Pedal-Powered Smart Bike

Team 29

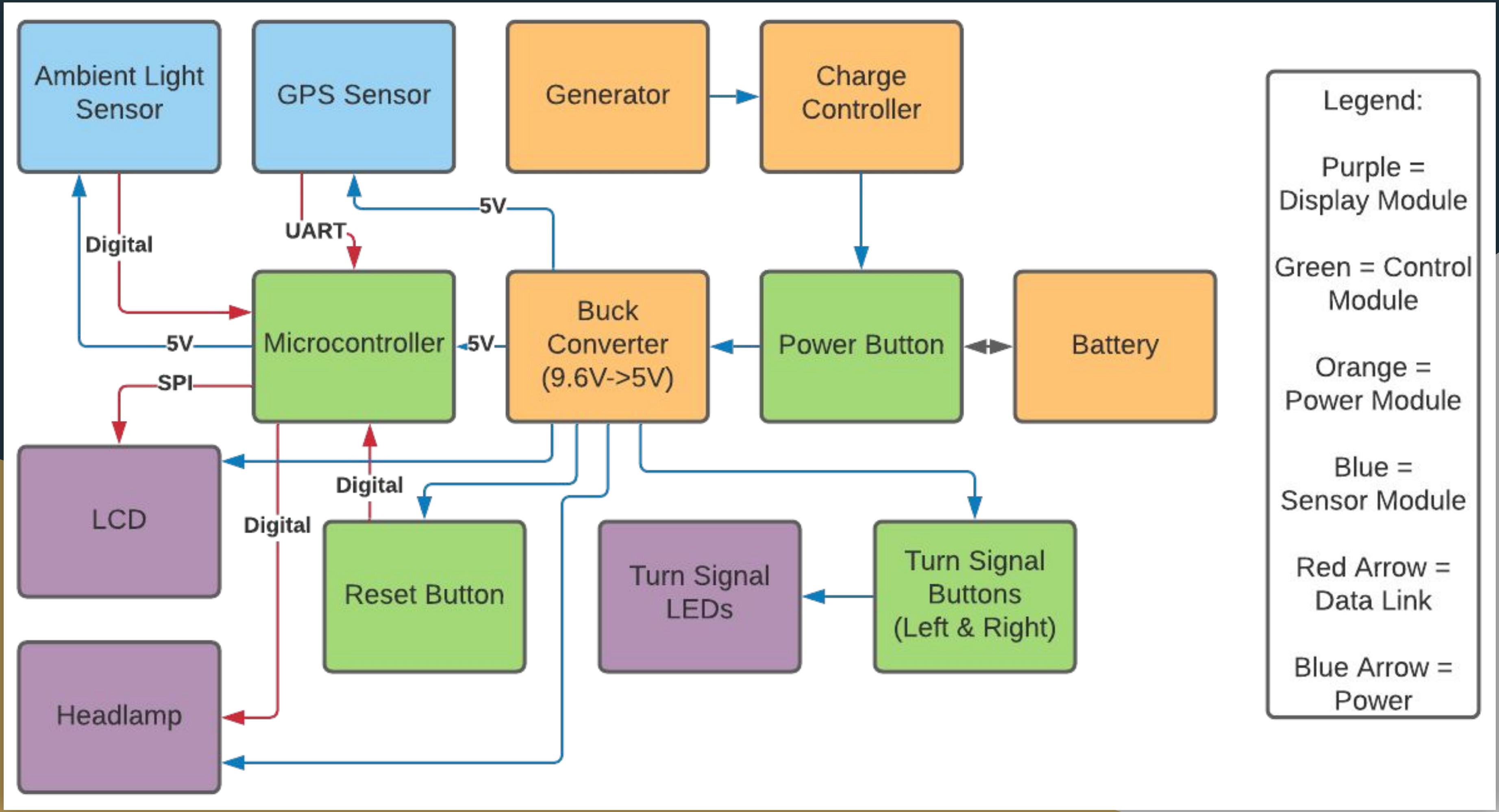
By Alex Sirakides, Karl Kamar and Anshul Desai

Introduction

- Lack of electrical development in marketed bikes
- Need for more bikers in growing cities
- Need to make bike user experience comparable to modern cars
- Need to make bikes more attractive to modern users

