Flux Capacitor - A State of Charge (SOC) and State of Health (SOH) estimator for Batteries with Improved Coulomb Counting Technique

Frank Li
Student, Earl of March Secondary School
April 2020

TABLE OF CONTENTS

1.INTRODUCTION

1.1 Project Overview

Flux Capacitor is supposed to be an affordable and effective battery capacity estimator intended for electric vehicles, but more precisely, Earl of March Secondary School's Waterloo Electric Vehicle. This vehicle is required to use the *Optima Yellowtop D35* 12V lead-acid battery. The battery monitoring system should be lightweight and capable to estimate with minimal input.

This paper will touch upon the theoretical background, hardware design, algorithm design, and data required for the Flux Capacitor. References are provided in the last chapter. In addition, the design and creation of this system can be monitored on this following repository: https://github.com/spacemonger/flux-capacitor.

2. THEORETICAL BACKGROUND

2.1 Lead Acid Batteries

Lead acid batteries provide energy through its chemical properties. Within the battery, there are cells that are composed of lead, lead oxide, water, and sulfuric acid. These cells undergo redox reactions to provide the 12V potential. There are two categories of lead-acid batteries: The first being startup batteries and the second being deep-cycle batteries. The *Optima D35* is a deep-cycle battery. This means it can be charged and discharged for multiple times. In addition, lead-acid batteries also form into three sub-categories: 1. Flooded 2. Gel Cell 3. Absorbed Glass Mat (AGM). The required battery is an AGM battery.

2.2 Battery Variables

2.2.1 Voltage

2.2.2 Capacity

2.2.3 Internal Resistance

2.3 State of Charge Monitoring Methods

2.3.1 Open Circuit Voltage Method

2.3.2 Specific Gravity Method

2.3.3 Coulomb Counting Method

2.3.4 Book-Keeping Method

2.3.5 Kalman Filter Method

2.3.6 Hybrid Methods

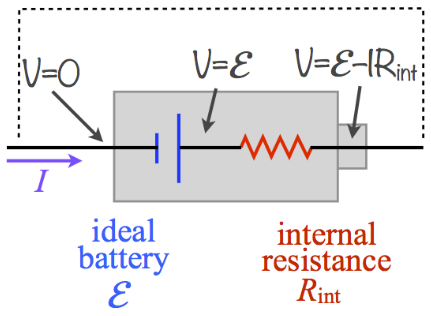
This is the proposed method. Combination of Open Circuit Voltage in the initial stage and then an enhanced coulomb counting method for the rest of the time.

2.4 Electric Equivalent Circuit Models

2.4.1 Simple Battery Models

Since there is no such thing as an ideal battery, this type of battery will be considered for this first version of the Flux Capacitor. This model consists of taking in consideration of internal resistance and provides this following formula:

$$\Delta V = \mathcal{E} - IR_{\text{int}}$$



The e would represent the electromotive force and the open-circuit voltage potential whilst the V would be the measured terminal voltage. Thus, with the known current, internal resistance, and terminal voltage the open circuit voltage can be found. This model is very simple as it does not take into account temperature.

2.4.2 Thevenin Battery Models

2.4.3 Third-order Battery Model

2.4.4 Simplified Circuit Model

3. PRODUCT DESIGN & IMPLEMENTATION

In this following section, the Flux Capacitor will be explained by its sub-circuits and their contributions to the complete system.

3.1 Block Diagram

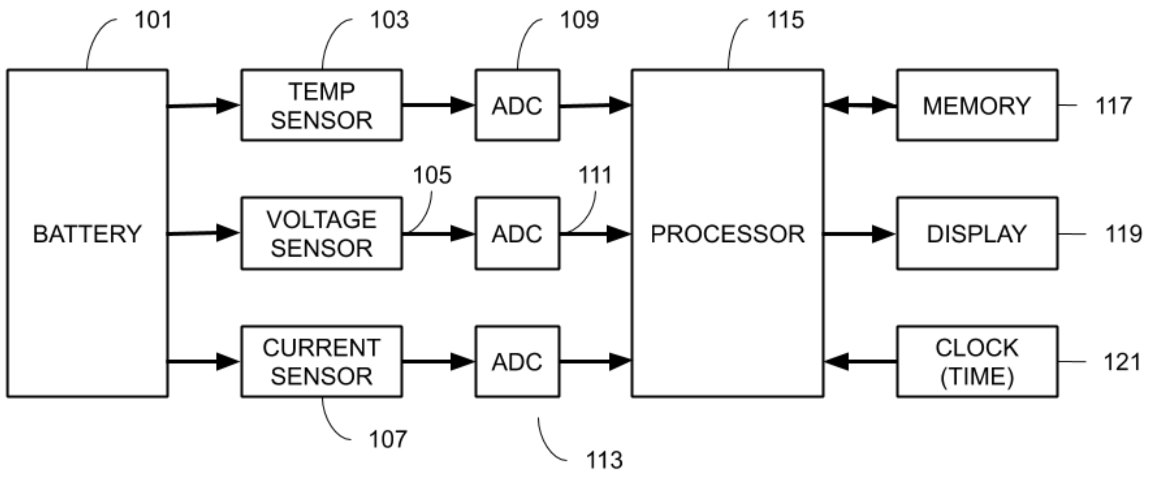
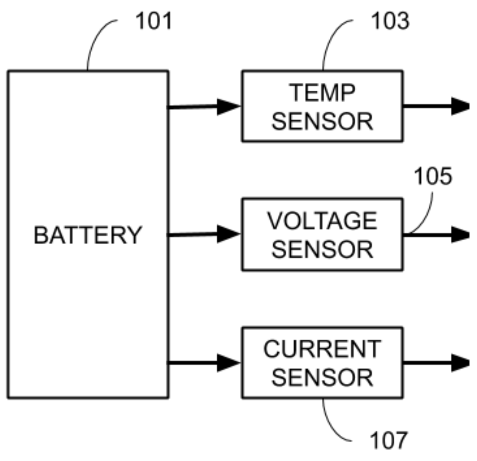


