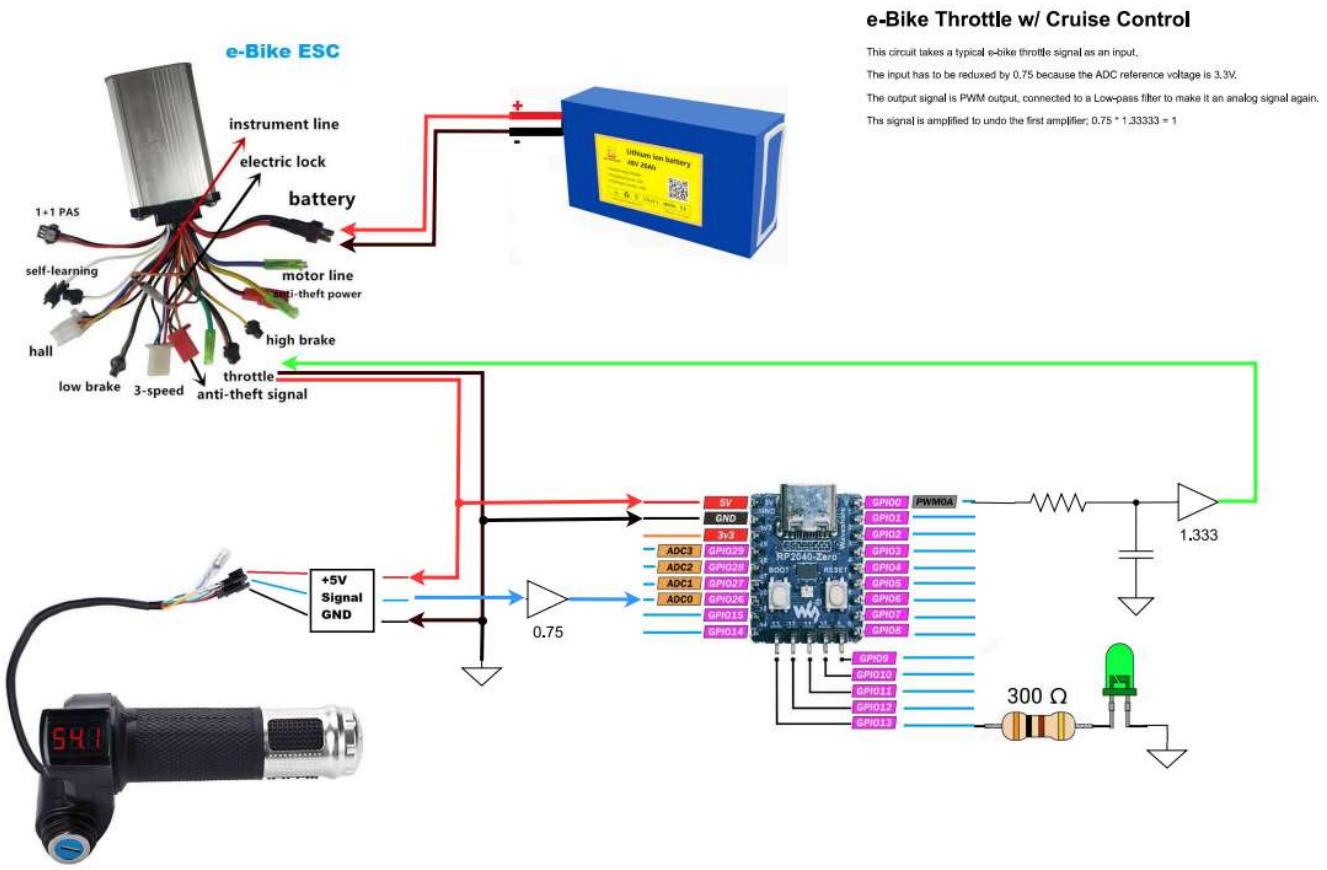
e-Bike Smart Throttle

By: Richard R. Vasquez

Date: 10/17/2024



Contents

Table of Figures

Table of Tables

Introduction

Objective:

This document provides a detailed overview of the design and development of an adapter module that converts a standard e-bike throttle into a “smart throttle” with features not found on standard off-the-shelf throttles.

Cruise control is the first such feature to be developed. It allows the rider to release the throttle once a hold speed has been engaged. This reduces rider stress during long rides and helps to extend the mechanical life of the throttle mechanism, which can be subject to damage with excessive wear and tear.

Scope:

The focus of this document is on the cruise control feature development for a standard e-bike throttle. However, it first reviews the Analog-to-PWM Throttle project, upon which this work is based (available in .docx and .pdf formats). The mathematical principles leading to the code values will be discussed, and all code development will be conducted using the Thonny IDE.
This document does not cover programming or running the Raspberry Pi RP2040; for those details, please refer to *How to Use RP2040 Zero* (available in .docx and .pdf formats).

