Sensible (To Me) Electric Bike Charging

Using An Arduino Uno With Touchscreen and Controlled Power Cord

Mark G.

Oct. 12, 2024

This presentation will look at how to control the charging of an e-bike in order to enhance the life-time of a battery. The controlling mechanism consists of an Arduino Uno microcontroller with a Touchscreen shield, coupled with a PowerTail isolated-relay controlled A/C extension cord.

The interface will be simply touchscreen based ON/OFF buttons, plus buttons which add or set time blocks of charging duration. The interface is written using the Arduino IDE and its C++ coding environment.

Figure 1: The e-bike which is the impetus for this project: Devinci E-Griffin

Contents

List of Figures

1 Introduction

My belief is currently that I can have my e-bike battery last longer if I start recharging at 20-40% and stop recharging at around 80% of battery capacity.

Alleged sources:

- Debunking Lithium-Ion Battery Charging Myths: Best Practices for Longevity - StablePSU[1].

This is an experiment of sorts. Problems with this experiment are:

1. No control subject. I can only test the up-to-80 theory, since I only have one battery. If I had two batteries (or more) I could test going to 100% charge, and see how that compares to 80%. It would be handy to spin up a parallel universe to test both cases.
2. I have no idea how the battery was treated before I possessed it. From the invoice, the e-bike seems to have been sourced around 2022, so the battery could already have had unknown charge cycles.

1.1 Caveats

There are some potential problems with this approach:

1. Range will be reduced. Obviously, an 80% charged battery will result in fewer kilometers.
2. You have to do initial measurements, a.k.a. work, to determine your hourly charge rate, and hence the amount of time to set for each charging session.
3. New research will come out and my beliefs about battery charging will be shown to be bunk.
4. My experiment in seeing how long this battery lasts might suggest my beliefs are bunk.

1.2 Advantages

There are some advantages to this approach:

1. Customized to your measured charging experience.
2. Charger agnostic. It uses whatever charger came with your e-bike. So based on your observed charge rate, you can apply power to the charger for as long as needed to get near 80%.

[1] https://www.stablepsu.com/lithium-ion-battery-charging-myths/

3. If power to the arduino is lost, power to the charger is terminated. This could be seen as a fail-in-a-safe manner benefit.

2 My Charge Rate

My battery gauge has five sections, or bars as they are commonly known, so each bar represents 20% charge level. This is handy, since the math won't be hard.

For example: from a low charge level (<20%), plug your charger in, set an hour timer, check the amount of charge gained, write it down, set another hour timer, check the charge, and so on until the battery is at 100%. You can calculate how many"bars" per hour you get.

Figure 2: The controller display on the handlebars

My data shows me that I charge approximately one bar per hour of wall time. This suggest that time preset buttons on the gadget's user interface could be `1 hour`, `2 hours`, and `3 hours`.

2.1 Occasional 100%

For the occasional extended trip (if I ever do any), it is perfectly fine to charge to 100%. This suggests a simple set of `ON/OFF` buttons on the user interface, ceding control to the charger, and requiring a manual powering-off of the gadget/charger.

3 Hardware

There are several connected parts in this project. The components needed are:

1. An alternating current (AC) extension cord controlled by a relay.
2. An AC cord Y-splitter.
3. A platform (Arduino Uno) with barrel jack power adapter.
4. A touch screen display shield.
5. Some 28 gauge wire (two strands) about 1 meter in length for flexibility of device placement.

Other items, like a computer for programming and a USB cable are needed to build the system, but are not runtime dependencies.

3.1 PowerTail Relay Controlled Device

The PowerTail device does not need anything other than an `OUTPUT` pin on the arduino. Since the touch screen has many pin requirements, we must make sure we don't pick one that is already in use. Digital pin 7 is one such pin. We also need a ground pin.

Figure 3: Relay controlled power pass through - PowerTail II

3.2 Y-Splitter for PowerTail and Arduino Power

This simple component allows us to easily keep the arduino power and the power controller together on a single outlet. I ordered a dozen from Amazon at some time in the past. They are easy to find.

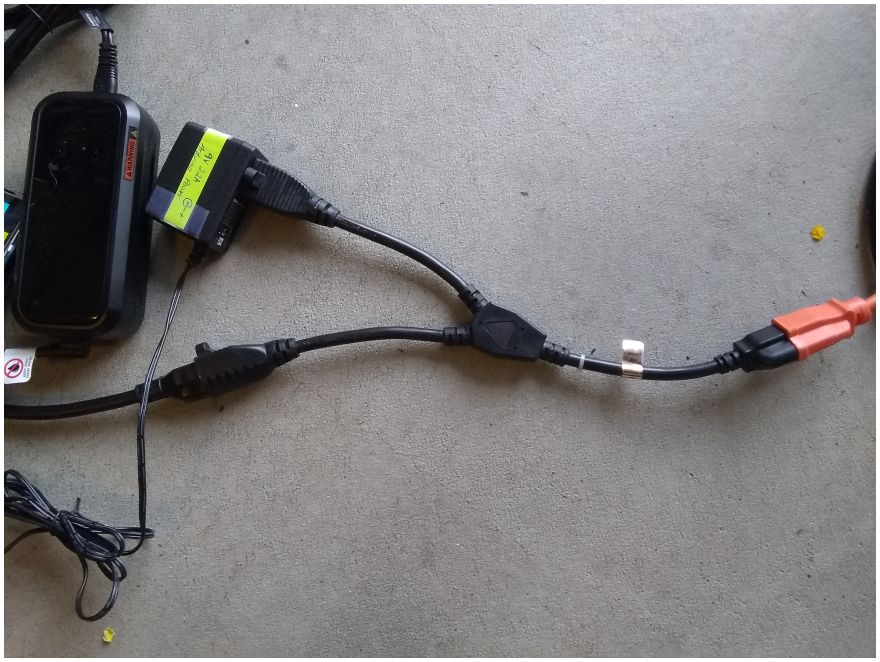

Figure 4: A Y-splitter for providing power to the Arduino and charger

3.3 Touch Screen

The touch screen (see figure 5) is an arduino shield and is designed to plug in directly on top of the arduino. It is an Adafruit 2.8" TFT Touch Shield v2 - Resistive touch screen[2].

The design of the touch screen doesn't allow for access to the rest of the pins on the arduino. So the choice of arduino variant is important.