High Level Requirements and Objectives

- Provide bikers with Safety Enhancements
- Increase available information to bikers on trips
- Introduce Automation to bike’s controls (ALS)
- Incorporate these features in a self-contained system


Ambient Light Sensor
GPS Sensor
Generator
Charge Controller
Digital
UART
5V
Microcontroller
5V
Buck Converter (9.6V->5V)
Power Button
Battery
5V
SPI
LCD
Digital
Digital
Reset Button
Turn Signal LEDs
Turn Signal Buttons (Left & Right)
Headlamp
Legend:
Purple = Display Module
Green = Control Module
Orange = Power Module
Blue = Sensor Module
Red Arrow = Data Link
Blue Arrow = Power

Calculations Behind Power Supply Design

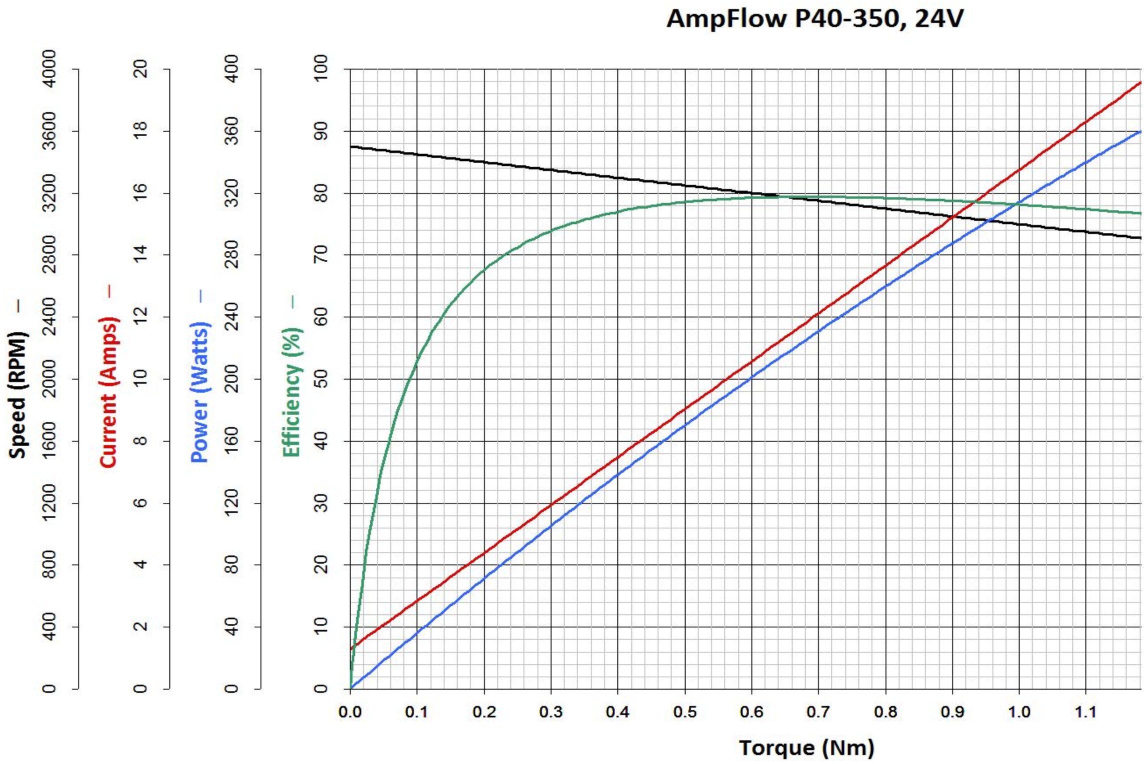
- **Average torque on pedals:**
  Biker Weight x Pedal Length x 2π = 65x9.81 x 0.17 x sin(θ)
  = 108.4 N-m
  >>>>> Drag is too big which makes it impossible to pedal. 0V output.

