for each x, y, z's acceleration values, which will be minus from later read-in calibration values. Also, we calculate the max drift value of x, y, z acceleration values by sampling 40000 values when the program startup. This value will be used to determine the dead-zone acceleration values. The accelerometer task will be passed down from Scheduler the elements: AccelerometerValue struct for each axis (x, y and z), a float pointer pointing to totalDistance, float value timeInMs, totalVelocity, float value of lastTotalVelocity, float value of secondLastVelocity, float value for lastTotalAccel and float value for secondTotalAccel. The reason why the timeInMs, totalVelocity, lastTotalVelocity, secondLastVelocity, lastTotalAccel and secondTotalAccel are not pointer since they're not need to share with other tasks and are just used for reducing

the errors of calculating values. The AccelerometerValue struct encapsulates the following elements of a single axis:

- pointer to float value of distance
- pointer to float value of angle
- array of the previous accelerations: this array is used to calculate the rolling average acceleration value of the axis.
- float value of velocity
- pin: a byte value of analog pin that we read acceleration value from
- integer value of movingAverages: this value is used to index through the array of previous accelerations and update those previous values.
- rollingAccel: the float value stores the rolling average acceleration
- lastRooling: the float value to store the most recently rolling average acceleration
- secondLastRolling: the float value to store the second most recent rolling average acceleration
- lastVelocity: the float value represents the most recent previous velocity
- secondLastVelocity: the float value represents the second most recent velocity.

The way that AccelerometerValue struct is represented clearly in the **Figure 3** - Class Diagram. The AccelerometerValue objects are then passed to the getMeasurement function to update the acceleration values. The getMeasurement calculates the average rolling acceleration by taking the average of current with two previous accelerations stored in accelerationArray. After getting the acceleration of the x, y and z axis, those values are used to calculate the magnitude of accelerations. There are two versions of magnitude of accelerations used here:

- the first version is calculated by the acceleration values of all three axis
- the second version is calculated by the acceleration values of only the x and y axis. The reason for omitting the acceleration of the z axis is to reduce the error in calculating the total traveled distance on the flat surface while not moving along the z-axis.

The rolling average acceleration of each axis with the first version of magnitude of accelerations is used to calculate angle of that axis with respect to the gravity when rotating about the x or y axis. The rolling average acceleration of the x and y axis (with dt = (currentMillis - timeInMs) value and the most recent rolling accelerations and two most recent velocities) are also passed to the updateVelocity function to calculate the velocity of moving along those axises. The updateVelocity uses the the Simpson method of polynomial integration to calculate the approximate velocity of moving along the axis. Also, the updateVelocity function also uses the deadband zone of 0.02 to eliminate the minor errors in the calculation in order to have a better approximation and detect when not moving. The calculated velocity along x and y axis will be then passed to updateDistance function to calculate approximate displacement along that axis. The updateDistance is also used the the Simpson method of polynomial integration to approximate the displacement along that axis. On the other hand, the total traveled distance is also calculated in the same way but the second version of magnitude of accelerations is passed down to updateVelocity and updateDistance instead of the rolling average acceleration along a specific axis. The way that data is passed down from analog ports to the accelerometer task and to display is shown

clearly in the **Figure 4** - DataFlow Diagram. The details of the above function is shown in the pseudo code in Appendix B.

Most of the touch screen implementation is the same as the previous lab with some minor changes to accommodate the display for the accelerometer screen. The part that is changed is in the PrintedData struct. In the previous lab, the PrintedData struct has a single volatile float value to store the old value and a single float pointer to point to current volatile value. However, the new version of PrintedData struct includes an array of volatile float values to store old values (instead of a single volatile value) and an array of float pointers point to volatile float values to store the current values instead of a single pointer as before. This new version of PrintedData struct to deal with multiple datas of the same category to be printed on the TFT screen. For instance, the value of x,y and z distance value will be encapsulated inside a single PrintedData struct for facilitating the printing task. Similarly, with angle values for x, y and z, they're also encapsulated in a single PrintedData struct. And the rest of implementations for touchScreenTask are the same as previous labs. This update can be view in **Figure 3**.

For the user interface, the final BMS will have a new added screen which is an accelerometer screen to let the users monitor the x, y, z's distances, angles and total traveled distance. The way that this new screen is used is specified clearly in the **Figure 22** (use case diagram for accelerometer screen). Also, the design of the new screen is shown in the **Figure 23** - Front Panel Design for Accelerometer Screen.

5.0 TEST PLAN

The test plan includes three parts:

1. Requirements
2. Test coverage
3. Test cases

5.1 Requirements

The task queue scheduler will run off of a 5 millisecond time-base.
The accelerometer will have an accuracy of +/- 1cm over a distance travelled of 5cm over a 5 second interval.
The angle of the accelerometer will be relative to the vector of gravity.
Acceleration values from the accelerometer will be scaled to it's respective units:

+-1.5g with a sensitivity between 740 to 860, averaged at 800 mv/g

XYZ displacement, total distance, and XYZ angle w.r.t. gravity will be calculated.

The display will now have an accelerometer screen which prints xyz displacement, total distance, and xyz angles.

5.2 Test Coverage

The timer interrupt will trigger every 5 milliseconds, which will set a flag for the task queue scheduler in the main loop. This ensures a time base of 5 milliseconds.
The total distance will be accurate to +/- 1cm after travelling a distance of 5cm over 5 seconds. This will be printed on the display screen as "total dist: ".
The XYZ displacement will be printed on the screen as "XYZ dist: _,_,_".
The XYZ angles will be printed on the screen as "XYZ Angle: _,_,_".
The angles will be calculated with respect to the gravity vector. The gravity vector is measured by assuming at startup that the z axis is in parallel with the gravity vector.
The ADC scaling for acceleration will use 800mv/g sensitivity, and it will be calculated as follows:

(ADC_READING+CALIBRATION_VALUE)/675.0 * 3.3/0.8

The idea is as follows: the calibration is applied to the direct ADC value, it is then converted from a range of (0-5v -> 1023 max) to (0-3.3v -> 675 max). After this, the percentage of ADC maximum is converted to voltage from 3.3v to 0.8v. Finally, the sensitivity of 800mv/g is applied. This will be checked to be 0,0,1g while sitting still at startup (gravity affecting only z axis).
The XYZ angles at startup should be 90 degrees, 90 degrees, 0 degrees because the gravity vector is aligned with the z axis and is 90 degrees from both X and Y.

5.3 Test Cases

The timer interrupt will trigger every 5ms and this timer will be printed to the serial console.
The total distance will be printed on the display so the test coverage will be checked visually. At startup it should be 0.
The XYZ displacement will also be printed on the display so the test coverage will be checked visually. At startup it should be 0,0,0.
The XYZ angle will be printed on the display so the test coverage will be checked visually. At startup it should be 90,90, 0.
The ADC → gravity formula will be checked by printing values to the serial console. At standstill after being calibrated, it should display 0,0,1g for the acceleration values. This will be checked visually in the serial console.
The total displacement should be +/- 1cm after travelling a distance of 5cm in 5 seconds. This will be checked using the printed value on the accelerometer screen.

6.0 PRESENTATION, DISCUSSION, AND ANALYSIS OF THE RESULTS

Our design is able to calculate the angle of the device w.r.t gravity perfectly fine. The XYZ displacement, and total distance, have issues even after heavy filtering. Our error rate varies each time we move 5cm in 5 seconds, but it generally is about 4cm. If we move 5cm in 1 second, it is closer to about an error of 2cm average. This is because we have to filter out lower acceleration values to the drift caused by the accelerometer. The XYZ displacement is not accurate either, mostly due to the fact we are using an accelerometer and we have to ignore low acceleration values due to drift. This is what occurs when moving the device forward:

Medium positive acceleration on axis, low positive acceleration on axis (just to counteract friction/air resistance), then high negative acceleration to stop at the desired position. If you measured velocity directly, it would always be positive. Measuring acceleration gives a high negative velocity at the end. Normally it would cancel out the positive velocity from low/medium acceleration, but the device is not accurate enough to work as such. So moving in a positive direction can sometimes result in an overall negative displacement due to this.

This is the main issue in our displacement filtering, but it does not affect the total distance as much considering the total acceleration magnitude is always positive. So overall, even with heavy filtering, the drift ends up causing issues because inherently it ruins the sensitivity of the device. If we assume a maximum 0.02g drift (which is realistic according to our measurements), the device will then ignore any acceleration below 0.02*9.8*100 cm/s^2, or 20cm/s^2. This is a very high acceleration for our device, so moving 5cm/5seconds doesn't work because you actually need a higher sensitivity when moving slower for an accelerometer. It seems counterintuitive, but an accelerometer with small drift actually does better when calculating displacement at higher acceleration/velocity than it does at low acceleration/velocity.

6.1 Analysis of Any Resolved Errors

We had to add a new area to the touch screen, but the way we set up our display driver didn't allow us to print arrays of values. Instead of using "magic numbers" to determine where to place three distinct printed values per array, we made our code more dynamic so now it is able to print arrays of values dynamically onto the screen using a single print statement and struct. Instead of doing everything statically, we made a few small changes to our PrintedData struct and in only about 30 minutes we were now able to print arrays of values onto the screen. This was of great benefit to us as it actually allows our display to run faster because we only print values once, instead of making three distinct tft.print calls per array.

6.2 Analysis of Any Unresolved Errors

This section will cover the amount of filtering needed to even get the non-working result that we currently have.
It includes:

Our sampling of the accelerometer values was done every 5ms.

All integration is done using second order integration using the Simpson method of polynomial integration. This helps improve the accuracy of the integration which is where a large amount of error lies.

A circular buffer was originally used with the idea to filter out high frequency spikes. This overall concept also wouldn't actually improve the integration either way because the integral of the average is the same as the integral of each individual sampled piece. The only reason we still use it is for the filtering technique described below.

A "dead stop" velocity spot is used. Whenever the acceleration has been low enough for the last X number of samples, it can be assumed that the velocity is zero. This removes any long-term drift caused on the device. This is currently the only reason the circular buffer is removed because it doesn't provide any actual advantages otherwise.