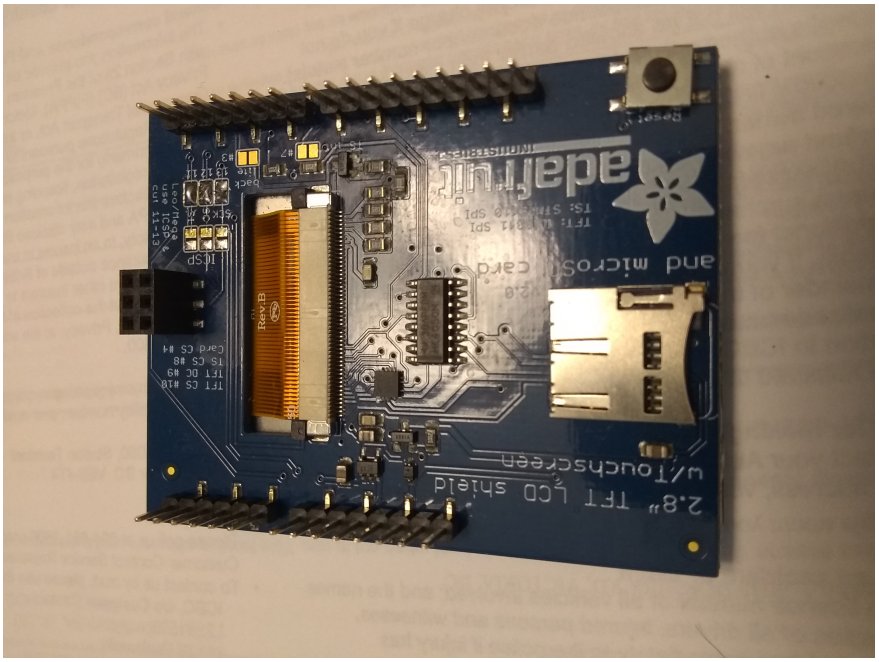

Figure 5: Underside view of the touch screen shield

[2]https://learn.adafruit.com/adafruit-2-8-tft-touch-shield-v2

I happened to have an arduino uno variant that had pinholes parallel to the standard risers.

3.4 Arduino Uno with Parallel Board Pin Holes

Figure 6 shows the arduino variant I had. I do not remember where I got it. Note the pin holes along the inside of the black female risers, which are directly connected to their respective riser pins.

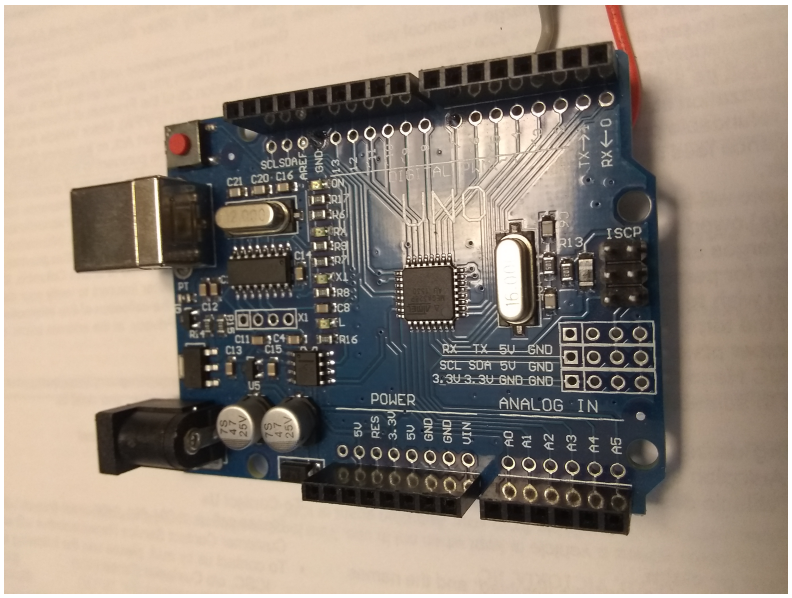

Figure 6: Top view of the arduino microcontroller

3.5 Arduino with Touch Screen

Both the touch screen and arduino before assembly are shown in figure 7.

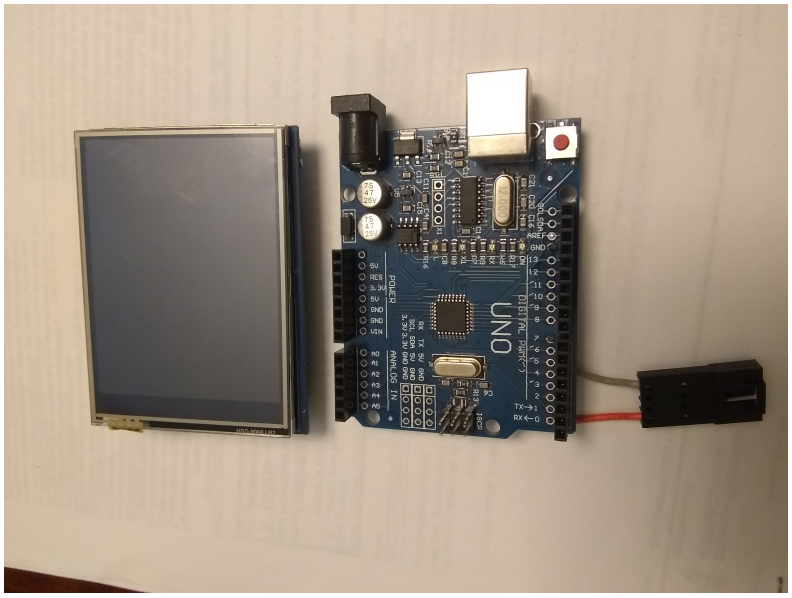

Figure 7: Both the arduino and touch screen before assembly

A teaser look at the final user interface follows on the assembled controller is shown in figure 8.

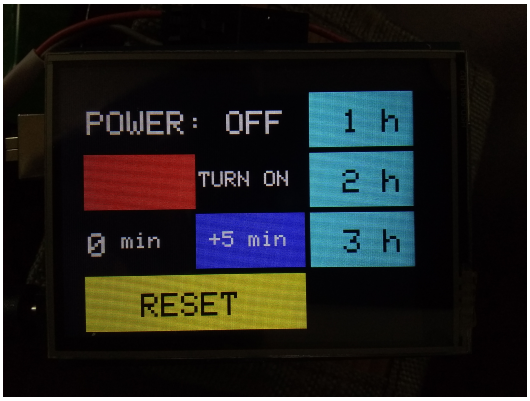

Figure 8: Arduino with touch screen attached

4 Putting the Hardware Together

The touch screen shield simply plugs into the arduino as any other shield would. Easy.

The harder part is soldering two solid copper (28 gauge) wires to the underside of the arduino board. There are extra holes in this board variant that allow for this, as can be seen in figure 9.

The orange wire is connected to digital pin 7 and the brown wire is connected to ground.