- **Torque in hinge system scenario:**
  Normal Reaction = Motor Weight = Mass x g = 5.6lb x 0.453592kg/lb x 9.81 = 24.92 N
  Friction = Dynamic Friction Constant x Normal Reaction = 0.5 x 24.92 = 12.46 N
  Torque = Friction x sin(θ) x Motor Brush Length = 12.46 x sin(90) x 1.1in x 0.0254in/m = 0.348 N-m

0.348 N-m corresponds to 124W, 6.6A, 3320 RPM at an Efficiency of 76%:

>>>>> 14.28V output

Motor Chart



AmpFlow P40-350, 24V
Speed (RPM)
0 400 800 1200 1600 2000 2400 2800 3200 3600 4000
Current (Amps)
0 2 4 6 8 10 12 14 16 18 20
Power (Watts)
0 40 80 120 160 200 240 280 320 360 400
Efficiency (%)
0 10 20 30 40 50 60 70 80 90 100
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 1.1
Torque (Nm)

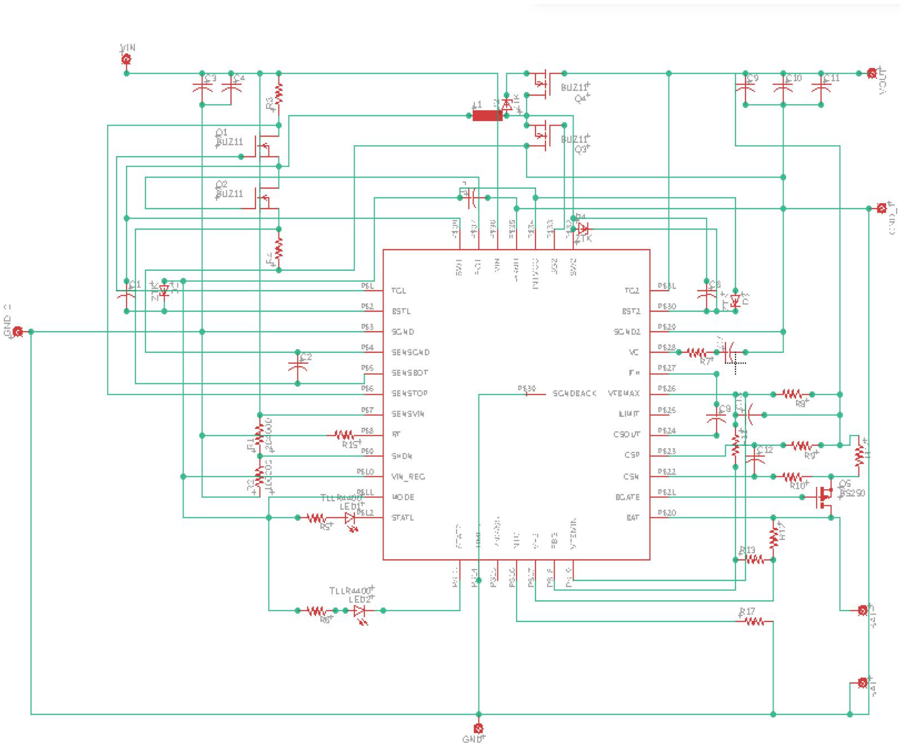
Power Supply Mechanism

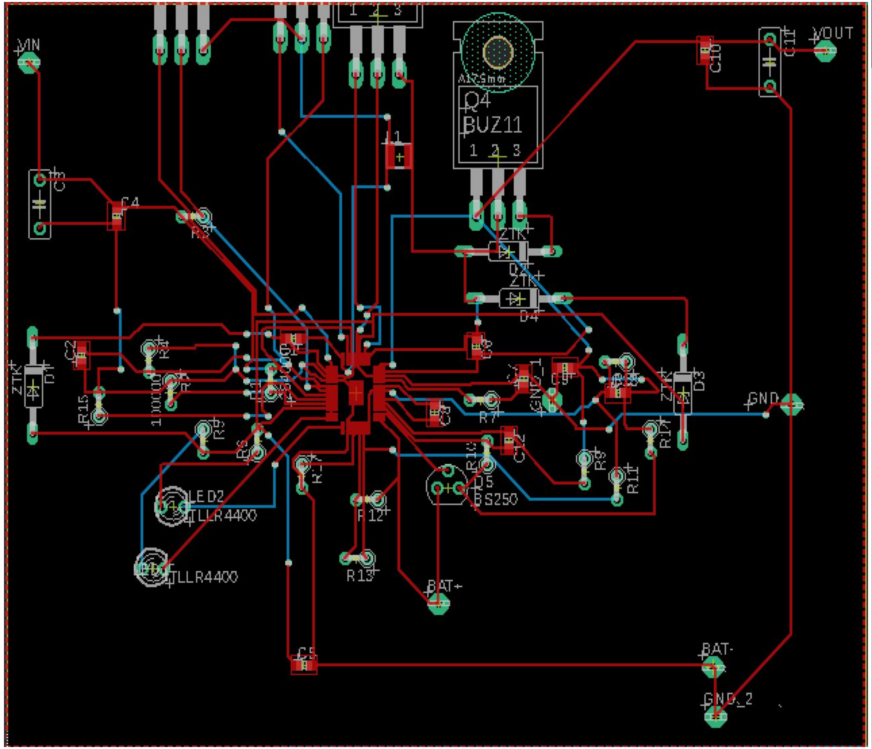
- 24VDC Brush Motor Produces Power
- Metal Cylinder Attached to the Brush
- Friction of Tyre on Cylinder Drives Rotation