Calibration values are dynamically determined at startup. The device takes 500 samples for each XYZ value over a 2.5 second period. These are then averaged to find the value that makes XYZ equal to 0,0,1g at stand still. This works accurately and has been verified.

Maximum drift values are calculated at startup after calibration. 20000 samples are taken and the maximum drift value for standstill is stored. This is then used to determine the general dead band zone for the accelerometer, or values that are counted as potential drift.

I even took the step to high pass filter the signals in real life by using extra 0.1uf and 10uf bypass capacitors on the board. This decreased the maximum drift from 0.026g to about 0.014g average. This still was not enough of a decrease though.

Even with this extensive filtering and probably over just testing the filtering methods, we still cannot get the accelerometer to be anywhere near accurate to the real-life values. It is just not possible when considering how high the maximum drift values are. Even a 0.014g max drift means we can only include 0.015g acceleration and higher. This means any acceleration under 15cm/s^2 cannot be used. Assuming we did 5cm over 5 seconds using a triangle shaped velocity, that gives a height of the triangle of 2 cm/s. This means our static accelerations in that period are (2cm/s)/(2.5s) -> 0.8cm/s^2 for the two static accelerations. Clearly our real acceleration will not be modeled after a perfect triangle and will most likely be an upside-down trapezoidal shape instead. These acceleration values will still be nowhere near 15cm/s^2 as shown by the triangle example. It does not seem like this task is possible by only using the Arduino Mega R3 and the MMA7361 accelerometer.

7.0 QUESTIONS

The frequency of operation of our accelerometer task is 200hz, it has a period of 5 milliseconds.

The accelerometer has 800mv/g sensitivity. Our ADC is 1024/5 -> 204.8 bits/V.

This gives us an overall resolution through the ADC of 163.84 bits/g, or 0.0061 g/bit.

Our error rate varies a lot over 5cm travelled. Are we travelling fast? We might read some values with an error of +/-3cm. Are we moving very slow? Then we will read zero because the acceleration values are counted as being in the "drift" zone. Overall, our readings are not consistent because the accelerometer readings themselves are not very accurate even after very heavy filtering. There is no way to get an "average" error rate. No two runs are the same, but I would assume our error rate is about +/-2cm out of 5cm travelled at a medium/fast velocity.

8.0 CONCLUSION

In conclusion our program does not perform to the expectations given in the specification. Even after including an extraneous amount of filtering, it is deemed that the accelerometer simply cannot be used for this purpose. Any values when moving are too close to the natural "drift" values that the accelerometer produces. This makes it impossible to tell what real movement is, and what is random drift from the accelerometer. The angle calculations work perfectly fine, as we are able to determine the static angle with respect to gravity from startup, and this is reported easily onto the display. It can be easily seen why nobody in industry uses accelerometers to measure distance, as the performance is just not there.

9.0 CONTRIBUTIONS

We both worked equally on this project. Almost all of the time spent working on this project we were in a zoom call, so we were both providing the same amount of input.

10.0 APPENDICES

10.1 Pseudocode

A. The following is the pseudo code for Scheduler Task:

```
Scheduler:
    for each task in linkedlist:
        executes task
```

```
timerISR:
    timerFlag = true

loop:
    if (timerFlag):
        if 100ms cycle:
            insert measurementTask, alarmTask, socTask and contactorTask
        if 1s cycle:
            insert touchScreenTask and remoteTerminalTask
        if 5s cycle:
            insert dataLoggingTask
        Scheduler()
        timerFlage = false
        if 100ms cycle:
            remove measurementTask, alarmTask, socTask and contactorTask
        if 1s cycle:
            remove touchScreenTask and remoteTerminalTask
        if 5s cycle:
            remove dataLoggingTask
```

B. The following is the pseudo code for Alarm Task:

```
getMeasurement(axis, calibrationValue):
    read-in 40 samples of the analog value via axis.pin
    take average of 40 samples of read-in analog values
    update the second last rolling average acceleration value
    update the last rolling average acceleration value
    compute rolling average of accelerations
    return the rolling average acceleration

getDegrees(acceleration, accelerationMagnitude):
    calculate arccos(acceleration/accelerationMagnitude)
    convert the radian angle value to degree angle
    return the angle in degree

calibration(xPin, yPin, zPin):
    read-in 500 samples and take the average error over that 500 readings.
    read-in 40000 samples and calculate the maxDrift for each x, y, z acceleration
    values.

simpsonIntegration(secondLast value, last value, current value, dt):
    return dt*(secondLastValue + 4*lastValue + currentValue)/6

updateVelocity(secondLastV, lastV, currentV, secondLastRolling, lastRolling,
                rollingAccel, dt, drift):
    secondLastV ← lastV
    lastV ← current
    if rollingAccel > drift or rollingAccel < -drift:
        currentV  += 9.8*100*(Simpson integration over secondLastRolling,
```

```
        lastRolling, rollingAccel and dt)
    if -drift < rollingAccel < drift:
        set currentV to 0.

updateDistance(distance, secondLastVelocity,  lastVelocity, velocity, dt):
    distance += Simpson integration over secondLastVelocity, secondLastVelocity,
    velocity and dt
```

10.2 Code File Names

StarterFile.ino: file that the program starts in. This file includes the startUpTask and two ISR() function.
StarterFile.h: header file for StarterFile.ino
Alarm.c: code for the alarm task
Alarm.h: header file for alarm.c
Contactor.ino: code for the contactor task
Contactor.h: header file for Contactor.c
Measurement.c: code file for the measurement task
Measurement.h: header file for Measurement.c
Soc.c: code file for the state of charge task
Soc.h: header file for Soc.c
Scheduler.c: code for the Scheduler task
Scheduler.h: the header file for Scheduler.c
TaskControlBlock.h: header file defining TaskControlBlock struct
TouchScreenTask.ino: code file for the touch screen task
TouchScreenTask.h: header file for TouchScreenTask.ino
DataLogging.h: header file for the data logging task
DataLogging.ino: code file for the data logging task
RemoteTerminal.h: header file for the remote termial task
RemoteTerminal.ino: code file for the data remote terminal task
Accelerometer.h: header file for accelerometer task
Accelerometer.ino: code file for accelerometer task

10.3 Figures

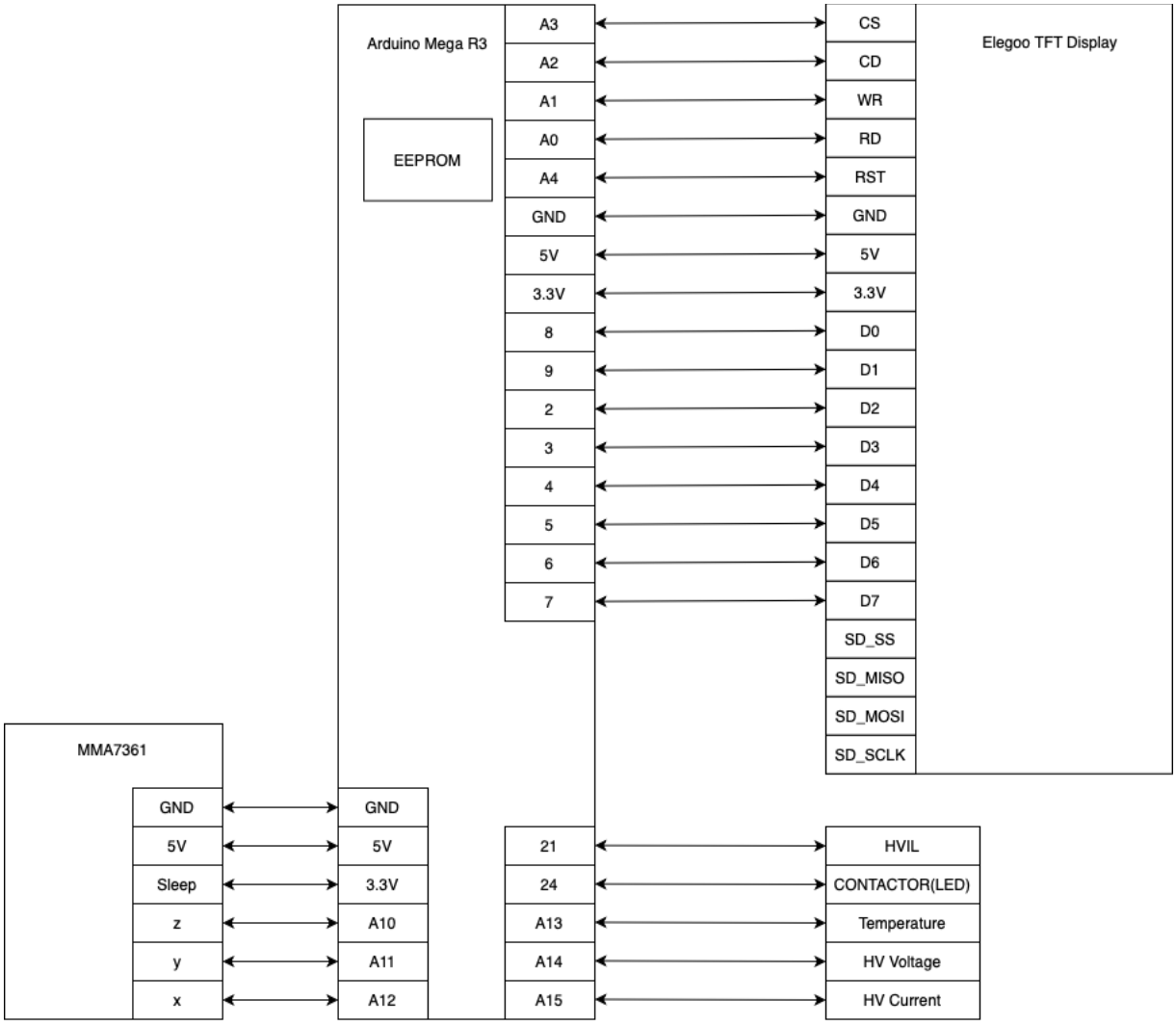


Figure 1. System Block Diagram - showing the ATMega input and output ports (and port numbers) labeled per I/O component