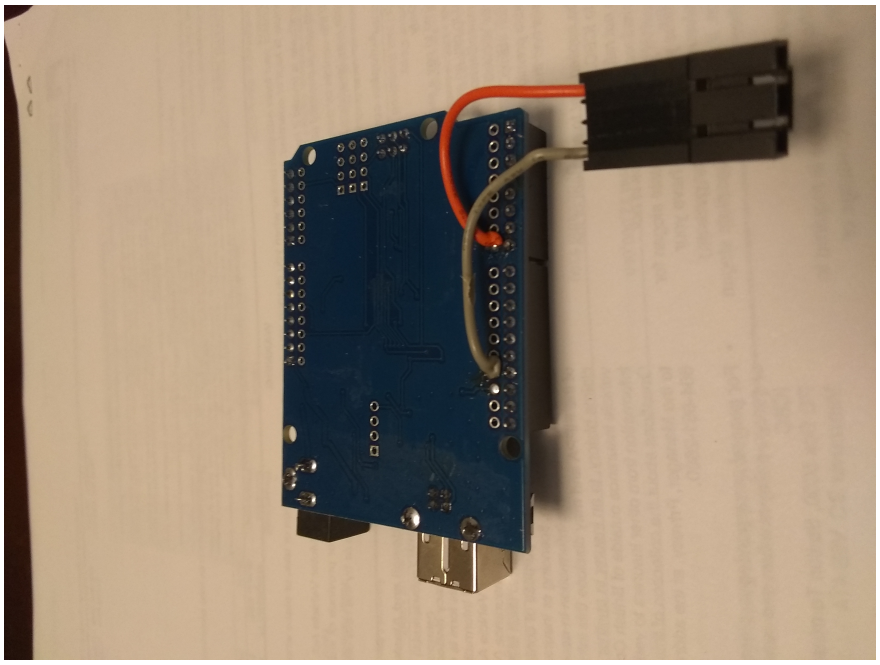

Figure 9: Bottom view of the arduino showing soldered wires for powertail connection

The black plastic bit at the end of the two wires is part of a Molex plug and socket connector kit. It allows us to easily disconnect the powertail from the arduino.

4.1 Full Assembly of Kit

Figure 10 shows all the pieces connected together in their fully functional state, ready to charge the bike.

Figure 10: Full kit connected seen from above

5 Software - C++

The software is not that complicated, since we are given extremely useful examples from the `Adafruit` learning pages. The usual inclusion of the required libraries and customization of the examples is how we proceed.

We use the `Arduino IDE` and our version is 1.8.5, which is the latest for FreeBSD. It works. The usual method for connecting the arduino to the computer via USB is necessary, and won't be covered.

This presentation has two versions of the software. The first is mostly just example code changed to suit the requirements in a prototypical way. The advanced version uses touch screen debouncing and a hardware interrupt timer for timed power durations.

5.1 Libraries

The program uses several libraries that can be obtained from Adafruit on their learning pages and github repositories.

1. `Adafruit GFX` (Graphics) library.
2. `Adafruit STMPE610` library (touch component of screen).
3. `Adafruit ILI9341` library (TFT display screen).
4. The standard `SPI` library.
5. The standard `Wire` library.
6. An interrupt driven timer library. The advanced version of the program uses an extra library, named `TimerInterrupt`.

We use objects from the touch screen library and the TFT library, named `ts` and `tft`, respectively. The other libraries are used internally by the TS and TFT.

We also use a timer object named `ITimer1` in the advanced version.

5.2 Inital Source Code

This was the initial implementation of the power timer source code. It has no touch debounce and does not use timer interrupts.

It has a problem that when the add-minutes button is pressed, more often than not, the press will be detected and acted upon twice, and hence will add more time than desired.

It also has poor response to touches due to the `delay()` calls used to control timing countdowns.