FIG. 1

3.2 Sub-Circuits

3.2.1 Battery-Sensor Subcircuit



3.2.1.1 Battery-Temperature Sensor Subcircuit

Battery(101) will be connected to the temperature sensor(103) to provide the processor the operating temperature of the battery. Operation temperatures determine weight factors of the estimated State of Charge of the battery. Depending on the type of battery, the range of the temperatures may vary. The possible cases of the temperature range and the battery performance can be as followed:

Lower the temperature is then the smaller this variable is <1. This results in adding a certain amount of voltage onto the sampled voltage

SATP results in the variable being ~1.

Higher temperatures result in the variable being >1. This results in subtracting a certain amount of voltage from the sampled voltage

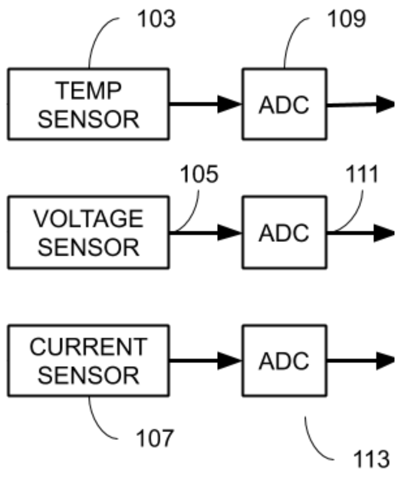
3.2.1.2 Battery-Voltage Sensor Subcircuit

Battery(101) will be connected to the voltage sensor(105) to provide the processor with the initial open circuit voltage. This voltage represents the initial voltage that the battery has for the drive. Any voltages determined afterwards will be voltage used to be compared to a lookup table that compares the State of Charge of the battery and the voltage. This can also provide a voltage for the state of health of the battery.

3.2.1.3 Battery-Current Sensor Subcircuit

Battery(101) will be connected to the current sensor(107) to provide the processor with the current drawn from the battery. This sampling rate can be done with 10Hz or above. Determining the current drawn will allow for the calculation of the depth of discharge.

3.2.2



3.2.2.1 Temperature Sensor-ADC(109) Subcircuit

Temperature sensor (103) will provide an analog signal that will be converted to a digital signal by ADC IC(109).

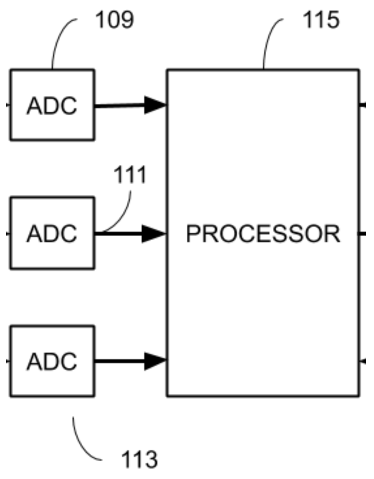
3.2.2.2 Voltage Sensor-ADC(111) Subcircuit

Voltage sensor (105) will provide an analog signal that will be converted to a digital signal by ADC IC(111).

3.2.2.3 Current Sensor-ADC(107) Subcircuit

Current sensor (107) will provide an analog signal that will be converted to a digital signal by ADC IC(113).

3.2.3



3.2.3.1 ADC(109)-Processor Subcircuit

ADC(109) will provide the processor (115) with digital signals of the temperature.