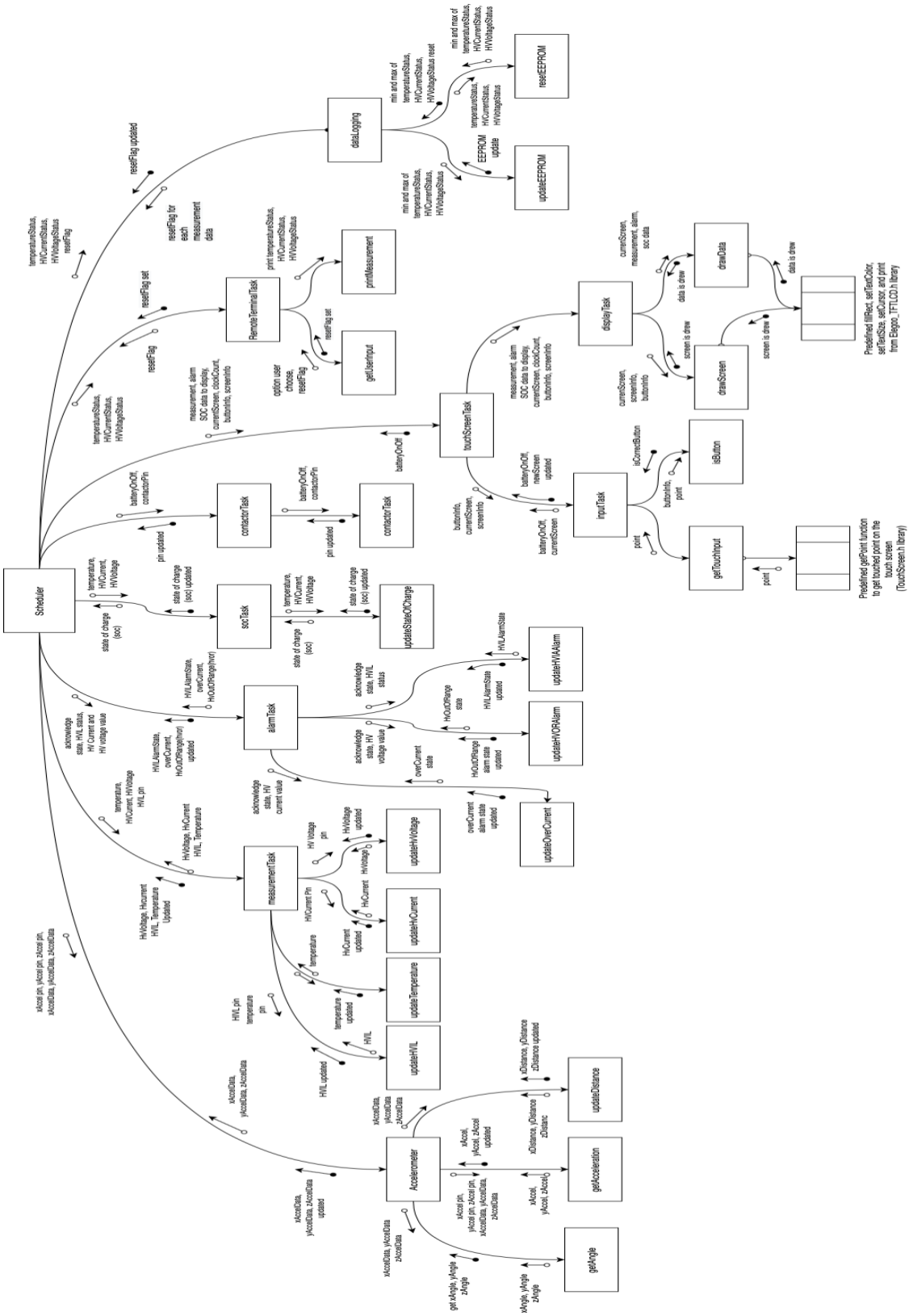


Figure 2. Structure Diagram - showing functional decomposition of tasks within the System Controller

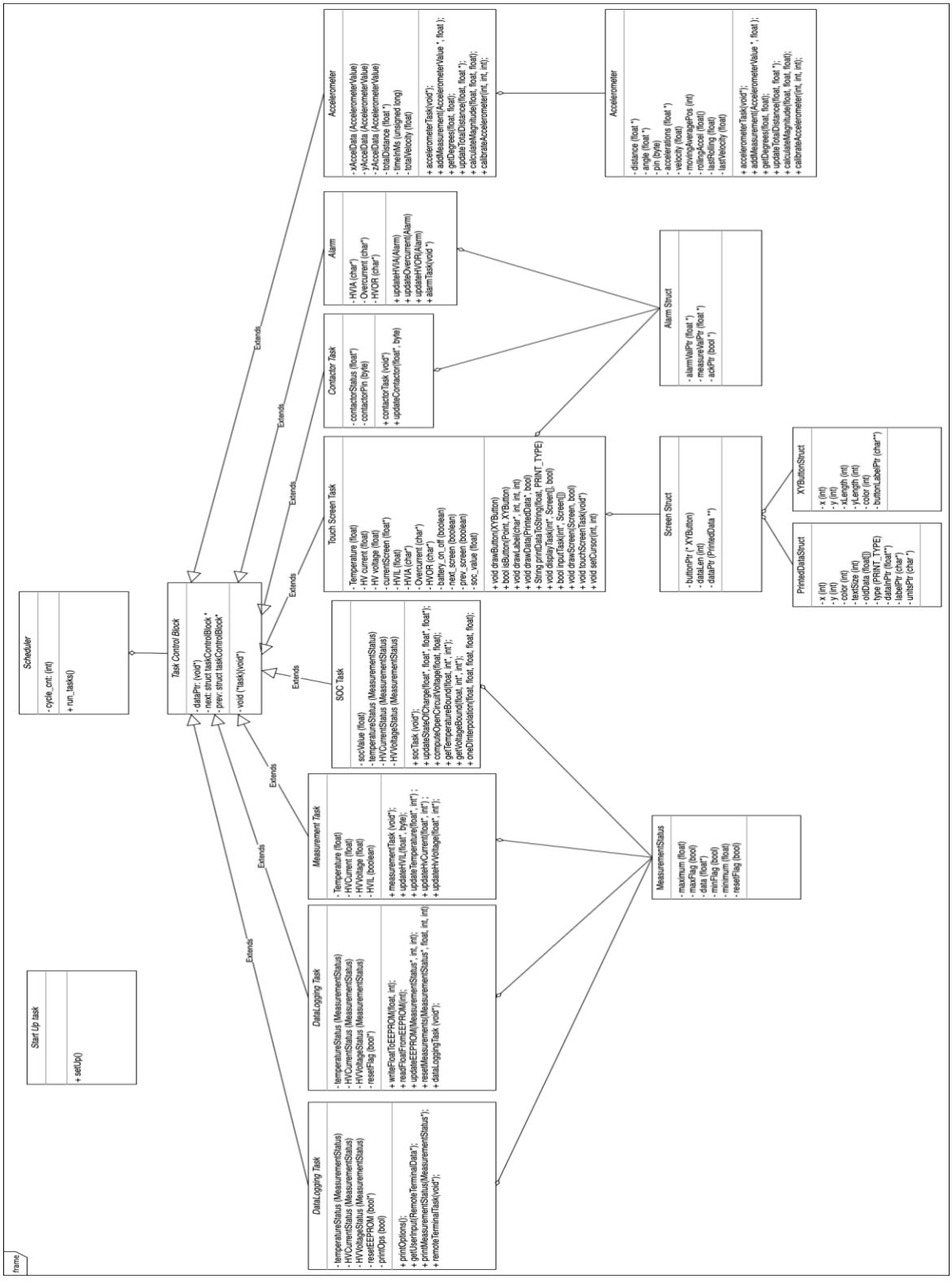


Figure 3. Class diagram - showing the structure of the tasks within the System Controller as reflected in the Structure Diagram.