Further, the names of some of the functions for displaying screen elements don't really make sense and the flow of control needs to be reorganized.

```
// Timed power controlling device firmware.
// Oct. 2024 modified by Mark G.
// Added RESET, status line, 1, 2 and 3 hours preset buttons,
// add 5 minutes button, as well as modded the ON/OFF switch.
// Comments by me or modified from the original are marked with 'mg:',
// sometimes.
//
//====================================
// Original description as of Oct. 2024
//This example implements a simple sliding On/Off button. The example
// demonstrates drawing and touch operations.
//
//Thanks to Adafruit forums member Asteroid for the original sketch!
//

#include <Adafruit_GFX.h>
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_ILI9341.h>
#include <Adafruit_STMPE610.h>

// This is calibration data for the raw touch data to the screen coordinates
#define TS_MINX 150
#define TS_MINY 130
#define TS_MAXX 3800
#define TS_MAXY 4000

#define STMPE_CS 8
Adafruit_STMPE610 ts = Adafruit_STMPE610(STMPE_CS);
#define TFT_CS 10
#define TFT_DC 9
Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC);

// Define our power tail pins
#define PT_CONTROL 7

boolean setHoursPlease = false;
int hoursToSet = 0;

boolean PowerOn = false;
int runningTimeMins = 0;   // 0 = no effect, unless counting down
int buttonInterval = 5;   // minutes
int runningTimeSecs = 0;

#define FRAME_X 20
#define FRAME_Y 20
#define FRAME_W 200
#define FRAME_H 50
#define FRAME_GAP 5
```

```
#define HOURSET_X (FRAME_X + ((FRAME_W + FRAME_GAP) * 1))
#define HOURSET_Y FRAME_Y
#define HOURSET_W (FRAME_W/2)
#define HOURSET_H FRAME_H

#define TWOHOURSET_X HOURSET_X
#define TWOHOURSET_Y (FRAME_Y + (FRAME_H + FRAME_GAP))
#define TWOHOURSET_W HOURSET_W
#define TWOHOURSET_H FRAME_H

#define THREEHOURSET_X HOURSET_X
#define THREEHOURSET_Y (FRAME_Y + ((FRAME_H + FRAME_GAP) * 2))
#define THREEHOURSET_W HOURSET_W
#define THREEHOURSET_H FRAME_H

#define STATUSLINE_X FRAME_X
#define STATUSLINE_Y FRAME_Y
#define STATUSLINE_W FRAME_W
#define STATUSLINE_H FRAME_H

#define REDBUTTON_X FRAME_X
#define REDBUTTON_Y (FRAME_Y + ((FRAME_H + FRAME_GAP) * 1))
#define REDBUTTON_W (FRAME_W/2)
#define REDBUTTON_H FRAME_H

#define GREENBUTTON_X (REDBUTTON_X + REDBUTTON_W)
#define GREENBUTTON_Y (FRAME_Y + ((FRAME_H + FRAME_GAP) * 1))
#define GREENBUTTON_W (FRAME_W/2)
#define GREENBUTTON_H FRAME_H

#define TIMERLINE_X FRAME_X
#define TIMERLINE_Y (FRAME_Y + ((FRAME_H + FRAME_GAP) * 2))
#define TIMERLINE_W (FRAME_W/2)
#define TIMERLINE_H FRAME_H

#define PLUS30MIN_X (FRAME_X + TIMERLINE_W)
#define PLUS30MIN_Y (FRAME_Y + ((FRAME_H + FRAME_GAP) * 2))
#define PLUS30MIN_W (FRAME_W/2)
#define PLUS30MIN_H FRAME_H

#define RESETLINE_X FRAME_X
#define RESETLINE_Y (FRAME_Y + ((FRAME_H + FRAME_GAP) * 3))
#define RESETLINE_W FRAME_W
#define RESETLINE_H FRAME_H


void drawFrame()
{
```

```
  tft.drawRect(0, 0, 320, 240, ILI9341_BLACK);
}

void redBtn()
{
  // mg: indicate with a bright red rectangle that power is off.
  tft.fillRect(REDBUTTON_X, REDBUTTON_Y, REDBUTTON_W, REDBUTTON_H, ILI9341_RED);
  // mg: set green the power is on rectangle to black.
  tft.fillRect(GREENBUTTON_X, GREENBUTTON_Y, GREENBUTTON_W, GREENBUTTON_H, ILI9341_BLACK);
  drawFrame();
  // mg: insert the text on the power on indicator instructing the
  // user to tap (the text) to turn the device on.
  tft.setCursor(GREENBUTTON_X + 3 , GREENBUTTON_Y + (GREENBUTTON_H/3));
  tft.setTextColor(ILI9341_WHITE);
  tft.setTextSize(2);
  // mg: since the power is set to off, show a command button to turn the
  // power on.
  tft.println("TURN ON");
  // mg: We have turned the power off by pressing the 'turn off' text. Set the toggle.
  PowerOn = false;

  //runningTimeMins = 0;
  showTimer();
}

void greenBtn()
{
  // mg: indicate with a bright green rectangle that power is on.
  tft.fillRect(GREENBUTTON_X, GREENBUTTON_Y, GREENBUTTON_W, GREENBUTTON_H, ILI9341_GREEN);
  // mg: set red the power is off rectangle to black.
  tft.fillRect(REDBUTTON_X, REDBUTTON_Y, REDBUTTON_W, REDBUTTON_H, ILI9341_BLACK);
  drawFrame();
  // mg: insert the text on the power off indicator instructing the
  // user to tap (the text) to turn the device off.
  tft.setCursor(REDBUTTON_X + 3, REDBUTTON_Y + (REDBUTTON_H/3));
  tft.setTextColor(ILI9341_WHITE);
  tft.setTextSize(2);
  // mg: since the power is set to on, show a command button to turn the
  // power off.
  tft.println("TURN OFF");
  // mg: We have turned the power on by pressing the 'turn on' text. Set the toggle.
  PowerOn = true;
}

void addMinsButton(int minutes)
{
  tft.fillRect(PLUS30MIN_X, PLUS30MIN_Y, PLUS30MIN_W, PLUS30MIN_H, ILI9341_BLUE);
  tft.fillRect(TIMERLINE_X, TIMERLINE_Y, TIMERLINE_W, TIMERLINE_H, ILI9341_BLACK);
  drawFrame();
```

```
  tft.setCursor(PLUS30MIN_X, PLUS30MIN_Y + (PLUS30MIN_H/3));
  tft.setTextColor(ILI9341_WHITE);
  tft.setTextSize(2);
  tft.print(" +");
  tft.print(minutes);
  tft.println(" min");
}


void showStatus()
{
  //tft.fillRect(GREENBUTTON_X, GREENBUTTON_Y, GREENBUTTON_W, GREENBUTTON_H, ILI9341_GREEN);
  tft.fillRect(STATUSLINE_X, STATUSLINE_Y, STATUSLINE_W, STATUSLINE_H, ILI9341_BLACK);
  drawFrame();
  tft.setTextSize(3);
  tft.setCursor(STATUSLINE_X + 3 , STATUSLINE_Y + (STATUSLINE_H/3));
  tft.println("POWER:    ");

  tft.setTextColor(ILI9341_WHITE);
  tft.setCursor(STATUSLINE_X + 3 , STATUSLINE_Y + (STATUSLINE_H/3));
    if (PowerOn)
    {
      tft.println("POWER: ON");
    }
    else //Power is off (PowerOn == false)
    {
      tft.println("POWER: OFF");
    }
}  // end showStatus

void resetButton()
{
  //tft.fillRect(GREENBUTTON_X, GREENBUTTON_Y, GREENBUTTON_W, GREENBUTTON_H, ILI9341_GREEN);
  tft.fillRect(RESETLINE_X, RESETLINE_Y, RESETLINE_W, RESETLINE_H, ILI9341_YELLOW);
  drawFrame();
  tft.setTextSize(3);
  tft.setTextColor(ILI9341_BLACK);
  tft.setCursor(RESETLINE_X + (RESETLINE_W/4) , RESETLINE_Y + (RESETLINE_H/3));
  tft.println("RESET");
}  // end resetButton