- Hinge System Permits Motor Mobility for Safety

Battery

- 12V/2.3Ah vs 8V/3.2Ah
- Power Supply for all the bike's UI
- Lead Acid Rechargeable

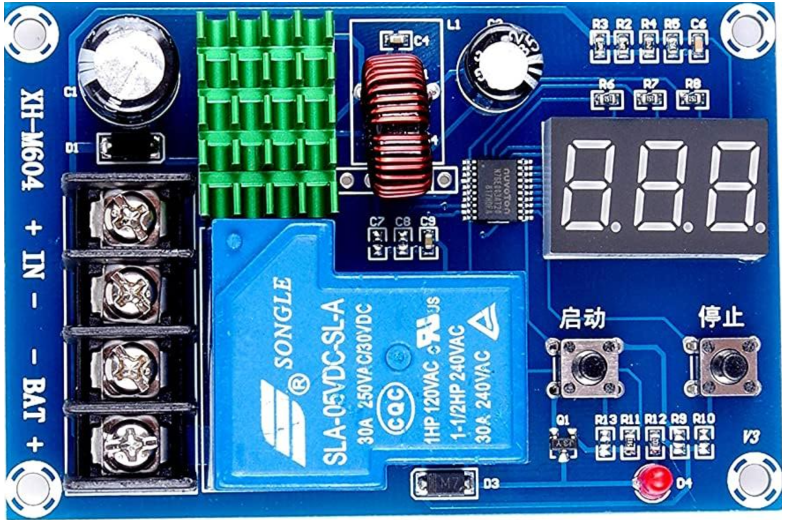
Initial Charge Controller Plan





Charge Controller

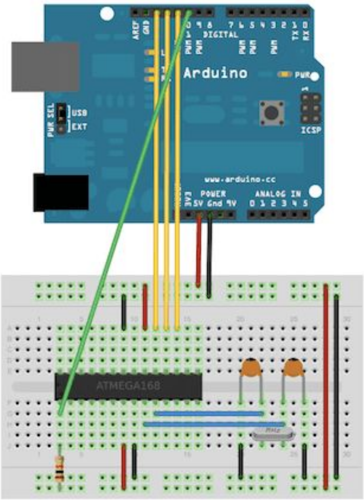
- Controls Flow of Current from Motor to Battery (0.1 x Rated Battery Current)
- Maintains Battery Voltage within a safe range (0.9 x Rated Voltage - 1.2 x Rated Voltage)
- Maintains Charge Flow Direction in one direction
- Based on Relay Switch system that connects/disconnects output from input
- Displays voltage across battery terminals
- Prefabricated vs. Soldered Ourselves
- Diode Added

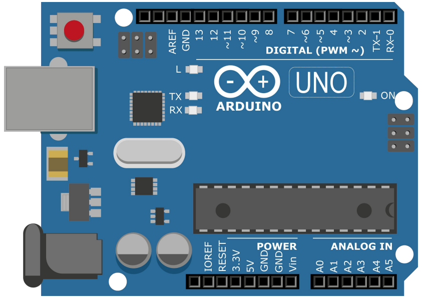


Buck Converter

- 5V/3A output
- Chosen over Linear Voltage Regulator (Loss and Heat)
- Powers MCU, Lights and Display
- Ensures Constant Voltage is provided to DC components