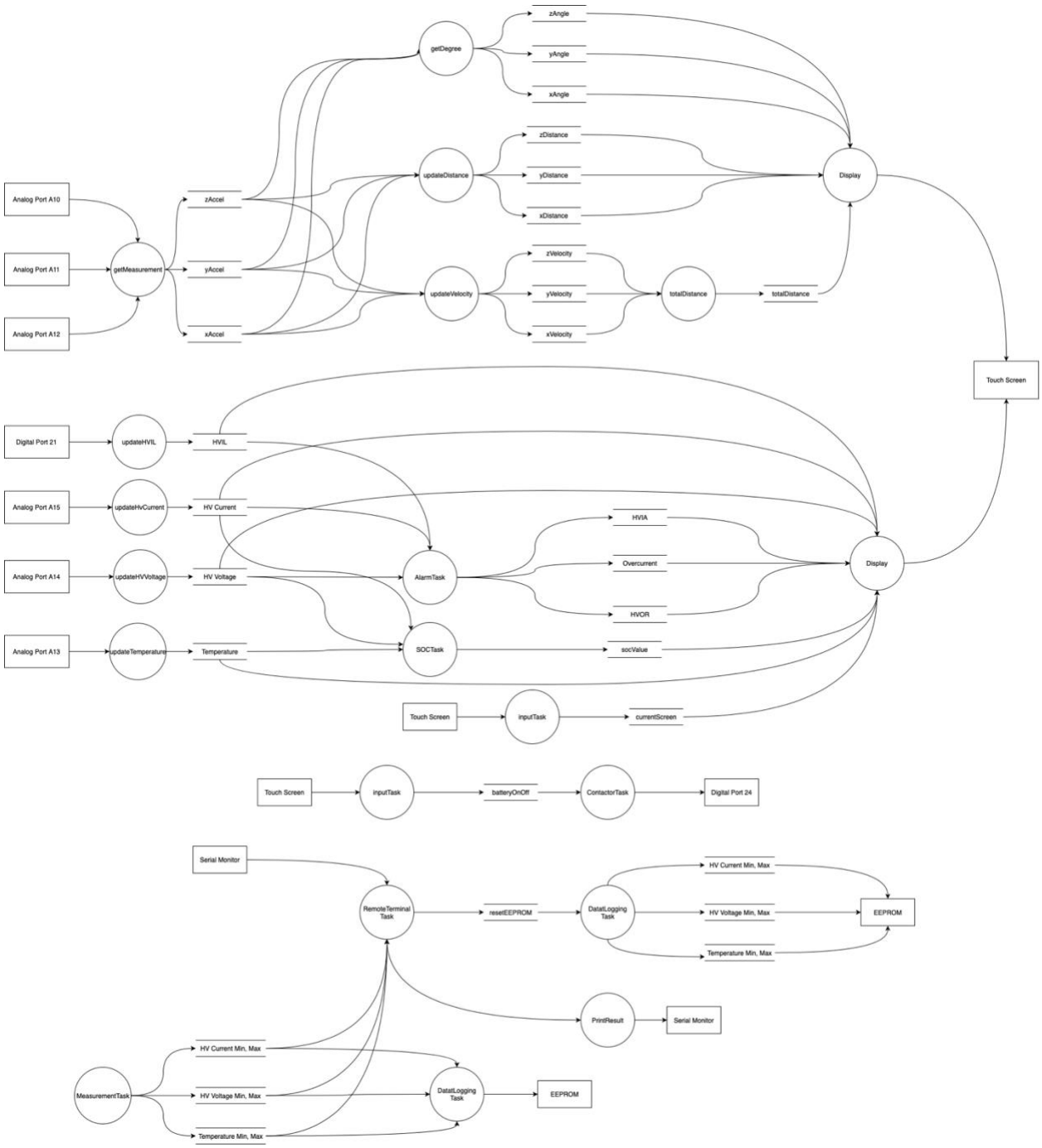


Figure 4. Data flow diagrams - shows data flow for inputs/outputs

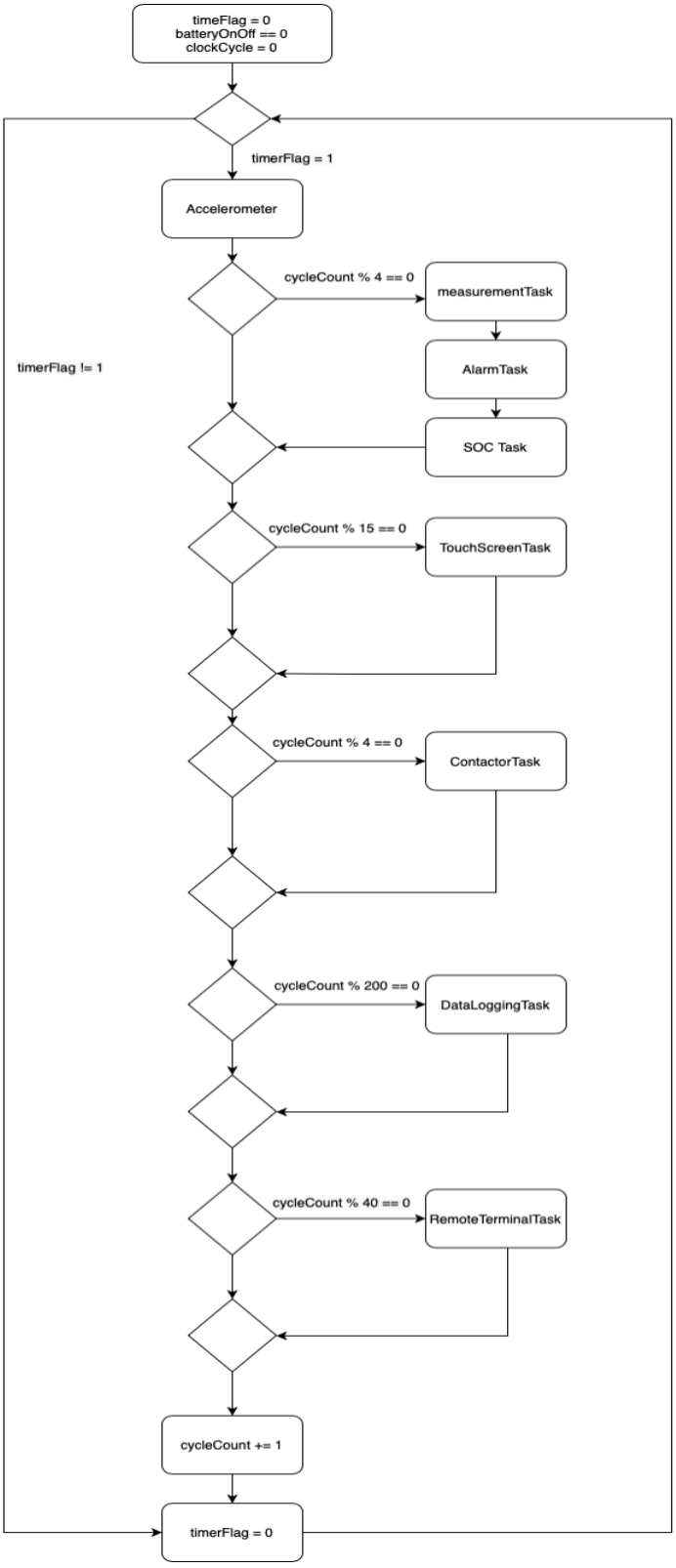


Figure 5. Activity Diagram - shows the System Controller's dynamic behavior from the initial entry in the loop() function