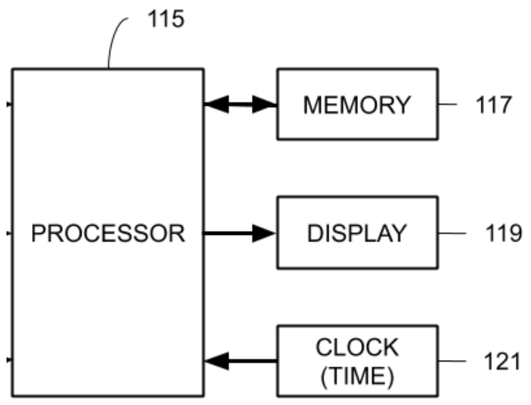
3.2.3.2 ADC(111)-Processor Subcircuit

ADC(109) will provide the processor (115) with digital signals of the voltage.

3.2.3.3 ADC(113)-Processor Subcircuit

ADC(109) will provide the processor (115) with digital signals of the current.

3.2.4



3.2.4.1 Processor-Memory Subcircuit

Processor (115) will log and update the temperature, voltage, and current values by writing to memory (117).

3.2.4.2 Processor-Display Subcircuit

Processor (115) will show the state of charge of the battery with display (119).

3.2.4.3 Processor-Clock Subcircuit

Processor (115) will perform the battery percentage estimation with regards to clock (121).

3.3 Product Interfaces

3.3.1 Internal Interfaces

I2C Communication Interface: Components including the 2 ADS1015 ICs, the 0.91" OLED screen, and the Arduino nano use this interface to communicate. The Arduino acts as the master while the other components are the slaves. Since there are 2 ADS1015 ICs, they must have different bus addresses, thus, one is connected to ground and the other to VDD.

3.3.2 External Interfaces

Switch: In order for the following device to function, the car battery must be cranked. This will be done by a switch. This switch will allow for current to flow, thus, powering the Flux Capacitor.

Display: The user will be able to monitor their batteries' performance with the 0.91" OLED screen.