ATMEGA328P Development Process





1) Burn Bootloader to multiple microcontrollers
2) Program microcontroller directly from Arduino Uno

Verification: 2 x pF Capacitors, 1 x 16 MHz Oscillating Crystal, 1 x 5V regulated power source

Removing MicroSD Card Reader

Previous use cases:

- Storing path data every update
- Distance calculation from summed path data
- Average speed calculation from stored path
- Ride time from first and most recent reading
- Potential for reverse geocoding

Issues:

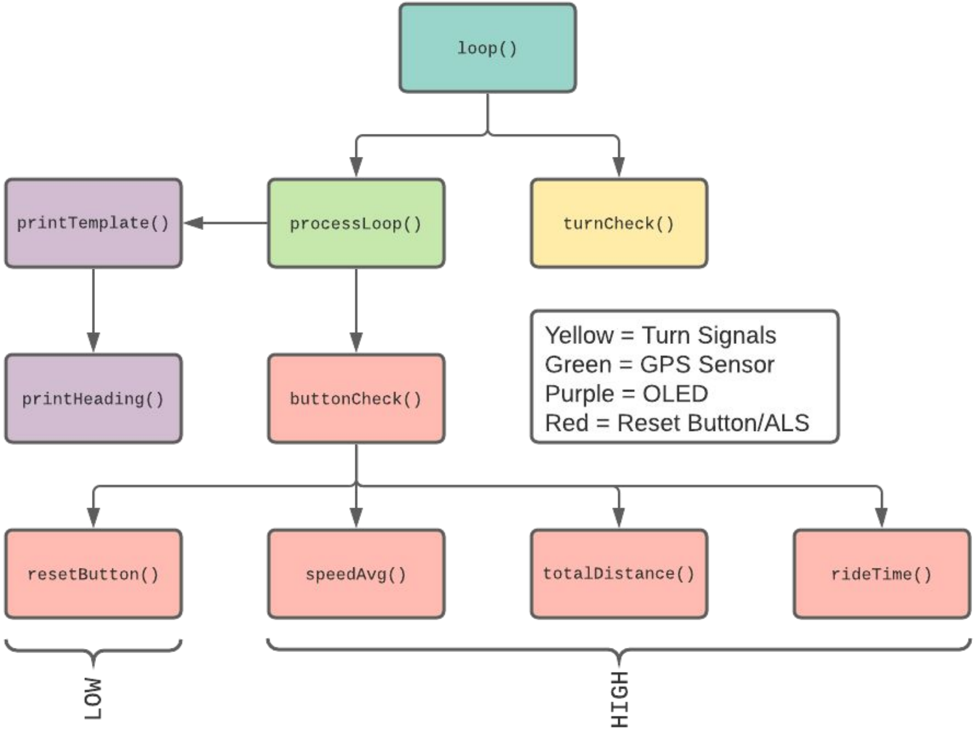
- SD card write would disrupt GPS data, feeding garbage values
- MCU not fast enough to reverse geocode

Solution:

- Utilize MCU EEPROM to save state data on update
- Instantly retrieve saved data on init
- Display relevant lat/long data

Bonus Verification:

- Pins available on PCB for ALS, FET, light and reset switches, no need for redesign!



Function Tree

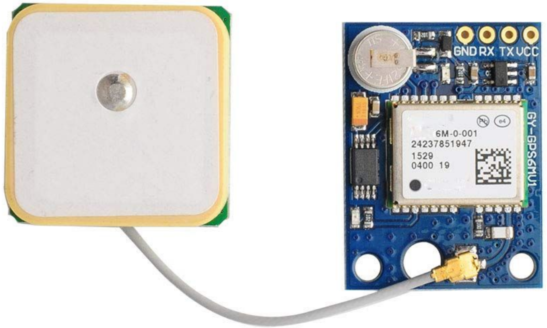
processLoop() Function

Libraries used: NeoSWSerial, NeoGPS

Called on availability of GPS Software Serial object (1 Hz)