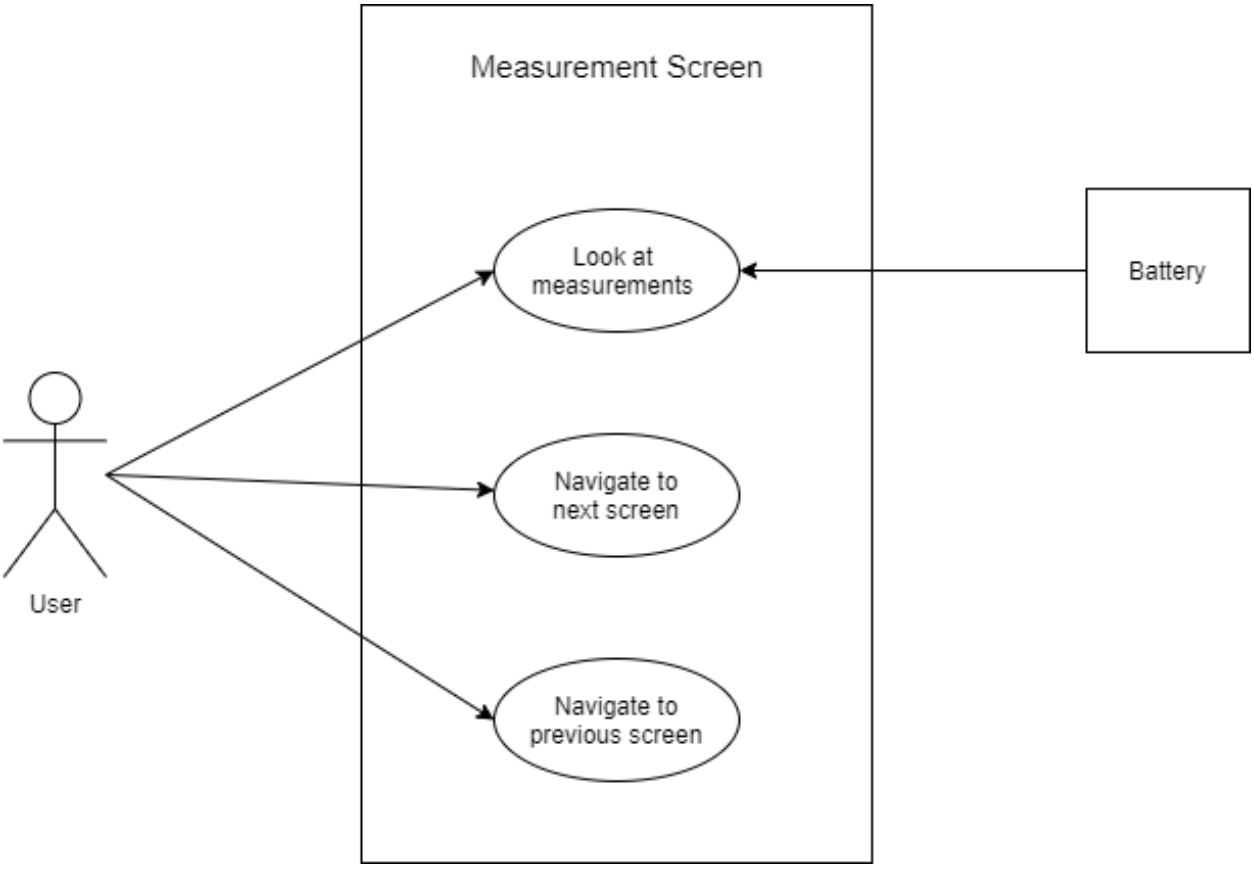


Figure 6. Use Case Diagram for Measurement Screen

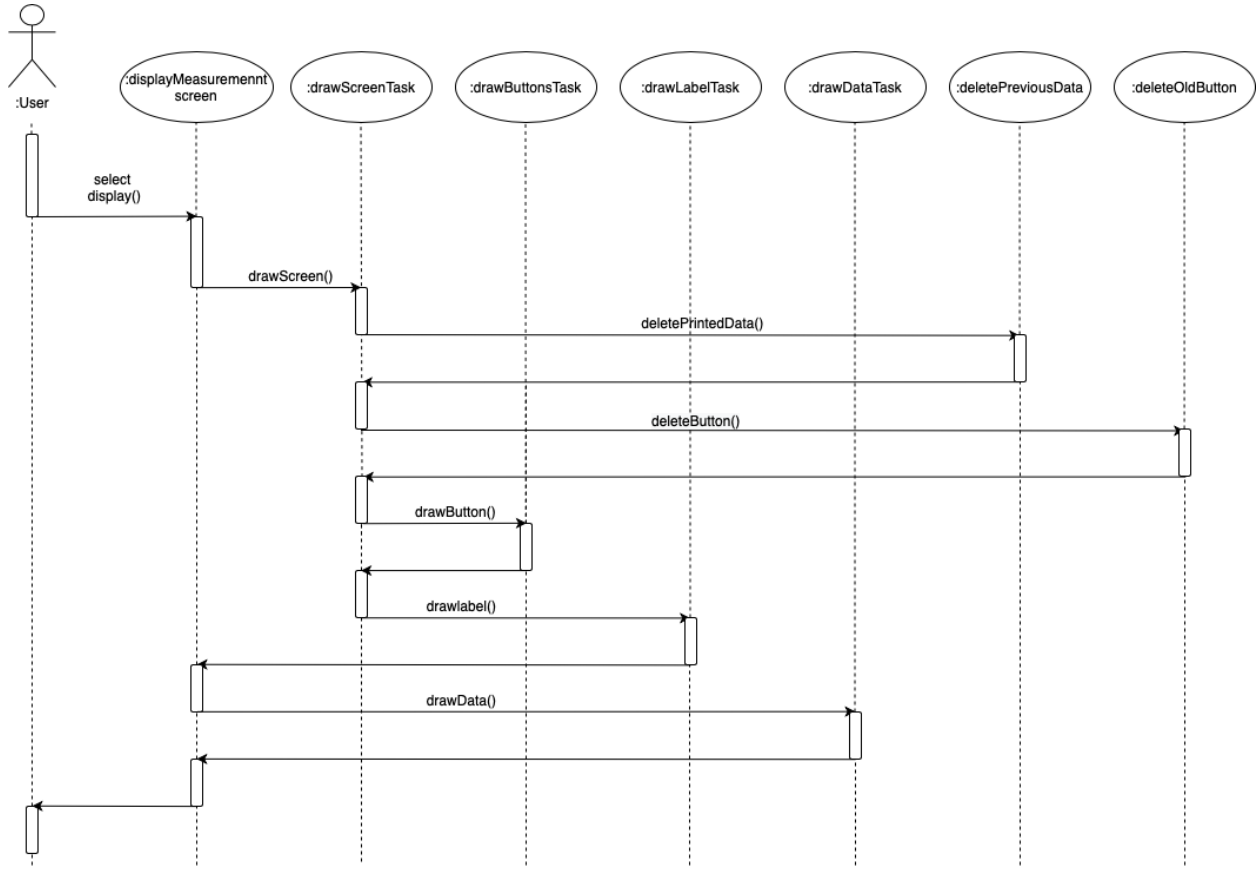


Figure 7. Sequence Diagram for Measurement Screen

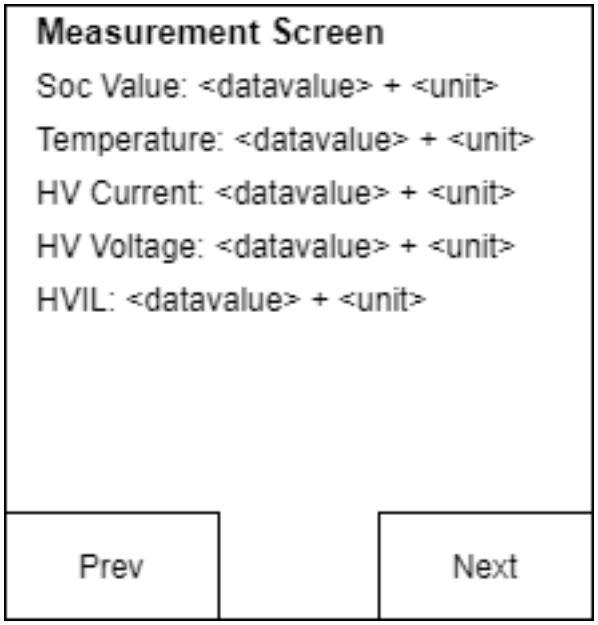


Figure 8. Front Panel Design for Measurement Screen

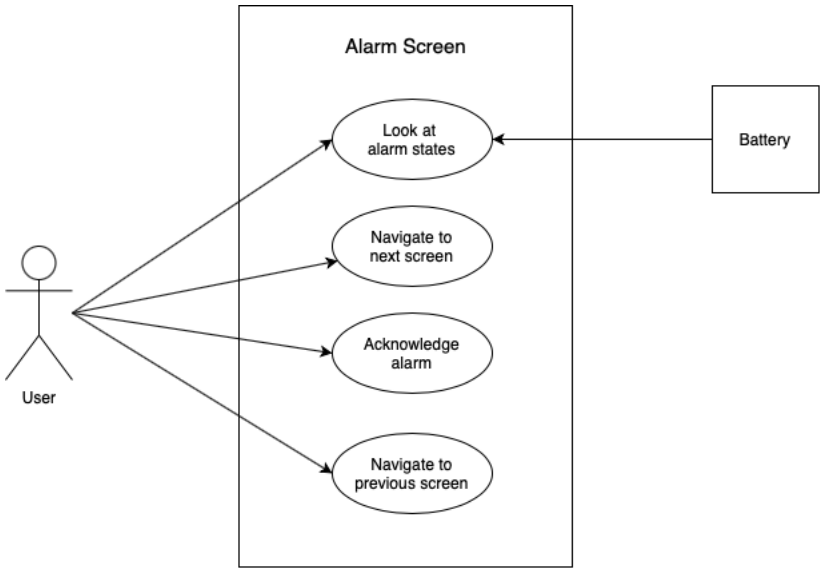


Figure 9. Use Case Diagram for Alarm Screen

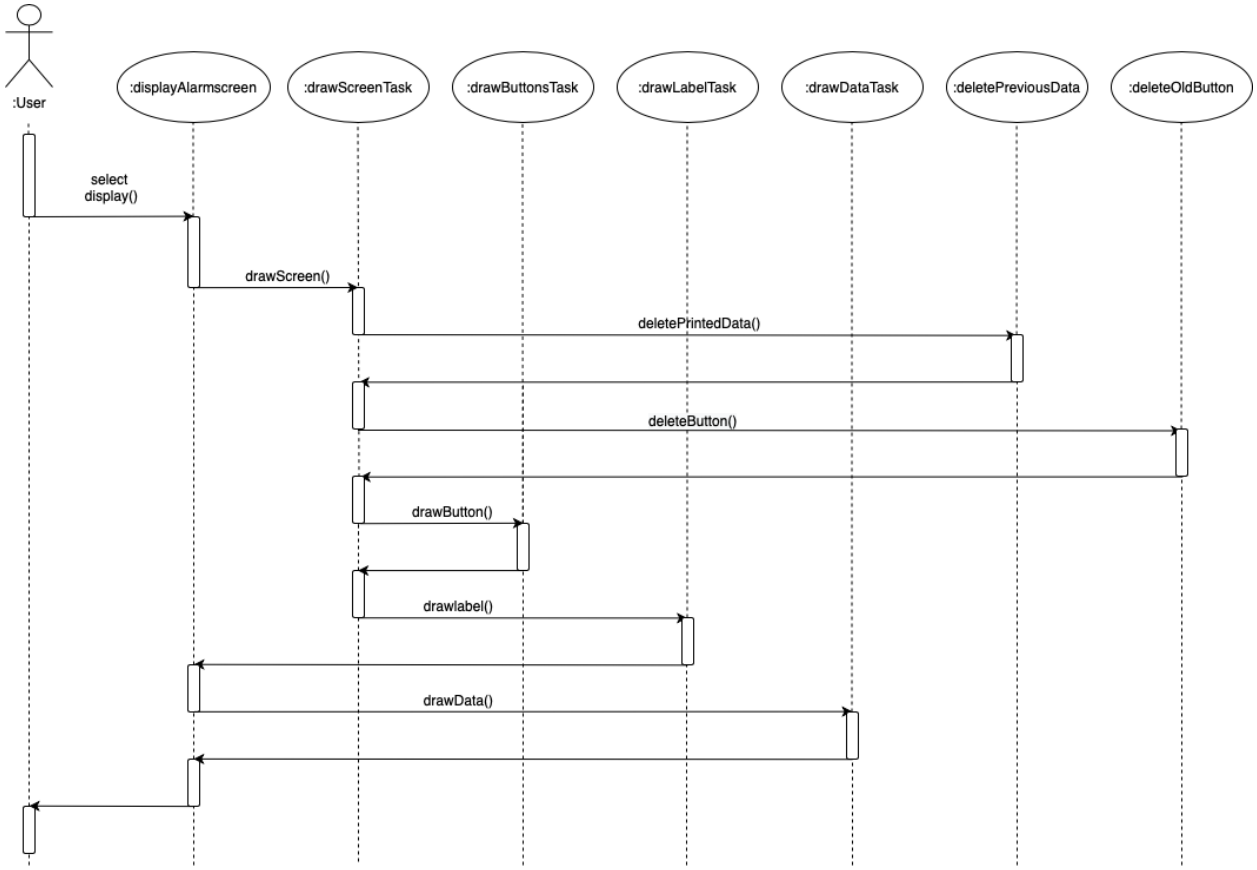
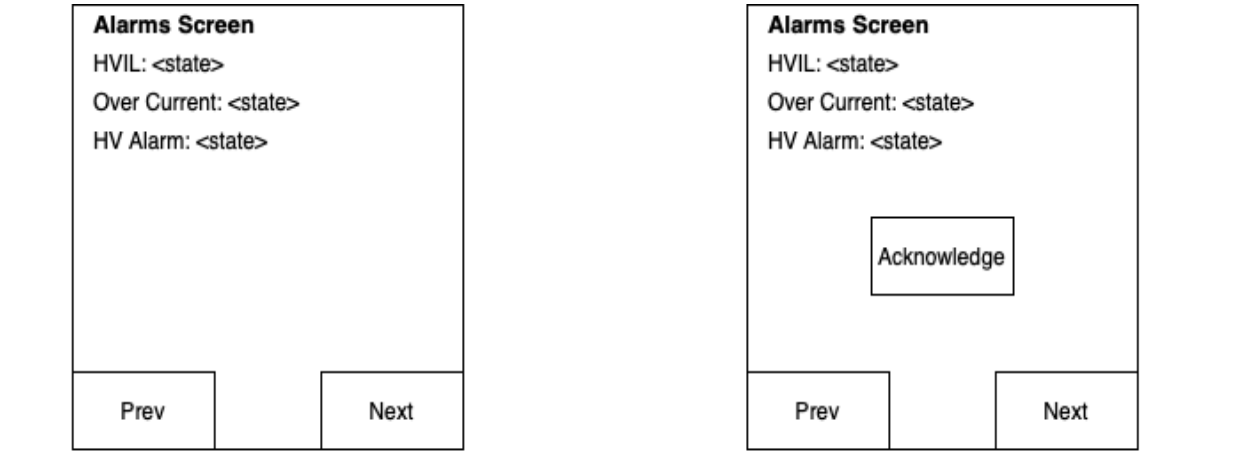