3.4 Connectors and Pinouts

3.4.1 Arduino's Pinouts

3.4.1.1 Arduino's Voltage In
   Connected to L7805CV OUT pin

3.4.1.2 Arduino's Ground
   Connected to OPTIMA D35 GROUND

3.4.1.3 Arduino's A4
   Connected to SDA of ADS1015s & OLED

3.4.1.4 Arduino's A5
   Connected to SCL of ADS1015s & SCK of OLED

3.4.1.5 Arduino's D2
   Connected to DQ of DS18B20

3.4.1.6 Arduino's Micro-USB to USB Type A V2.0
   Connected to Computer USB Type A port to upload code

3.4.2 ME0909 Connectors

3.4.2.1 Positive Terminal
   Connected to Shunt's POSITIVE

3.4.2.1 Negative Terminal
   Connected to Battery GROUND

3.4.3 ADS1015 Voltage Pinouts

3.4.3.1 Vin
   Connected to Arduino's 5V

3.4.3.2 Ground
   Connected to Arduino's GROUND

3.4.3.3 SDA

Connected to Arduino’s A4

3.4.3.4 SCL

Connected to Arduino’s A5

3.4.3.5 ADDR

Connected to Arduino’s GROUND

3.4.3.6 A0

Connected to Voltage Divider

3.4.4 ADS1015 Current Pinouts

3.4.4.1 Vin

Connected to Arduino’s 5V

3.4.4.2 Ground

Connected to Arduino’s GROUND

3.4.4.3 SDA

Connected to Arduino’s A4

3.4.4.4 SCL

Connected to Arduino’s A5

3.4.4.5 ADDR

Connected to Arduino’s 5V

3.4.4.6 A0

Connected to Shunt POSITIVE

3.4.4.7 A1

Connected to Shunt NEGATIVE

3.4.5 DS18B20 Pinouts

3.4.5.1 VDD

Connected to Arduino’s 5V

3.4.5.2 DQ

Connected to Arduino’s D2

3.4.5.3 GND

Connected to Arduino’s GROUND

3.4.6 0.91” OLED Display Pinouts

3.4.6.1 Vin

Connected to Arduino’s 5V

3.4.6.2 GND

Connected to Arduino’s GROUND

3.4.6.3 SCK

Connected to Arduino’s A5

3.4.6.4 SDA

Connected to Arduino’s A4

3.4.7 L7805CV Pinouts

3.4.7.1 Vin

Connected to Batteries Positive Terminal in Parallel

3.4.7.2 GND

Connected to Batteries GROUND

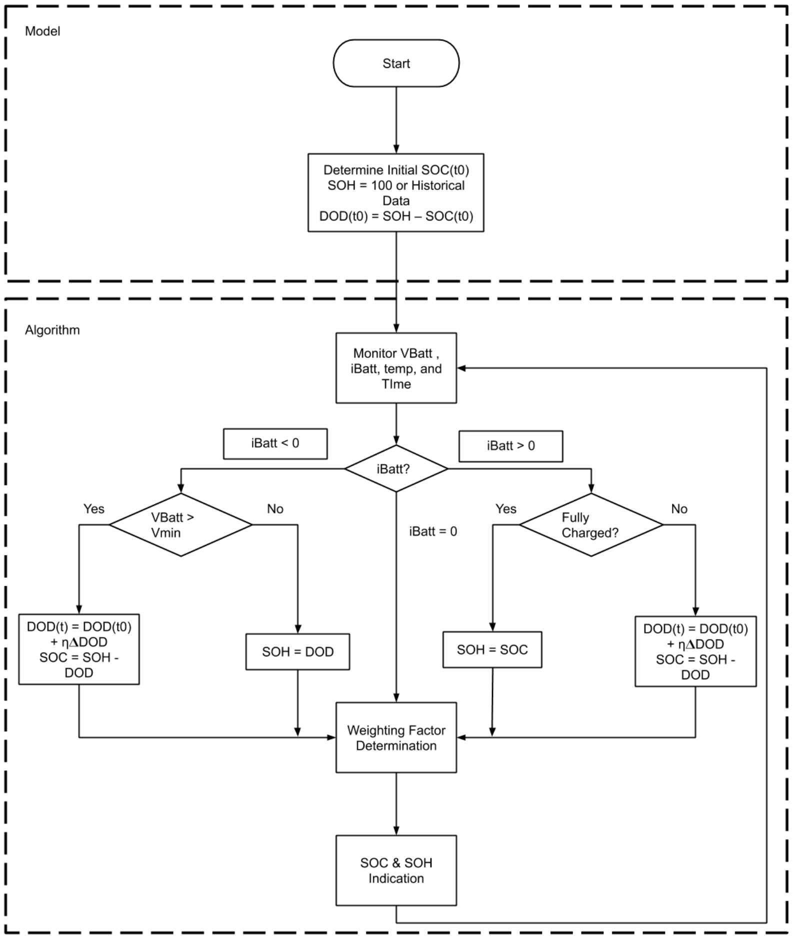
3.4.7.3 OUT

Connected to Capacitor 2's POSITIVE SIDE

4. ALGORITHM DESIGN & IMPLEMENTATION

4.1 Designed Method

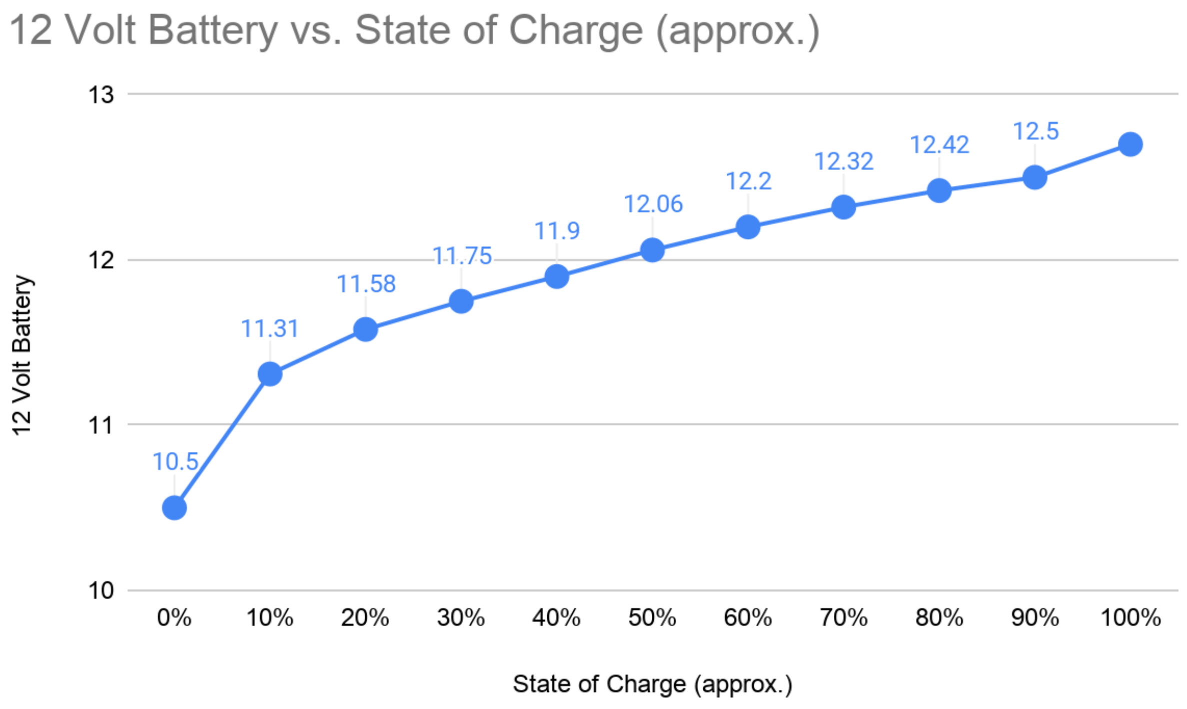
This designed method will incorporate OCV at the initial cranking period to determine the starting State of Charge and then used an enhanced coulomb counting method that uses SOC, SOH, and DOD.