Pseudo-algorithm:

1) Read `ALS` and `RESET_BUTTON` state
2) Save GPS data to `fix`
3) Call `buttonCheck()` and `printTemplate()`
4) Save exercise statistics to EEPROM



NEO-6M GPS Sensor

Processing NMEA GPS Data

Verification: Serial monitor of Arduino IDE outputs NMEA sentences

$GPRMC (Min. recommendation for GPS data) ex:

$GPRMC,220516,A,5133.82,N,00042.24,W,173.8,231.8,130694,004.2,W*70

Timestamp Latitude/Longitude Speed (Knots) Datestamp

GPS `fix` updated at beginning of `processLoop()`:

`fix.speed_mph()` - Speed converted from knots to miles per hour

`fix.latitude()`, `fix.longitude()` - Latitude and Longitude returned as floating-point

`fix.heading()` - Calculated heading returned in degrees clockwise from North

etc…

ALSO verifiable from Arduino IDE Serial monitor

printTemplate() Function

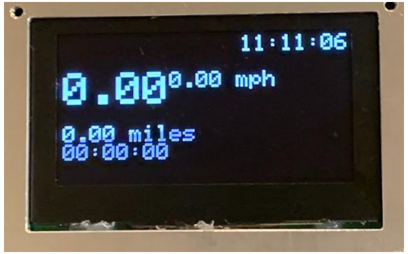
Libraries used: AdafruitGFX, Adafruit SSD1325 Display Driver

Updates with GPS module (1 Hz)

Pseudo-algorithm:

1) Clear display video buffer (prepping cursor)
2) Validate `fix`
    a) Prints with only GPS time fix (ride time increments)
    b) Prints with GPS time and GPS location fix (all stats increment)*
    c) Like (a), default time “6:00:00”, no ride time
3) Build video buffer with variables saved to flash memory
4) Display video buffer

*printHeading() called with `fix.heading()`



(a) Time Fix

(b) Time + Location Fix

Statistics Functions

`speedAvg():`

1. Return (reject) if speed is below 0.5 miles per hour
2. Redistribute average based on current speed, previous speed, previous average, and previous count
3. Increment previous count

`totalDistance():`

1. Return (reject) if speed is below 1.0 miles per hour and location is invalid
2. Calculate square (or Haversine) distance between current and previous location
3. Update previous location to new location

`rideTime():`

1. On init (startup/record), save start time
   a. Pull elapsed time from previous sessions via EEPROM (startup only)
2. Else, calculate difference between current time and start time
   a. Include previous session time (startup only)

resetButton() and buttonCheck()

`resetButton()` simply resets all statistics and sets previous time for `rideTime()` to current (constant)

`buttonCheck()` pseudo-algorithm:

1. If `RESET_BUTTON` is `LOW`, call `resetButton()`
2. Else, call `speedAvg()`, `totalDistance()`, `rideTime()`
3. ALSO, verify state of ALS-FET system (a -> b -> c -> a -> …):
    a. If ALS previous state `LOW`, now `HIGH`, pulse head and tail lights on
    b. Else if ALS previous state `HIGH`, now `LOW`, save start time
    c. Else if ALS previous state `LOW`, still `LOW`, and 5 seconds from start time, pulse head and tail lights off
4. Save previous state of ALS-FET system

Buttons in Circuit

Power Button:

- Mechanical switch disconnecting battery from CC & System
- Mounted above the battery

Reset Button:

- Mounted next to right turn signal
- Active HIGH with pull down resistor

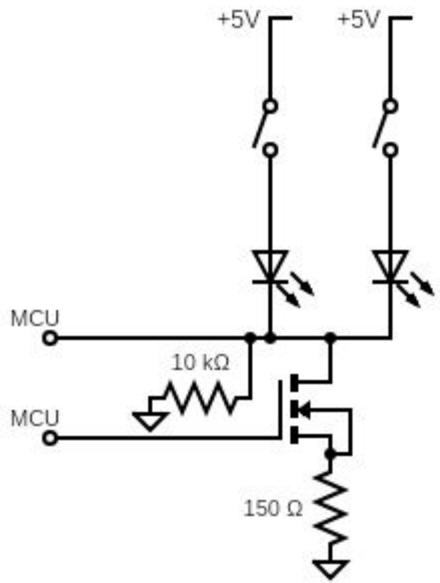
Turn Signals

Turn Signal Buttons

- Active HIGH with pull down resistors
- Cannot provide necessary flashing

N-Channel FET & Duty Cycle

- Flashing driven via the microcontroller
- Allows for any flashing speed
- Allows for pin reduction