Figure 10. Sequence Diagram for Alarm Screen



14.a: Alarm Screen when no alarm active **14.b: Alarm Screen when there is an active alarm**

Figure 11. Front Panel Design for Alarm Screen

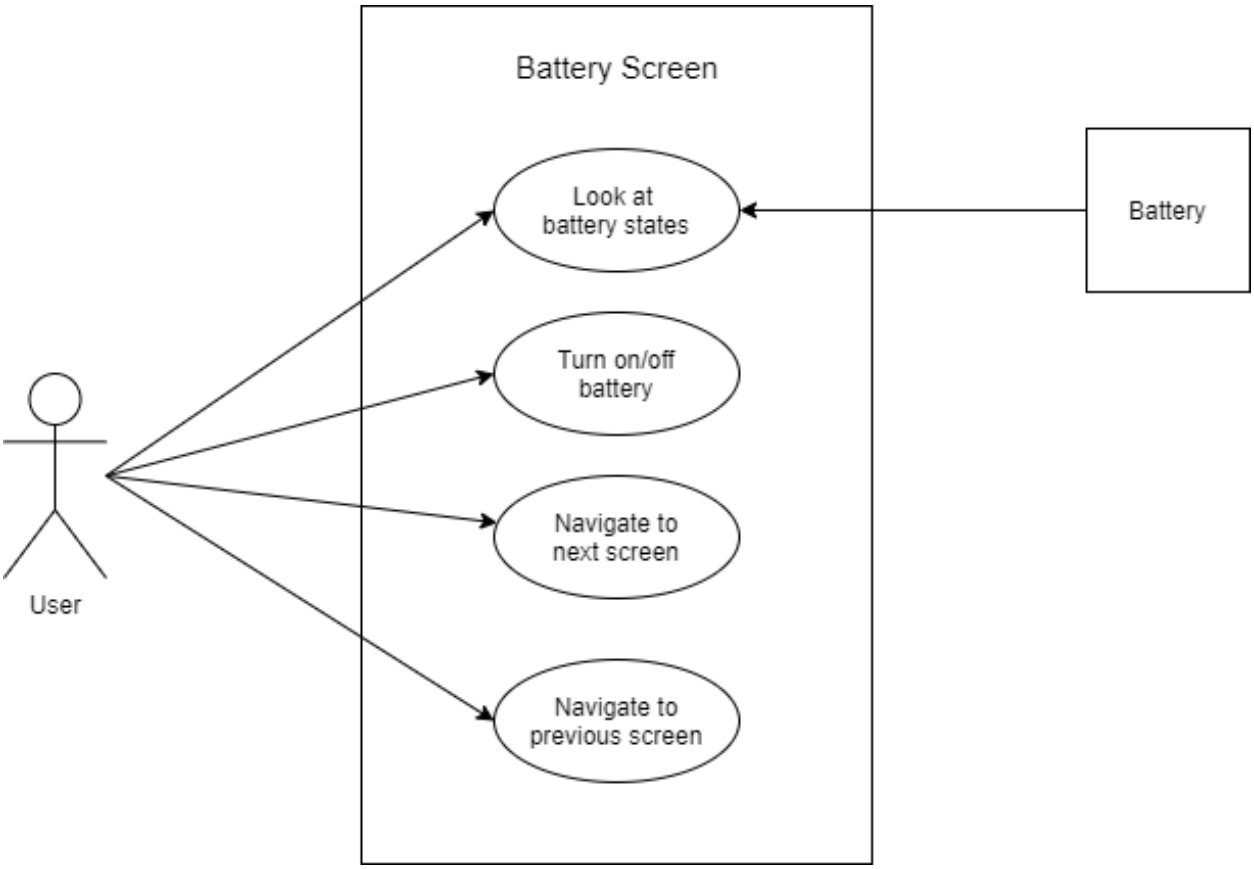


Figure 12. Use Case Diagram for Battery Screen

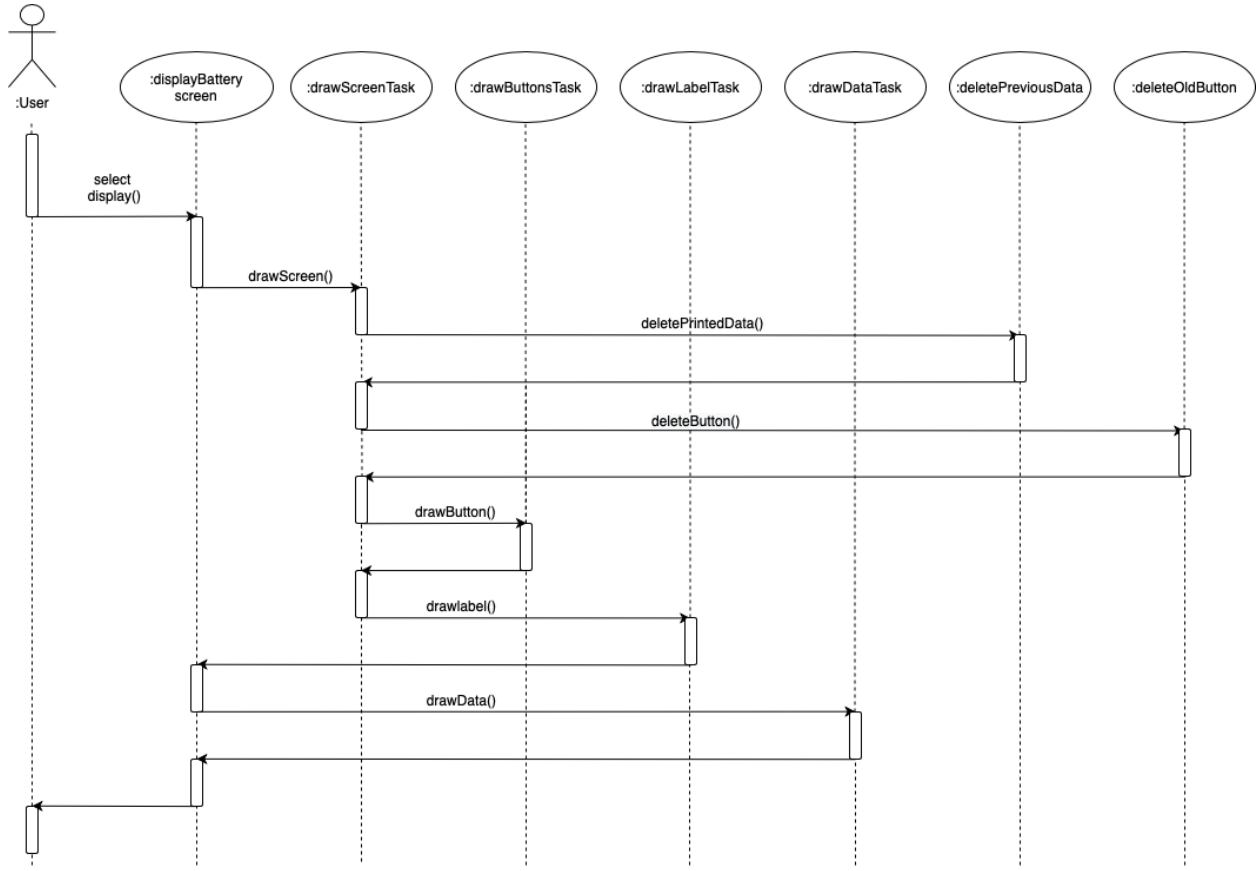


Figure 13. Sequence Diagram for Battery Screen

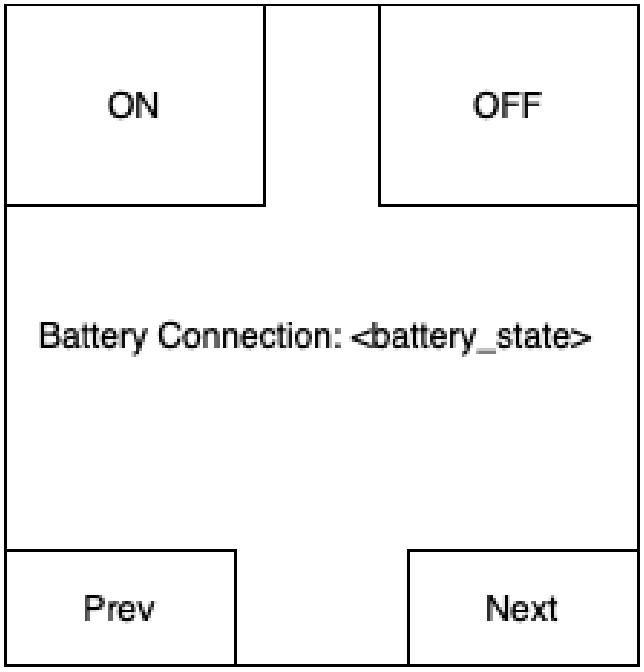