4.1.1 OCV

There is a universal SOC algorithm that works especially well for lead-acid batteries. This algorithm consists of measuring the OCV and using a lookup table to find the SOC, but there is a required time in between samples to make this precise. Usually, this time period may be ~1.5-2 hours. Thus, in this improved method this relationship is only used once at the start. The definition of SOC: $SOC = \frac{C_{Releasable}}{C_{Rated}} 100\%$. This relationship does not provide a straight line, but with sample points a linear equation can be used for the domains of voltages:$f(OCV) = a *$

$OCV - b$. These are approximations that need to be further tested with real experiments and simulations.



State of Charge (approx.)	12 Volt Battery	Volts per Cell	Linear Equation
100%	12.70	2.12	Y = 2x + 10.7
90%	12.50	2.08	Y = 0.8x + 11.78
80%	12.42	2.07	Y = x + 11.62
70%	12.32	2.05	Y = 1.2x + 11.48
60%	12.20	2.03	Y = 1.4x + 11.36
50%	12.06	2.01	Y = 1.6x + 11.26
40%	11.90	1.98	Y = 1.5x + 11.3
30%	11.75	1.96	Y = 1.7x + 11.24
20%	11.58	1.93	Y= 2.7x + 11.4
10%	11.31	1.89	Y = 8.1x + 10.5
0%	10.50	1.75	

There is also a temperature adjustment.

4.1.2 State of Health(SOH) Estimation

The definition of SOH: $SOH = \frac{C_{Max}}{C_{Rated}} 100\%$. However, the state of health of the battery will decrease over time due to cycles and leakages. Therefore there must be an adjustment factor for the initial rated capacity. In this case, 48Ah @ C20. This error can be found by finding the difference in measured open circuit voltage and theoretical OCV. This provides a voltage loss that is then divided by the largest possible voltage loss that the battery can handle to provide a percentage: $SOH\%error = \frac{OCV - V_0}{OCV - Vlimit}$. Thus, an actual SOH can be estimated: $Actual\ SOH = \frac{C_{Max}}{C_{Rated}} 100\% \times SOH\%error$.

4.1.3 Depth of Discharge(DOD) & State of Charge(SOC)

$$\eta_{discharge} = \frac{I_1T_1 + I_2T_2}{C_{Max}}$$

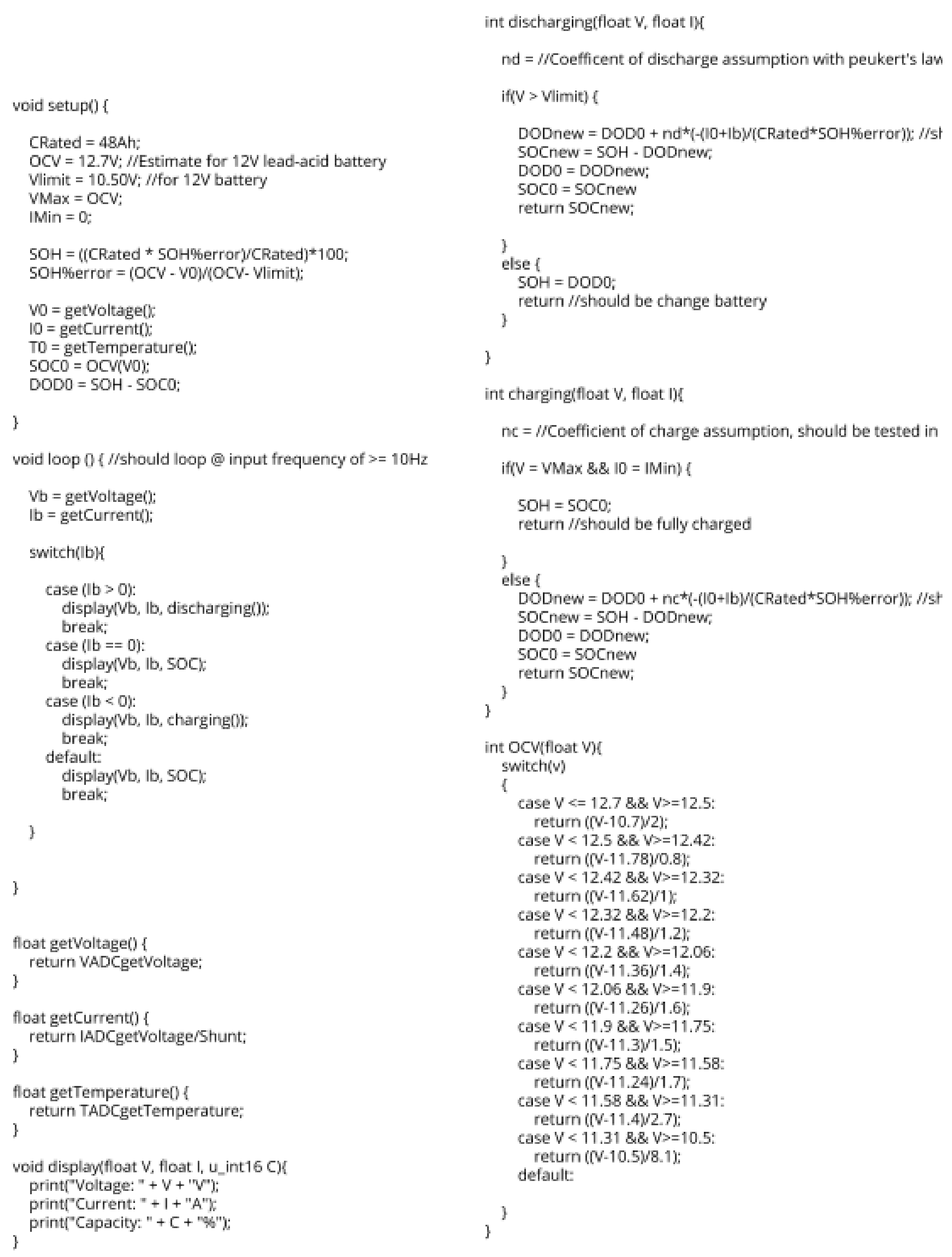
$$\eta_{charge} = \frac{C_{Discharge,rate\ max}}{C_{Charge,rate\ max}}$$