void showHours() {

  tft.fillRect(HOURSET_X, HOURSET_Y, HOURSET_W, HOURSET_H, ILI9341_CYAN);
  tft.fillRect(TWOHOURSET_X, TWOHOURSET_Y, TWOHOURSET_W, TWOHOURSET_H, ILI9341_CYAN);
  tft.fillRect(THREEHOURSET_X, THREEHOURSET_Y, THREEHOURSET_W, THREEHOURSET_H, ILI9341_CYAN);
  drawFrame();
```

```
  tft.setTextSize(3);
  tft.setTextColor(ILI9341_BLACK);
  for (int i = 1; i <= 3; i++) {
    tft.setCursor(HOURSET_X + 30 , (HOURSET_Y + ((HOURSET_H + FRAME_GAP) * (i-1))) + (HOURSET_H/3
    tft.print(i);
    tft.println(" h");
  }
}


void setHours(int hours) {

  runningTimeMins = hours * 60;
  greenBtn();
  digitalWrite(PT_CONTROL, HIGH);  // sets power on
  showStatus();
  showTimer();
}

void showTimer()
{
  addMinsButton(buttonInterval);
  //tft.fillRect(GREENBUTTON_X, GREENBUTTON_Y, GREENBUTTON_W, GREENBUTTON_H, ILI9341_GREEN);
  tft.fillRect(TIMERLINE_X, TIMERLINE_Y, TIMERLINE_W, TIMERLINE_H, ILI9341_BLACK);
  drawFrame();
  tft.setTextSize(3);
  tft.setTextColor(ILI9341_WHITE);
  tft.setCursor(TIMERLINE_X + 3 , TIMERLINE_Y + (TIMERLINE_H/3));
  tft.print(runningTimeMins);
  tft.setTextSize(2);
  tft.println(" min");
}  // end showTimer


void checkTimer() {
  delay(300);
  int prevMins = runningTimeMins;
  if (runningTimeMins > 0) {
    runningTimeSecs = runningTimeSecs + 1;
    if (runningTimeSecs == 120) {
      runningTimeMins = runningTimeMins - 1;
      Serial.print("CT: New running time (mins): ");
      Serial.println(runningTimeMins);
      runningTimeSecs = 0;
    }
    Serial.print("CT: New running time (secs): ");
    Serial.println(runningTimeSecs);
    if (runningTimeSecs % 2 == 0) {
      showTimer();
```

```
    }
  }
  if (runningTimeMins == 0) {
    if (prevMins > 0) {
      redBtn();
      digitalWrite(PT_CONTROL, LOW);  // sets power off
      showStatus();
    }
  }
}

void setup(void)
{
  // Power tail pin(s).  Uses ground and pin 7 (soldered)
  pinMode(PT_CONTROL, OUTPUT);

  Serial.begin(9600);
  tft.begin();
  if (!ts.begin()) {
    Serial.println("Unable to start touchscreen.");
  }
  else {
    Serial.println("Touchscreen started.");
  }

  tft.fillScreen(ILI9341_BLACK);
  // origin = left,top landscape (USB left upper)
  tft.setRotation(1);
  redBtn();
  showStatus();
  showTimer();
  resetButton();
  showHours();
}

void loop()
{
  // See if there's any  touch data for us
  if (!ts.bufferEmpty())
  {
    // Retrieve a point
    TS_Point p = ts.getPoint();
    // Scale using the calibration #'s
    // and rotate coordinate system
    p.x = map(p.x, TS_MINY, TS_MAXY, 0, tft.height());
    p.y = map(p.y, TS_MINX, TS_MAXX, 0, tft.width());
    int y = tft.height() - p.x;
    int x = p.y;
```

```
if((x > PLUS30MIN_X) && (x < (PLUS30MIN_X + PLUS30MIN_W))) {
    if ((y > PLUS30MIN_Y) && (y <= (PLUS30MIN_Y + PLUS30MIN_H))) {
      Serial.println("+min btn hit");
      runningTimeMins = runningTimeMins + buttonInterval;
      if (runningTimeMins > 1000) {
        runningTimeMins = 990;
      }
      Serial.print("New running time (mins): ");
      Serial.println(runningTimeMins);
      greenBtn();
      digitalWrite(PT_CONTROL, HIGH);  // sets power on
      showStatus();
    }
}

// mg: look to see if any of the hourly present buttons were
// pressed.  Set the number of hours and signal to set hours.
if((x > HOURSET_X) && (x < (HOURSET_X + HOURSET_W))) {
    if ((y > HOURSET_Y) && (y <= (HOURSET_Y + HOURSET_H))) {
      Serial.println("+ 1 hour button hit");
      setHoursPlease = true;
      hoursToSet = 1;
    }
}
// mg: one could write a loop around these three presets and allow for varying the
// number of presets.  Would require top, left, height and width for the
// containing rectangle, as well as preset increment, and rows/columns.  And so on.
if((x > TWOHOURSET_X) && (x < (TWOHOURSET_X + TWOHOURSET_W))) {
    if ((y > TWOHOURSET_Y) && (y <= (TWOHOURSET_Y + TWOHOURSET_H))) {
      Serial.println("+ 2 hour button hit");
      setHoursPlease = true;
      hoursToSet = 2;
    }
}
if((x > THREEHOURSET_X) && (x < (THREEHOURSET_X + THREEHOURSET_W))) {
    if ((y > THREEHOURSET_Y) && (y <= (THREEHOURSET_Y + THREEHOURSET_H))) {
      Serial.println("+ 3 hour button hit");
      setHoursPlease = true;
      hoursToSet = 3;
    }
}

// mg: see if any of the presets for hourly runtime were pressed.
// Timer starts immeditaly and power is turned on (in setHours).
if (setHoursPlease) {
  setHoursPlease = false;
  setHours(hoursToSet);
}
```

```
    // Only checking for off button if power is actually on.
    if (PowerOn)
    {
      if((x > REDBUTTON_X) && (x < (REDBUTTON_X + REDBUTTON_W))) {
        if ((y > REDBUTTON_Y) && (y <= (REDBUTTON_Y + REDBUTTON_H))) {
          Serial.println("Off btn hit");
          redBtn();
          digitalWrite(PT_CONTROL, LOW);  // sets power off
          showStatus();
        }
      }
    }
    else //Power is off (PowerOn == false)
    {
      if((x > GREENBUTTON_X) && (x < (GREENBUTTON_X + GREENBUTTON_W))) {
        if ((y > GREENBUTTON_Y) && (y <= (GREENBUTTON_Y + GREENBUTTON_H))) {
          Serial.println("Green btn hit");
          greenBtn();
          digitalWrite(PT_CONTROL, HIGH);  // sets power on
          showStatus();
        }
      }
    }

    // mg: Always check for reset button
    if((x > RESETLINE_X) && (x < (RESETLINE_X + RESETLINE_W))) {
      if ((y > RESETLINE_Y) && (y <= (RESETLINE_Y + RESETLINE_H))) {
        Serial.println("Reset button hit");
        runningTimeMins = 0;
        redBtn();
        digitalWrite(PT_CONTROL, LOW);  // sets power off
        showStatus();
        showTimer();

      }
    }

    Serial.println(PowerOn);
  }
  if (PowerOn) {
    // mg: this delay is added to a further (300 currently) delay
    // in the checkTimer to act as a half second counter per loop.
    // This is a crappy timing method, but it works.
    // Button presses take a half second or so to register as a
    // consequence.  Annoying, but easy to cope with.
    delay(200);
    checkTimer();
  }
}
```