Figure 14. Front panel Design for Battery Screen

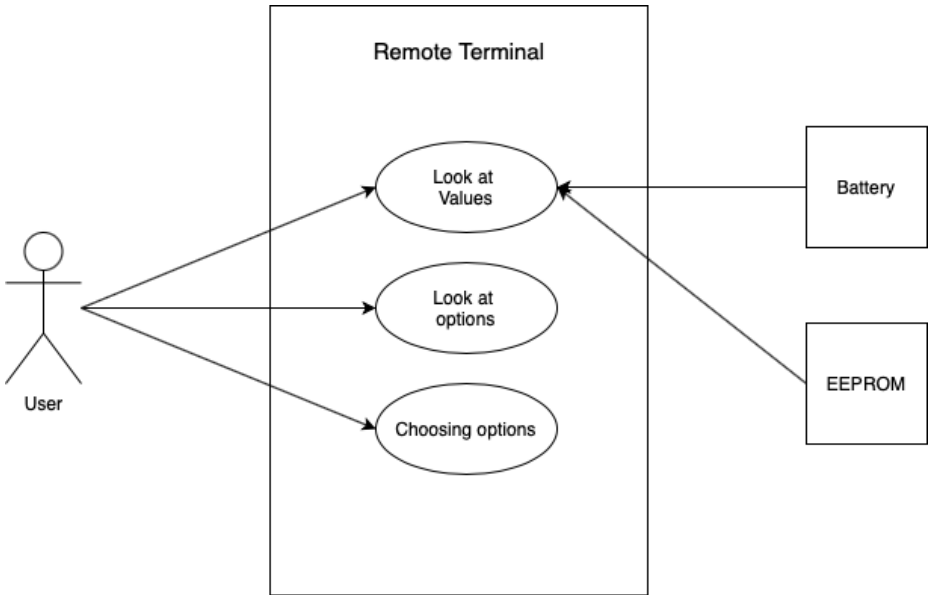


Figure 15. Use Case Diagram for Remote Terminal

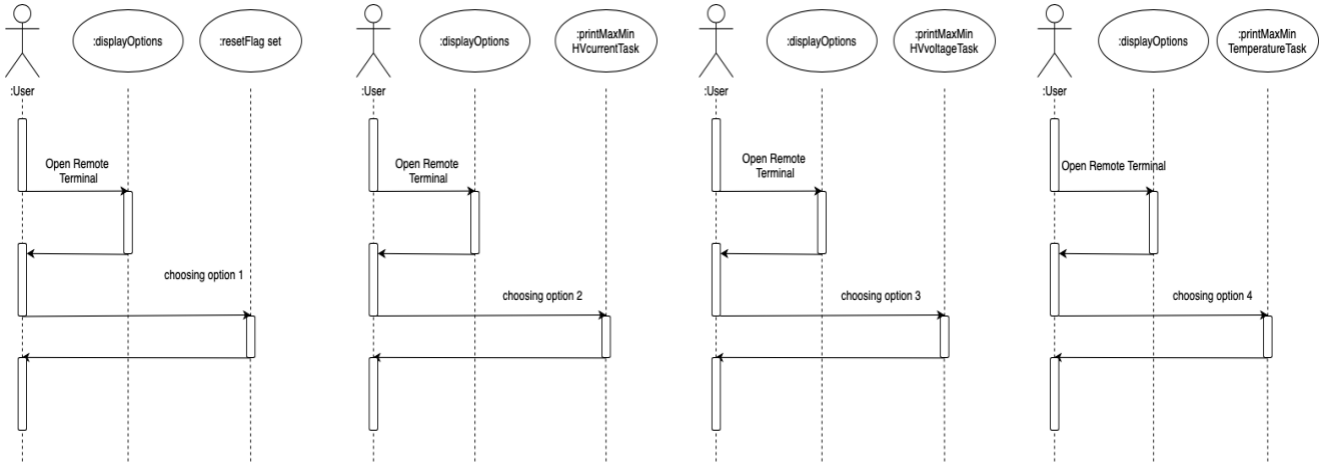


Figure 16. Sequence Diagram for Remote Terminal

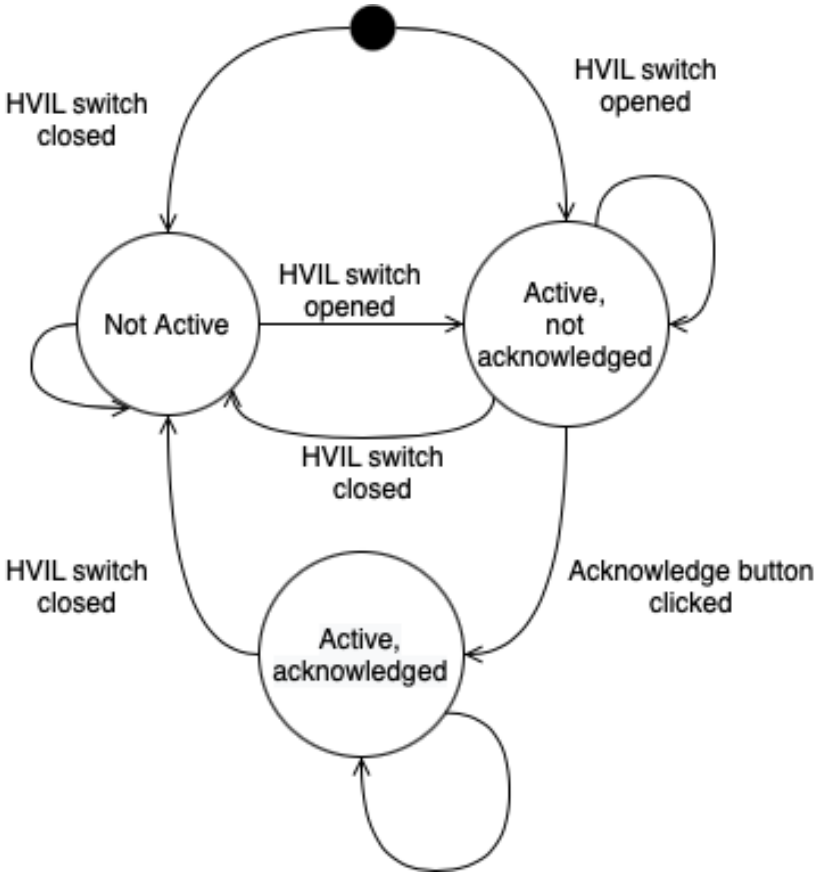


Figure 17. State Diagram for HVIL Alarm

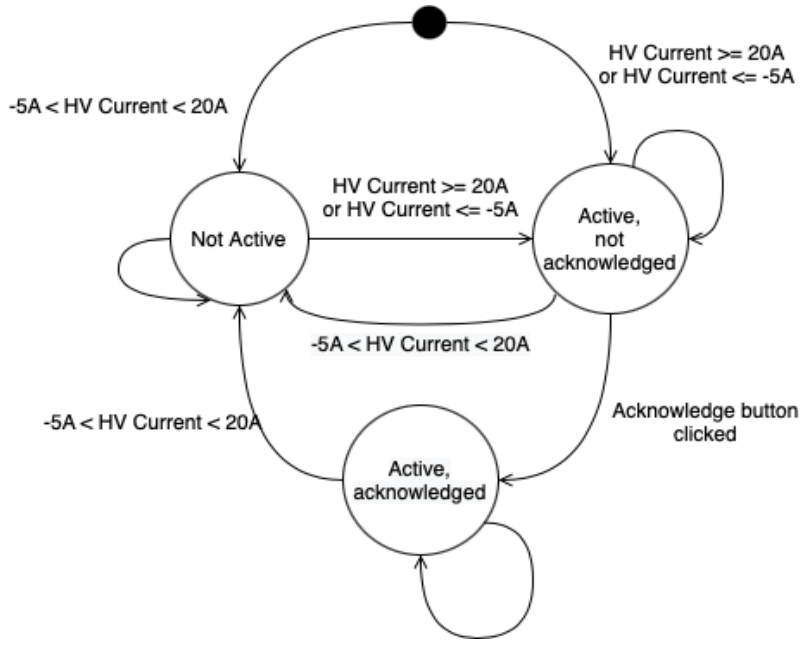


Figure 18. State Diagram for Overcurrent Alarm

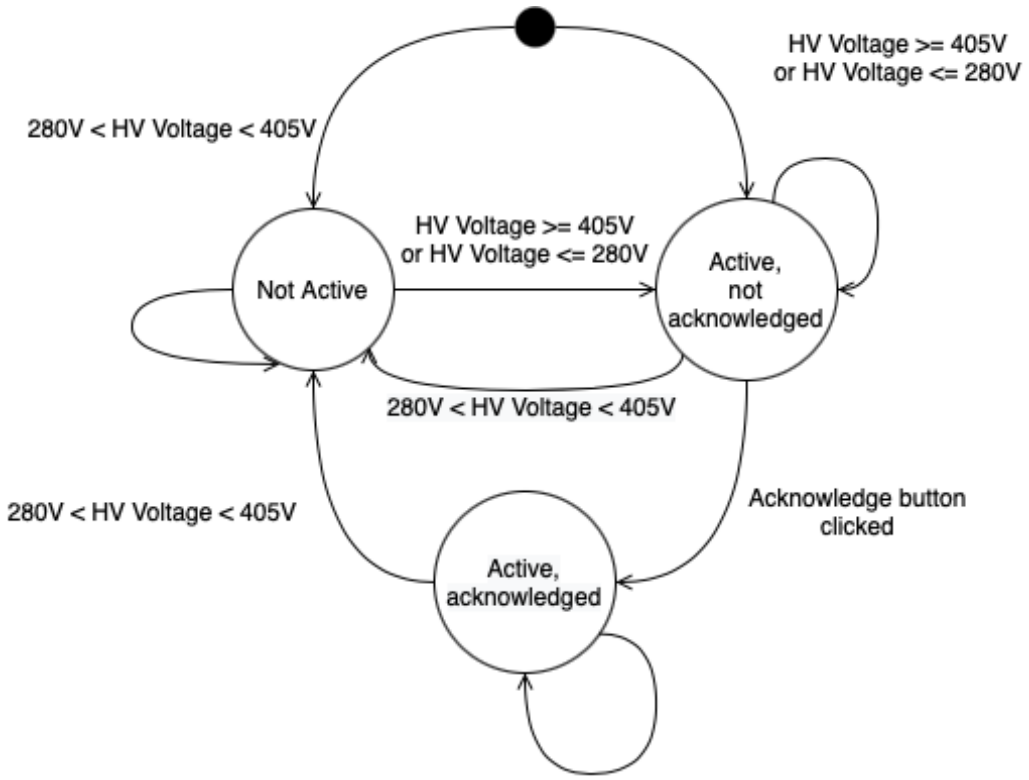