$$DOD(t) = DOD(t_0) + \eta\Delta DOD$$

$$SOC(t) = SOH(t) - DOD(t)$$

4.2 Implementation

C-Based Pseudo Code:



```
void setup() {

    CRated = 48Ah;
    OCV = 12.7V; //Estimate for 12V lead-acid battery
    Vlimit = 10.50V; //for 12V battery
    VMax = OCV;
    IMin = 0;

    SOH = ((CRated * SOH%error)/CRated)*100;
    SOH%error = (OCV - V0)/(OCV- Vlimit);

    V0 = getVoltage();
    I0 = getCurrent();
    T0 = getTemperature();
    SOC0 = OCV(V0);
    DOD0 = SOH - SOC0;

}

void loop () { //should loop @ input frequency of >= 10Hz

    Vb = getVoltage();
    Ib = getCurrent();

    switch(Ib){

        case (Ib > 0):
            display(Vb, Ib, discharging());
            break;
        case (Ib == 0):
            display(Vb, Ib, SOC);
            break;
        case (Ib < 0):
            display(Vb, Ib, charging());
            break;
        default:
            display(Vb, Ib, SOC);
            break;

    }


}


float getVoltage() {
    return VADCgetVoltage;
}

float getCurrent() {
    return IADCgetVoltage/Shunt;
}

float getTemperature() {
    return TADCgetTemperature;
}

void display(float V, float I, u_int16 C){
    print("Voltage: " + V + "V");
    print("Current: " + I + "A");
    print("Capacity: " + C + "%");
}
```

```
int discharging(float V, float I){

    nd = //Coefficent of discharge assumption with peukert's law

    if(V > Vlimit) {

        DODnew = DOD0 + nd*(-(I0+Ib)/(CRated*SOH%error)); //sh
        SOCnew = SOH - DODnew;
        DOD0 = DODnew;
        SOC0 = SOCnew
        return SOCnew;

    }
    else {
        SOH = DOD0;
        return //should be change battery
    }

}

int charging(float V, float I){

    nc = //Coefficient of charge assumption, should be tested in

    if(V = VMax && I0 = IMin) {

        SOH = SOC0;
        return //should be fully charged

    }
    else {
        DODnew = DOD0 + nc*(-(I0+Ib)/(CRated*SOH%error)); //sh
        SOCnew = SOH - DODnew;
        DOD0 = DODnew;
        SOC0 = SOCnew
        return SOCnew;
    }
}

int OCV(float V){
    switch(v)
    {
        case V <= 12.7 && V>=12.5:
            return ((V-10.7)/2);
        case V < 12.5 && V>=12.42:
            return ((V-11.78)/0.8);
        case V < 12.42 && V>=12.32:
            return ((V-11.62)/1);
        case V < 12.32 && V>=12.2:
            return ((V-11.48)/1.2);
        case V < 12.2 && V>=12.06:
            return ((V-11.36)/1.4);
        case V < 12.06 && V>=11.9:
            return ((V-11.26)/1.6);
        case V < 11.9 && V>=11.75:
            return ((V-11.3)/1.5);
        case V < 11.75 && V>=11.58:
            return ((V-11.24)/1.7);
        case V < 11.58 && V>=11.31:
            return ((V-11.4)/2.7);
        case V < 11.31 && V>=10.5:
            return ((V-10.5)/8.1);
        default:

    }
}
```