5.3 Advanced Source Code

This iteration of the source code uses a timer interrupt to stabilize the timing functions and implements a usable debounce for touch screen presses. The result is a more responsive interface to presses, as well as fixing the multiple minute additions problem.

```
// Timed power controlling device firmware.
// Oct. 2024 modified by Mark G.
// Added RESET, status line, 1, 2 and 3 hours preset buttons,
// add 5 minutes button, as well as modded the ON/OFF switch.
// Comments by me or modified from the original are marked with 'mg:',
// sometimes.
//
//====================================
// Original description as of Oct. 2024
//This example implements a simple sliding On/Off button. The example
// demonstrates drawing and touch operations.
//
//Thanks to Adafruit forums member Asteroid for the original sketch!
//

// Select the timers you're using, here ITimer1
// The actual variable name is also ITimer1 and is
// set in the two includes beneath these defines.
#define USE_TIMER_1     true
#define USE_TIMER_2     false
#define USE_TIMER_3     false
#define USE_TIMER_4     false
#define USE_TIMER_5     false

#define TIMER_INTERVAL_MS        1000L

#include "TimerInterrupt.h"
#include "ISR_Timer.h"

#include <Adafruit_GFX.h>
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_ILI9341.h>
#include <Adafruit_STMPE610.h>

// This is calibration data for the raw touch data to the screen coordinates
#define TS_MINX 150
#define TS_MINY 130
#define TS_MAXX 3800
#define TS_MAXY 4000

#define STMPE_CS 8
Adafruit_STMPE610 ts = Adafruit_STMPE610(STMPE_CS);
#define TFT_CS 10
#define TFT_DC 9
Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC);

// Define our power tail pin number which is an output pin.
#define PT_CONTROL 7
```

```
boolean setHoursPlease = false;
int hoursToSet = 0;

boolean PowerOn = false;  // global power state variable.
int runningTimeMins = 0;  // 0 = no effect, unless counting down.
int buttonInterval = 5;   // minutes
int runningTimeSecs = 0;  // used in our countdown/timing routine.

// This is to blink our timer box whenever these differ by 1 in checkTimer.
// No longer in use, as our blink is now naturally done by the
// 1 second timer routine.
int prevSecs = runningTimeSecs;

// Useful anchor values for placement of UI rectangles.
#define FRAME_X 20
#define FRAME_Y 20
#define FRAME_W 200
#define FRAME_H 50
#define FRAME_GAP 5

// Preset buttons (3)
#define HOURSET_X (FRAME_X + ((FRAME_W + FRAME_GAP) * 1))
#define HOURSET_Y FRAME_Y
#define HOURSET_W (FRAME_W/2)
#define HOURSET_H FRAME_H

#define TWOHOURSET_X HOURSET_X
#define TWOHOURSET_Y (FRAME_Y + (FRAME_H + FRAME_GAP))
#define TWOHOURSET_W HOURSET_W
#define TWOHOURSET_H FRAME_H

#define THREEHOURSET_X HOURSET_X
#define THREEHOURSET_Y (FRAME_Y + ((FRAME_H + FRAME_GAP) * 2))
#define THREEHOURSET_W HOURSET_W
#define THREEHOURSET_H FRAME_H

// Status line location/size.
#define STATUSLINE_X FRAME_X
#define STATUSLINE_Y FRAME_Y
#define STATUSLINE_W FRAME_W
#define STATUSLINE_H FRAME_H

#define REDBUTTON_X FRAME_X
#define REDBUTTON_Y (FRAME_Y + ((FRAME_H + FRAME_GAP) * 1))
#define REDBUTTON_W (FRAME_W/2)
#define REDBUTTON_H FRAME_H

#define GREENBUTTON_X (REDBUTTON_X + REDBUTTON_W)
```

```
#define GREENBUTTON_Y (FRAME_Y + ((FRAME_H + FRAME_GAP) * 1))
#define GREENBUTTON_W (FRAME_W/2)
#define GREENBUTTON_H FRAME_H

#define TIMERLINE_X FRAME_X
#define TIMERLINE_Y (FRAME_Y + ((FRAME_H + FRAME_GAP) * 2))
#define TIMERLINE_W (FRAME_W/2)
#define TIMERLINE_H FRAME_H

#define PLUS30MIN_X (FRAME_X + TIMERLINE_W)
#define PLUS30MIN_Y (FRAME_Y + ((FRAME_H + FRAME_GAP) * 2))
#define PLUS30MIN_W (FRAME_W/2)
#define PLUS30MIN_H FRAME_H

#define RESETLINE_X FRAME_X
#define RESETLINE_Y (FRAME_Y + ((FRAME_H + FRAME_GAP) * 3))
#define RESETLINE_W FRAME_W
#define RESETLINE_H FRAME_H


void refreshScreen() {
  // This function displays all elements of the user interface.
  // They are:
  //   - place a black framed rectangle around the border of the screen.
  //   - display the status bar.
  //   - depending of the power state, display a on/off indicator/control bar.
  //   - display timer elements on the timer bar.
  //   - display a reset button.
  //   - display hourly preset buttons.
  tft.drawRect(0, 0, 320, 240, ILI9341_BLACK);
  // Display each screen object.
  showStatus();
  if (PowerOn) {
    greenPowerOnWithOffButton();
  } else {
    redPowerOffWithOnButton();
  }
  showTimer();
  resetButton();
  showHours();
}


void turnPowerOff() {
  // This function is called to perform all power off actions.
  // Toggle PIN and set power off state, then update our screen.
  digitalWrite(PT_CONTROL, LOW);  // sets power off
  PowerOn = false;
  refreshScreen();
```

```
}


void turnPowerOn() {
  // This function is called to perform all power on actions.
  // Toggle PIN and set power on state, then update our screen.
  digitalWrite(PT_CONTROL, HIGH);  // sets power on
  PowerOn = true;
  refreshScreen();
}


void redPowerOffWithOnButton() {
  // This function displays a powered off state with a red rectangle
  // indicator.  It then displays a pressable 'TURN ON' area of
  // white text on black for toggling on power.

  // mg: indicate with a bright red rectangle that power is off.
  tft.fillRect(REDBUTTON_X, REDBUTTON_Y, REDBUTTON_W, REDBUTTON_H, ILI9341_RED);
  // mg: set the green power-is-on indicator rectangle to black, ready
  // for the white text.
  tft.fillRect(GREENBUTTON_X, GREENBUTTON_Y, GREENBUTTON_W, GREENBUTTON_H, ILI9341_BLACK);

  // mg: since the power is currently off, show a command button to turn the
  // power on.
  tft.setCursor(GREENBUTTON_X + 3 , GREENBUTTON_Y + (GREENBUTTON_H/3));
  tft.setTextColor(ILI9341_WHITE);
  tft.setTextSize(2);
  tft.println("TURN ON");
}

void greenPowerOnWithOffButton() {
  // This function displays a powered on state with a green rectangle
  // indicator.  It then displays a pressable 'TURN OFF' area of
  // white text on black for toggling off power.