Figure 19. State Diagram for High Voltage out of Range Alarm

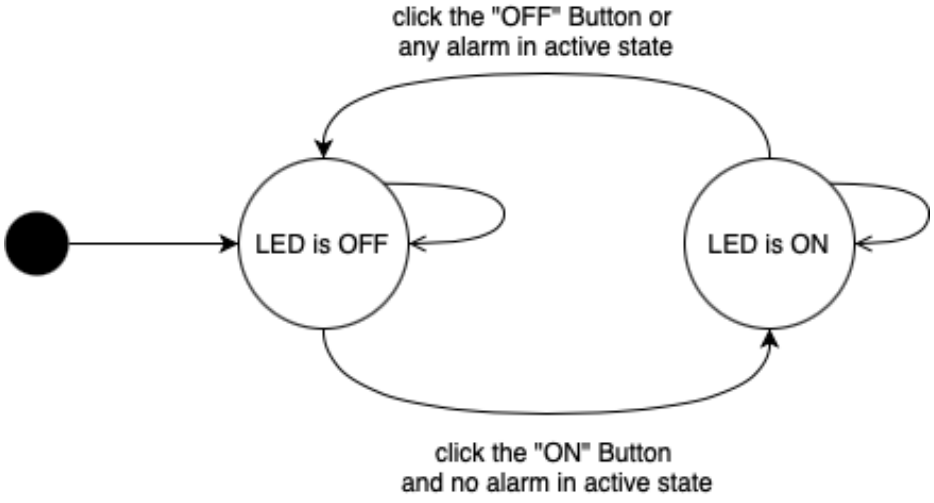


Figure 20. State Diagram for Contactor

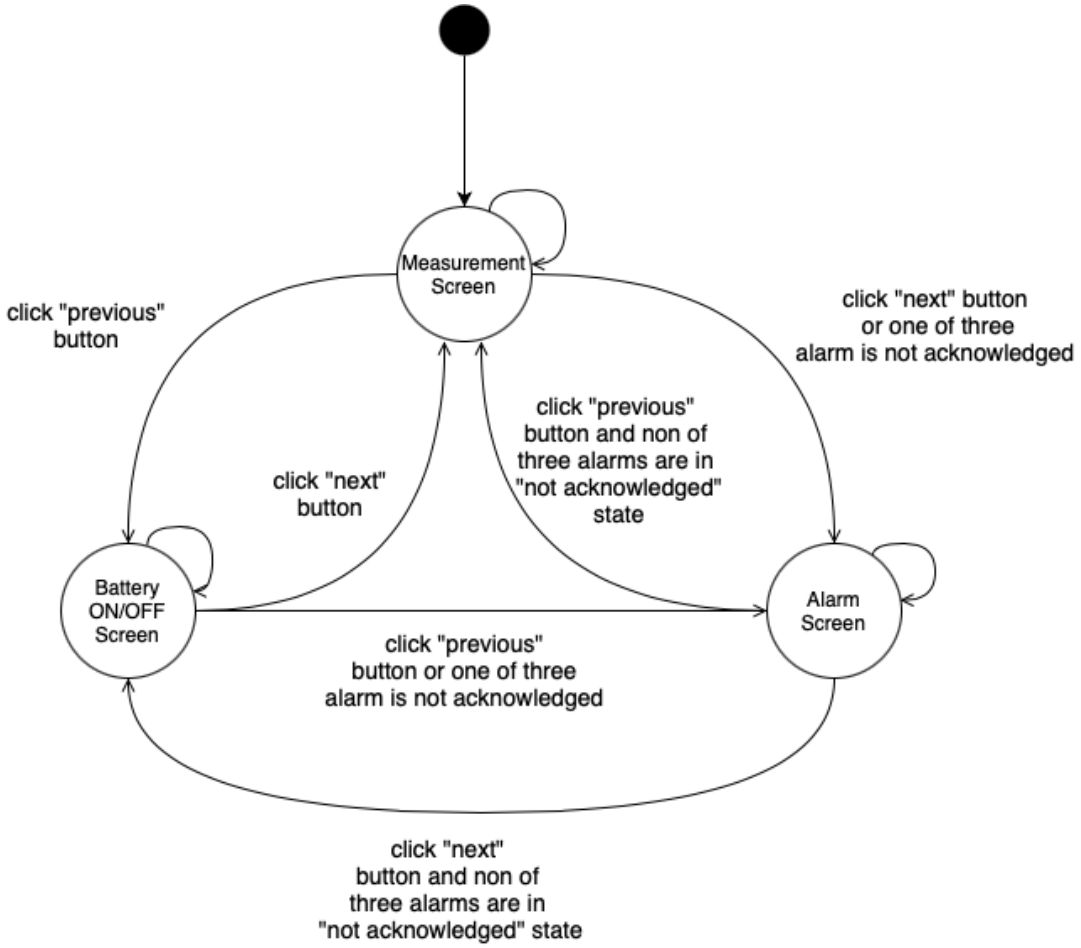


Figure 21. State Diagram for Touch Screen Display

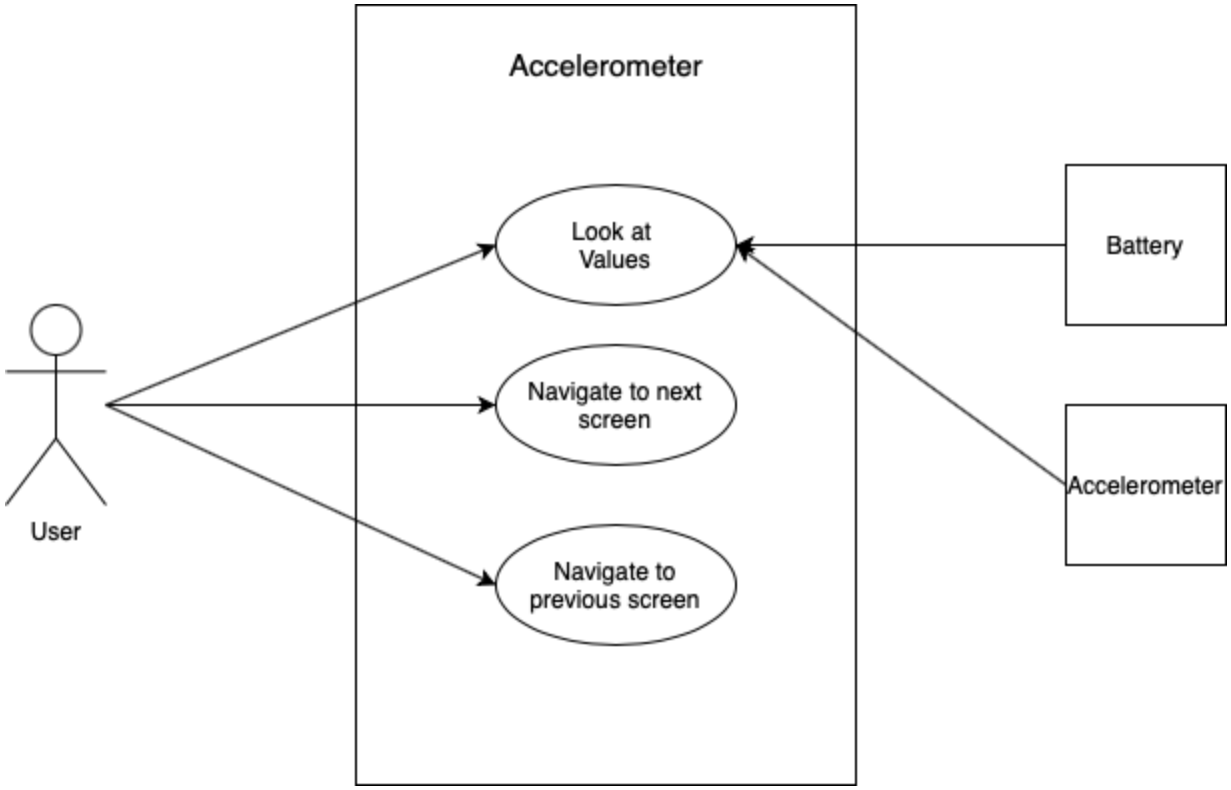


Figure 22. Use case diagram for Accelerometer screen

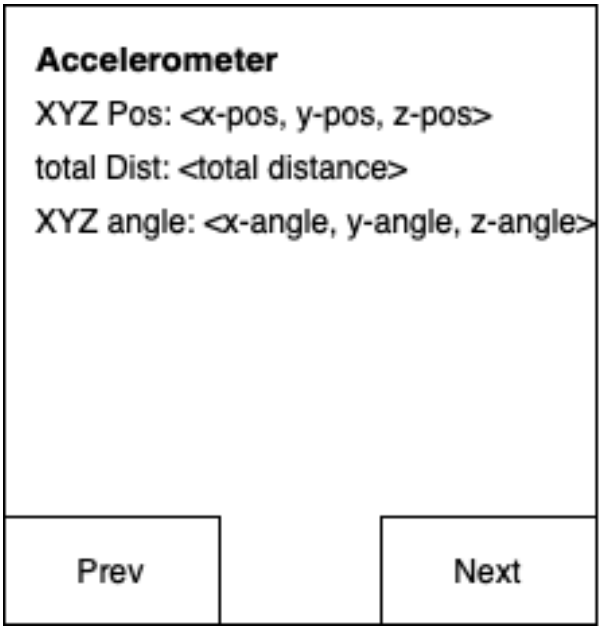


Figure 23. Front panel design for accelerometer screen