5. SCHEDULING

5.1 Tasks

- Ordering Parts
    - Creating requirements list
    - Creating potential parts list
    - Filtering parts list and creating final list
    - Calculate Total Cost & Potential future options
    - Order parts
- Hybrid Coulomb Coulomb Algorithm
    - Research different methods of estimating battery health
        - Voltage Method
        - Chemical Method
        - Coulomb Counting
        - Kalman Filtering Method
        - Book-keeping
        - Data based
    - Compare different Coulomb Counting Method
    - Come up with a Hybrid Method
        - Create Flowchart
        - Create an Embedded Language Flowchart
            - Create a list of required variables
            - Create a list of required classes
            - Create a list of required functions
    - Create the embedded software in arduino
    - Revise and entail the software to the requirements of hardware
- Lead Acid Battery Simulink
    - Register for Free Matlab Trial
    - Download Matlab & Simulink
    - Perform the following Simulations
        - Internal Resistance Estimation-Simulink
        - Battery Discharge Constant load –Simulink
        - Battery Discharge Variable load –Simulink
- Proof of Concepts & Sub-Circuits
    - Creating a simulated Car Battery
        - Creating Schematics of Sub-Circuits
            - CurrentADC.fzz
            - VoltageADC.fzz
            - TemperatureADC.fzz
            - VoltageRegulator.fzz
        - Testing of Sub-Circuits

      - Current Shunt Resistor
      - Voltage Divider
      - Temperature
      - Voltage Regulator
  - Testing with real Car Battery
    - Current Shunt Resistor
    - Voltage Divider
    - Temperature
    - Voltage Regulator
- Combination of Sub-Circuits
  - Creating full circuit schematic
    - Implementation of data logging
  - Implementation of Hybrid Coulomb Counting Algorithm
- Testing
  - Testing and Debugging of Flux Capacitor
  - Comparing theoretical values and experimental values
    - Simulink values
    - Flux Capacitor Values
- Revisions
  - Creation of improved parts list
  - Package for Flux Capacitor

5.2 Gantt Chart Schedule

Tasks	April 5	April 12	April 19	April 26	May 3	May 10	May 17	May 24	June 8
Parts List	🟦								
Research Methods	🟦								
Algorithm Flowchart	🟦	🟦							
Embedded Language Flowchart		🟦	🟦						
Embedded Software			🟨	🟨	🟨	🟨	🟨	🟨	
Internal Resistance Estimation		🟥	🟥	🟥					
Battery Discharge Constant load		🟥	🟥	🟥					
Battery Discharge Variable load		🟥	🟥	🟥					
Schematics of Sub-Circuits		🟩							
Testing of Sub-Circuits			🟩	🟩					
Battery Tests				🟩	🟩				
Creating full circuit schematic			🟩	🟩					
Creating full Circuit				▩	▩	▩	▩	▩	
Implementating Algorithm						🟨	🟨	🟨	
Testing of full Circuit							▩	▩	
Package								▩	
Improved Parts List									🟦

Legend	
Planning & Research	🟦
Software	🟨
Simulations	🟥
Sub-circuits	🟩
Final Product	▩

APPENDIX

Appendix A

$$SOC = \frac{C_{Releasable}}{C_{Rated}} 100\% \; or \; = f(OCV) = a * OCV - b$$

$$DOD = \frac{C_{Released}}{C_{Rated}} 100\%$$

$$\Delta DOD = \frac{-\int_{t_0}^{t_0+\tau} iBatt(t)dt}{C_{Rated}} 100\% \qquad \tau \in [t_0\,, t]$$