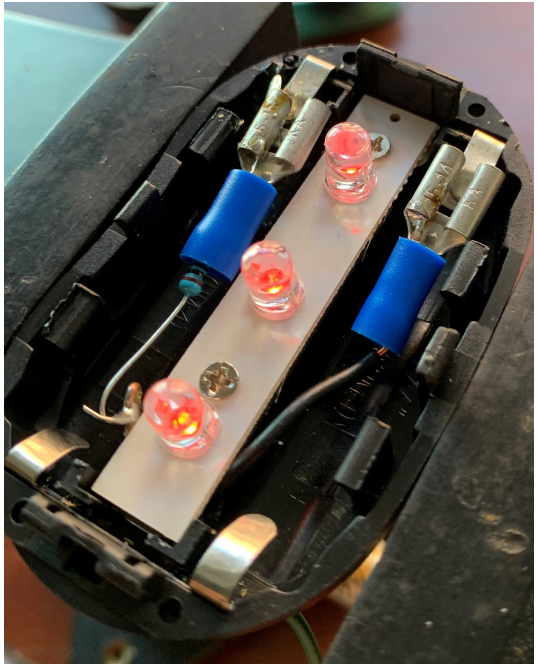
Turn Signals Continued

Turn Signal FSM

- LEDs were 3-state FSMs (Off, On, Blink)
- Could only be toggled mechanically
- Initial State is on

Soldering

- Cannot solder to aluminium
- Additional resistor to offset additional voltage



Head and Tail Lights

LED FSM

- These were 5-state FSMs
- Initiale State was on
- Could be digitally controlled via microcontroller
- Button active low, otherwise high-z
- 2M ohm pull-up resistor

Ambient Light Sensor

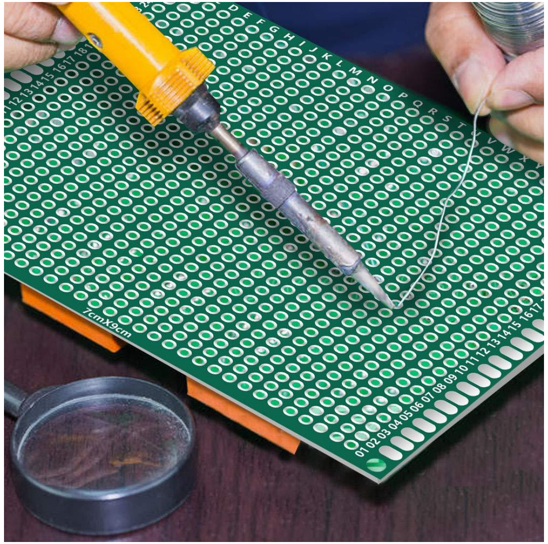
Controls Head & Tail Lights

- Head and tail lights are on when system power is turned on
- If light out -> 5 second delay -> pulse 4x to turn off
- If dark out -> ALS goes HIGH -> pulse to turn on head & tail lights
- When operating off Arduino board power, we had to adjust for a race condition, this went away once integrated

Fixing Oversights

Anything which was overlooked in development was added to a through hole board inserted behind our PCB

- Pull up resistors
- Pull down resistors
- Crystal oscillator



Debugging

Most of the system was verifiable via inspection but still things did occasionally go wrong and when they did we debugged as follows:

- Check if everything had power - Check if GPS sensor LEDs were on - Check if OLED displayed anything - Check if ALS was outputting properly	- Check if head & tail light were working - Check if reset button worked - Check if turn signals work

We approached it this way once the battery, CC, and programming had been completed.

Recommendations for Further Works

- Adding USB charging ports to the bike
- Added temperature sensors
- Reverse geocoding using faster microcontroller
- Simplifying the wiring of the system
- Higher resolution color display with touch interface

Questions?