  // mg: indicate with a bright green rectangle that power is on.
  tft.fillRect(GREENBUTTON_X, GREENBUTTON_Y, GREENBUTTON_W, GREENBUTTON_H, ILI9341_GREEN);
  // mg: set the red power-is-off indicator rectangle to black, ready
  // for the white text.
  tft.fillRect(REDBUTTON_X, REDBUTTON_Y, REDBUTTON_W, REDBUTTON_H, ILI9341_BLACK);

  // mg: since the power is currently on, show a command button to turn the
  // power off.
  tft.setCursor(REDBUTTON_X + 3, REDBUTTON_Y + (REDBUTTON_H/3));
  tft.setTextColor(ILI9341_WHITE);
  tft.setTextSize(2);
  tft.println("TURN OFF");
}
```

```
void addMinsButton(int minutes) {
  // This function displays a pressable screen area that will add
  // time to the current number of minutes that power is to remain
  // on, when touched.
  // The number of minutes added per press is passed as an argument.
  // It will be black text on a blue background.
  tft.fillRect(PLUS30MIN_X, PLUS30MIN_Y, PLUS30MIN_W, PLUS30MIN_H, ILI9341_BLUE);
  tft.fillRect(TIMERLINE_X, TIMERLINE_Y, TIMERLINE_W, TIMERLINE_H, ILI9341_BLACK);
  tft.setCursor(PLUS30MIN_X, PLUS30MIN_Y + (PLUS30MIN_H/3));
  tft.setTextColor(ILI9341_WHITE);
  tft.setTextSize(2);
  tft.print(" +");
  tft.print(minutes);
  tft.println(" min");
}


void showStatus() {
  // Display the current state of the power controller.
  // It is white text on a black background.
  // This is a black background rectangle at the top of the
  // display with large print indicating ON/OFF status.
  // Note the green rectangle indicator is also a visual clue to
  // power on, and so the red rectangle acts as an indicator of power off.
  tft.fillRect(STATUSLINE_X, STATUSLINE_Y, STATUSLINE_W, STATUSLINE_H, ILI9341_BLACK);
  tft.setTextSize(3);
  tft.setCursor(STATUSLINE_X + 3 , STATUSLINE_Y + (STATUSLINE_H/3));
  tft.println("POWER:    ");

  tft.setTextColor(ILI9341_WHITE);
  tft.setCursor(STATUSLINE_X + 3 , STATUSLINE_Y + (STATUSLINE_H/3));
    if (PowerOn)
    {
      tft.println("POWER: ON");
    }
    else //Power is off (PowerOn == false)
    {
      tft.println("POWER: OFF");
    }
}  // end showStatus


void resetButton() {
  // Display our large yellow reset button with black text.
  tft.fillRect(RESETLINE_X, RESETLINE_Y, RESETLINE_W, RESETLINE_H, ILI9341_YELLOW);
  tft.setTextSize(3);
  tft.setTextColor(ILI9341_BLACK);
```

```
  tft.setCursor(RESETLINE_X + (RESETLINE_W/4) , RESETLINE_Y + (RESETLINE_H/3));
  tft.println("RESET");
}  // end resetButton


void showHours() {
  // Display our three hourly preset buttons as black text over a cyan
  // background.
  tft.fillRect(HOURSET_X, HOURSET_Y, HOURSET_W, HOURSET_H, ILI9341_CYAN);
  tft.fillRect(TWOHOURSET_X, TWOHOURSET_Y, TWOHOURSET_W, TWOHOURSET_H, ILI9341_CYAN);
  tft.fillRect(THREEHOURSET_X, THREEHOURSET_Y, THREEHOURSET_W, THREEHOURSET_H, ILI9341_CYAN);

  tft.setTextSize(3);
  tft.setTextColor(ILI9341_BLACK);
  for (int i = 1; i <= 3; i++) {
    tft.setCursor(HOURSET_X + 30 , (HOURSET_Y + ((HOURSET_H + FRAME_GAP) * (i-1))) + (HOURSET_H/3
    tft.print(i);
    tft.println(" h");
  }  // i loop
} // showHours


void setHours(int hours) {
  // Take the number of hours provided, turn it into minutes,
  // and then turn the power on.
  runningTimeMins = hours * 60;
  turnPowerOn();
}  // setHours


void showTimer() {
  // This function displays the timer line of the user interface.
  // It places the "add time" button first, and then draws the
  // cuurent number of minutes of time remaining.

  addMinsButton(buttonInterval);
  // Fill the time remaining rectangle with black so our white text
  // has good contrast/
  tft.fillRect(TIMERLINE_X, TIMERLINE_Y, TIMERLINE_W, TIMERLINE_H, ILI9341_BLACK);

  // Multiply the default text size by three, and set the text colour to white.
  tft.setTextSize(3);
  tft.setTextColor(ILI9341_WHITE);
  // Place the text cursor slightly indented (+ 3), and a third of the way into the
  // time remaining rectangle.
  tft.setCursor(TIMERLINE_X + 3 , TIMERLINE_Y + (TIMERLINE_H/3));
  // Print the number of minutes and the text 'min'.  Sometimes this overflows
  // into the add minutes button.
  tft.print(runningTimeMins);
```

```
  tft.setTextSize(2);
  tft.println(" min");
}  // end showTimer


void TimerHandler() {
  // This function is called after each TIMER_INTERVAL_MS,
  // 1000 milliseconds for our use case.  It is attached
  // to the timer interrupt (ITimer1) in the setup() function.

  //Serial.println("TimerHandler: called");
  if (runningTimeMins > 0) {
    if (PowerOn) {
      prevSecs = runningTimeSecs;
      runningTimeSecs = runningTimeSecs + 1;
      // This call effectively blinks the timer every second.
      showTimer();
    } // PowerOn
  } // running time > 0

}  // TimerHandler


void checkTimer() {
  // This is called at the end of the main loop to see if
  // the timer has expired.

  int prevMins = runningTimeMins;

  if (runningTimeMins > 0) {

    if (runningTimeSecs == 60) {
      runningTimeMins = runningTimeMins - 1;
      Serial.print("CT: New running time (mins): ");
      Serial.println(runningTimeMins);
      runningTimeSecs = 0;
    }
  }

  if (runningTimeMins == 0) {
    if (prevMins > 0) {
      turnPowerOff();
    }
  }
}


void setup(void)
{
```

```
  // Power tail pin(s).  Uses ground and pin 7 (soldered)
  pinMode(PT_CONTROL, OUTPUT);

  // Init timer ITimer1
  ITimer1.init();

  Serial.begin(9600);
  tft.begin();
  if (!ts.begin()) {
    Serial.println("Unable to start touchscreen.");
  }
  else {
    Serial.println("Touchscreen started.");
  }
  // Blank the screen (all black).
  tft.fillScreen(ILI9341_BLACK);
  // origin = left,top landscape (USB left upper)
  tft.setRotation(1);
  // Show the user interface.
  refreshScreen();

  // Setup our timer routine for a 1 second interval.
  // Interval in unsigned long millisecs.
  if (ITimer1.attachInterruptInterval(TIMER_INTERVAL_MS, TimerHandler))
    Serial.println("Starting ITimer OK, millis() = " + String(millis()));