$$SOH = \frac{C_{Max}}{C_{Rated}} 100\%$$

$$DOD(t) = DOD(t_0) + \eta \Delta DOD$$

$$SOC(t) = SOH(t) - DOD(t)$$

$$OCV = Extracted\ from\ Voltage\ Sensor, no\ load$$

$$iBatt = Extracted\ From\ Current\ Sensor$$

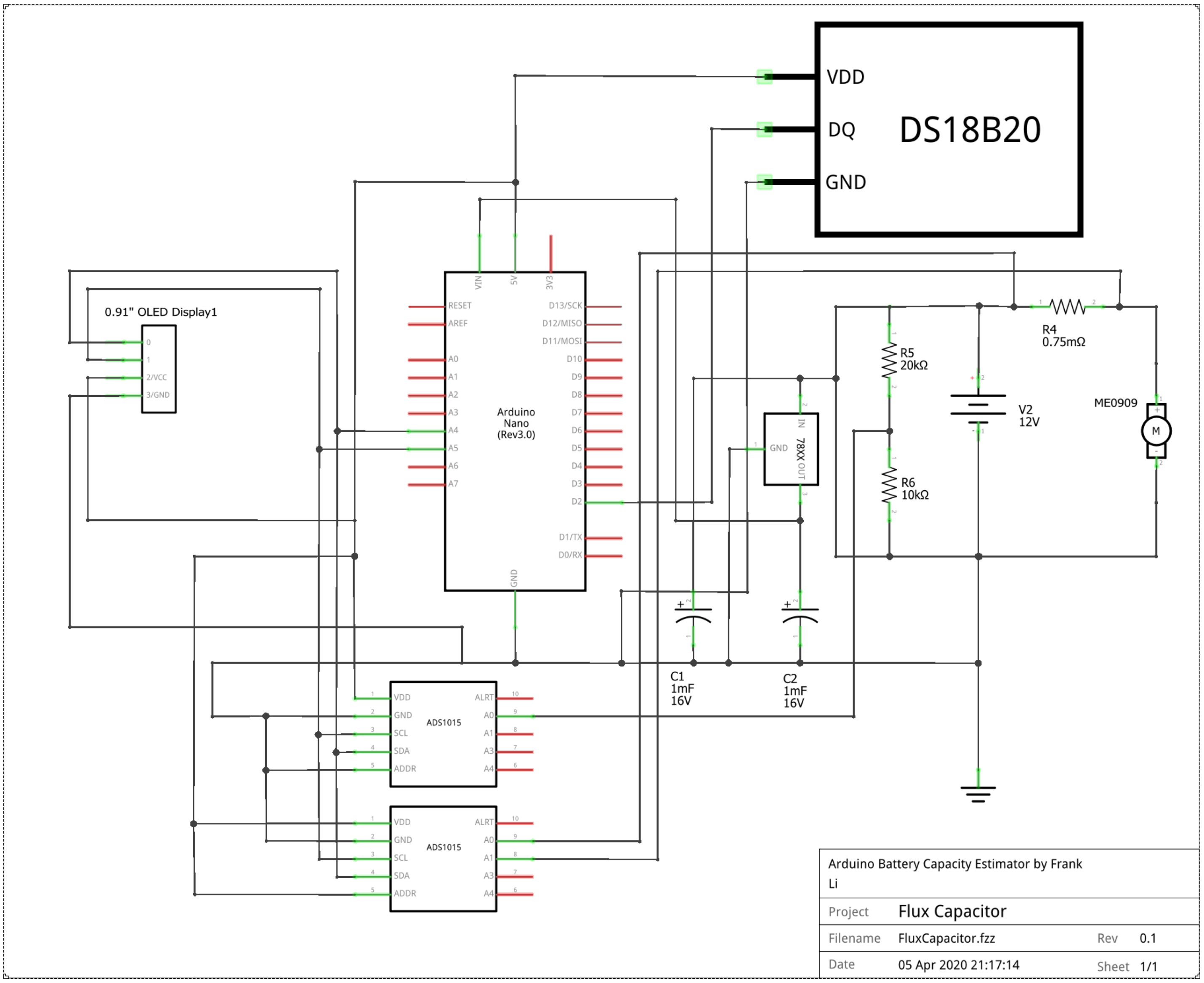
$$VBatt = Extracted\ from\ Voltage\ Sensor$$

$$V_{min} = dependant\ but\ should\ be\ 11.6 - 11.8\ volts\ for\ AGM\ batteries\ or\ 10.5$$

$$\eta_{discharge} = \frac{I_1T_1 + I_2T_2}{C_{Max}}$$

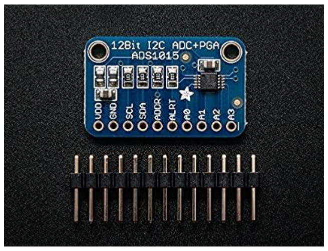
$$\eta_{charge} = \frac{C_{Discharge,rate\ max}}{C_{Charge,rate\ max}}$$

Appendix B Schematics



Appendix C Parts List

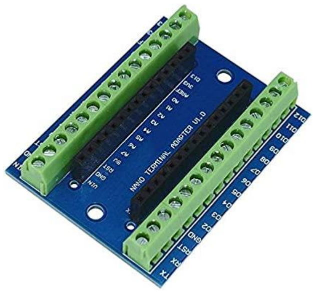
ADS1015



Optima Yellowtop D35

Arduino Nano

Arduino Nano Expansion Board



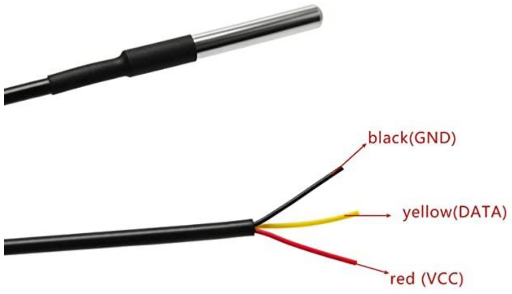
100 Amp Shunt Resistor 0.75mOhm Resistance

Voltage Regulator

1x 10 kΩ resistor
1x 20 kΩ resistor

OLED Display

Temperature Sensor



Capacitors