  else
    Serial.println("Can't set ITimer. Select another freq. or timer");

  // Some debug information.
  Serial.print("TS_MINY: ");
  Serial.print(TS_MINY);
  Serial.print(", TS_MAXY: ");
  Serial.print(TS_MAXY);
  Serial.print(", height: ");
  Serial.println(tft.height());

  Serial.print("TS_MINX: ");
  Serial.print(TS_MINX);
  Serial.print(", TS_MAXX: ");
  Serial.print(TS_MAXX);
  Serial.print(", width: ");
  Serial.println(tft.width());

} // setup

// We need this to help with touch debounce.
TS_Point dummy;
```

```
void loop()
{
  // See if there's any touch data for us.
  if (!ts.bufferEmpty())
  {
    // It is required to note that there are spurious touch events
    // occuring some of the time.  Maybe one every 10 seconds or so.
    // Thankfully, they return bogus data which does not match any
    // of our control surfaces.  How long this remains true is
    // unknown.

    // Retrieve a point.  This is the useful data,
    // all the other getPoint() calls are debounce related.
    TS_Point p = ts.getPoint();

    // Scale using the calibration numbers
    // and rotate coordinate system.
    p.x = map(p.x, TS_MINY, TS_MAXY, 0, tft.height());
    p.y = map(p.y, TS_MINX, TS_MAXX, 0, tft.width());

    Serial.print("a: p.x: ");
    Serial.print(p.x);
    Serial.print(", p.y: ");
    Serial.println(p.y);

    int y = tft.height() - p.x;
    int x = p.y;
    Serial.print("x: ");
    Serial.print(x);
    Serial.print(", y: ");
    Serial.print(y);
    Serial.print(", Touch pressure: ");
    Serial.println(p.z);

    // This seems to tbe the best way to debounce the screen touches.
    // See the forums on adafruit.com for more information.  My code
    // differs slightly from the discussion, as I needed the dummy
    // variable instead of using p in the while loop below.
    // It needs another dummy = ts.getPoint(); at the bottom of this
    // loop() function.  Search for debounce to find all required statements
    // to implement a decent debounce.
    while(!ts.bufferEmpty() || ts.touched())
    {
      Serial.println("Buffer not empty or screen touched spin cycle.");
      dummy = ts.getPoint();
    } // debounce spinner

    // This is our add-minutes-button screen area.
    if((x > PLUS30MIN_X) && (x < (PLUS30MIN_X + PLUS30MIN_W))) {
```

```
    if ((y > PLUS30MIN_Y) && (y <= (PLUS30MIN_Y + PLUS30MIN_H))) {
      Serial.println("+min button hit");
      runningTimeMins = runningTimeMins + buttonInterval;
      if (runningTimeMins > 1000) {
        runningTimeMins = 990;
      }
      Serial.print("New running time (mins): ");
      Serial.println(runningTimeMins);
      turnPowerOn();
    }
} // add minutes hitbox

// mg: look to see if any of the hourly present buttons were
// pressed.  Set the number of hours and signal to set hours.
if((x > HOURSET_X) && (x < (HOURSET_X + HOURSET_W))) {
    if ((y > HOURSET_Y) && (y <= (HOURSET_Y + HOURSET_H))) {
      Serial.println("+ 1 hour button hit");
      setHoursPlease = true;
      hoursToSet = 1;
    }
} // 1 hour hitbox
// mg: one could write a loop around these three presets and allow for varying the
// number of presets.  Would require top, left, height and width for the
// containing rectangle, as well as preset increment, and rows/columns.  And so on.
if((x > TWOHOURSET_X) && (x < (TWOHOURSET_X + TWOHOURSET_W))) {
    if ((y > TWOHOURSET_Y) && (y <= (TWOHOURSET_Y + TWOHOURSET_H))) {
      Serial.println("+ 2 hour button hit");
      setHoursPlease = true;
      hoursToSet = 2;
    }
} // 2 hour hitbox
if((x > THREEHOURSET_X) && (x < (THREEHOURSET_X + THREEHOURSET_W))) {
    if ((y > THREEHOURSET_Y) && (y <= (THREEHOURSET_Y + THREEHOURSET_H))) {
      Serial.println("+ 3 hour button hit");
      setHoursPlease = true;
      hoursToSet = 3;
    }
} // 3 hour hitbox

// mg: see if any of the presets for hourly runtime want us to do something.
// Timer starts immediately and power is turned on (in setHours).
// setHours calls turnPowerOn.
if (setHoursPlease) {
  setHoursPlease = false;
  setHours(hoursToSet);
} // set hours requested

// Only checking for off button if power is actually on.
if (PowerOn) {
```

```
      if((x > REDBUTTON_X) && (x < (REDBUTTON_X + REDBUTTON_W))) {
        if ((y > REDBUTTON_Y) && (y <= (REDBUTTON_Y + REDBUTTON_H))) {
          Serial.println("Off button hit");
          turnPowerOff();
        }
      } // off button hitbox

    } else {
      //Power is off (PowerOn == false)
      if((x > GREENBUTTON_X) && (x < (GREENBUTTON_X + GREENBUTTON_W))) {
        if ((y > GREENBUTTON_Y) && (y <= (GREENBUTTON_Y + GREENBUTTON_H))) {
          Serial.println("On button hit");
          turnPowerOn();
        }
      } // on button hit box
    }

    // mg: Always check for reset button
    if((x > RESETLINE_X) && (x < (RESETLINE_X + RESETLINE_W))) {
      if ((y > RESETLINE_Y) && (y <= (RESETLINE_Y + RESETLINE_H))) {
        Serial.println("Reset button hit");
        runningTimeMins = 0;
        turnPowerOff();
      }
    }

    Serial.println(PowerOn);

  }  // end if touchscreen buffer is empty

  // Final try to clear the touch screen buffer for debounce.
  dummy = ts.getPoint();

  // If we are currently powered on, then see if we are using a timer
  // and whether it has expired.
  if (PowerOn) {
    checkTimer();
  } // power on check

} // end loop()
```

6 Conclusion

This project used up several pieces of hardware that were otherwise just sitting in boxes. It just goes to show you that when a new purchase (the e-bike) comes around, you may need to find a novel solution to its usage.

There are likely off-the-shelf products that can do similar functions, but it is getting harder to find such products using internet searches these days.

Much more fun to make your own. Device in action:

Figure 11: Full kit connected to bike

Of course, some space can be found on the wall for mounting the whole system at some time in the future. It's good enough for now.