This document presents the necessary mathematics to develop a general-purpose algorithm that compensates for variations in throttle potentiometer values due to manufacturing tolerances. The purpose of this document is to support the development of a viable commercial product. This device is not currently available for sale on the internet, unless you buy an e-bike or scooter with this feature. I designed and developed it for my own personally owned e-bikes made with off-the-shelf throttles.

Theory of Operation:

The operation is intended to be intuitive, acting as a common throttle. The smart features are implemented automatically. Cruise control is activated when the devices is in Hold mode to maintain a Hold_Speed value.

Hold mode is activated when the rider maintains a constant speed for more than 10 seconds. Hold mode is disengaged if the rider uses the brakes or slightly adjusts the throttle, returning the system to Run mode. In Run mode, the throttle position is directly converted to a PWM output.

When Hold mode is first activated, the rider may not realize it, so the system continuously monitors whether the hold speed is exceeded by more than 1 MPH. If this happens, the throttle switches back to Run mode, adjusting the output according to the rider's input. If the rider releases the throttle, the system ignores the input until the speed increases to over 3 MPH.

In Hold mode, the system maintains the set speed until a speed deviation of 1 to 3 MPH occurs, at which point it reverts to Run mode. The exact threshold for these transitions will be determined through further experimentation and analysis.

Glossary

ADC – Analog to Digital Converter

Brass – Hand-built circuit board. Functional, but does not comply with standard manufacturing practices or methodologies.

PWM – Pulse Width Modulation

e-Bile – Electric Bicycle

ESC – Electronic Speed Controller

Prototype Description

Description: The prototype hardware consists of the following list. I will start off by prototyping everything on breadboard. Once perfected a permanently soldered *brass* prototype using prototyping board will be implemented. This design is simple enough to be made by hand for small quantities. A PCB layout would make this a viable product for mass manufacture.

- Raspberry Pi RP2040 Zero (made by Waveshare)
- 3-Pin JST female connector housing with pre-crimped sockets (connects to old throttle)
- 3-Pin JST male connector housing with pre-crimped pins (to ESC in place of old throttle)
- Breadboard + hook-up wire (development)
- TL062 Op Amp (amplification/buffering)
- 2 x 10 KΩ resistors (signal conditioning)
- 2 x 33 KΩ resistors (signal conditioning)
- 1 x 1 KΩ resistor (LED current limiting)
- 1 x 1 MΩ resistor (RC time constant)
- 3 x 0.1 μF capacitors (power decoupling, RC time constant)
- 1 x LED (mode/calibration indicator)

Prototypes:

Breadboard:

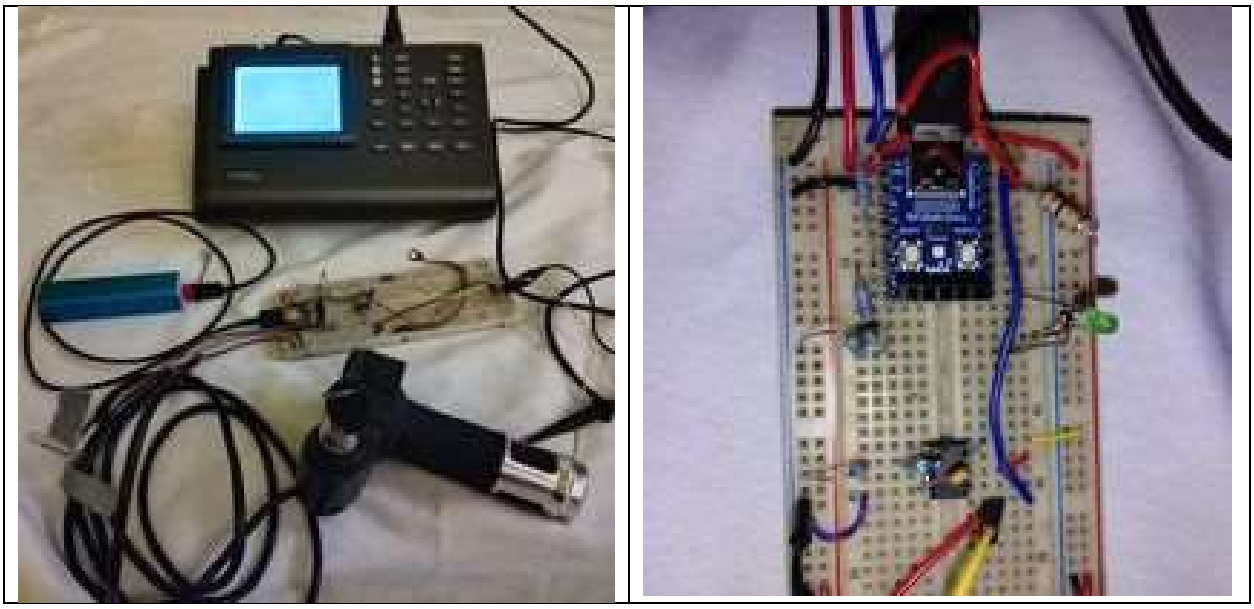


Figure 1- e-Bike Smart Throttle Breadboard Test Setup.

Brass Board:

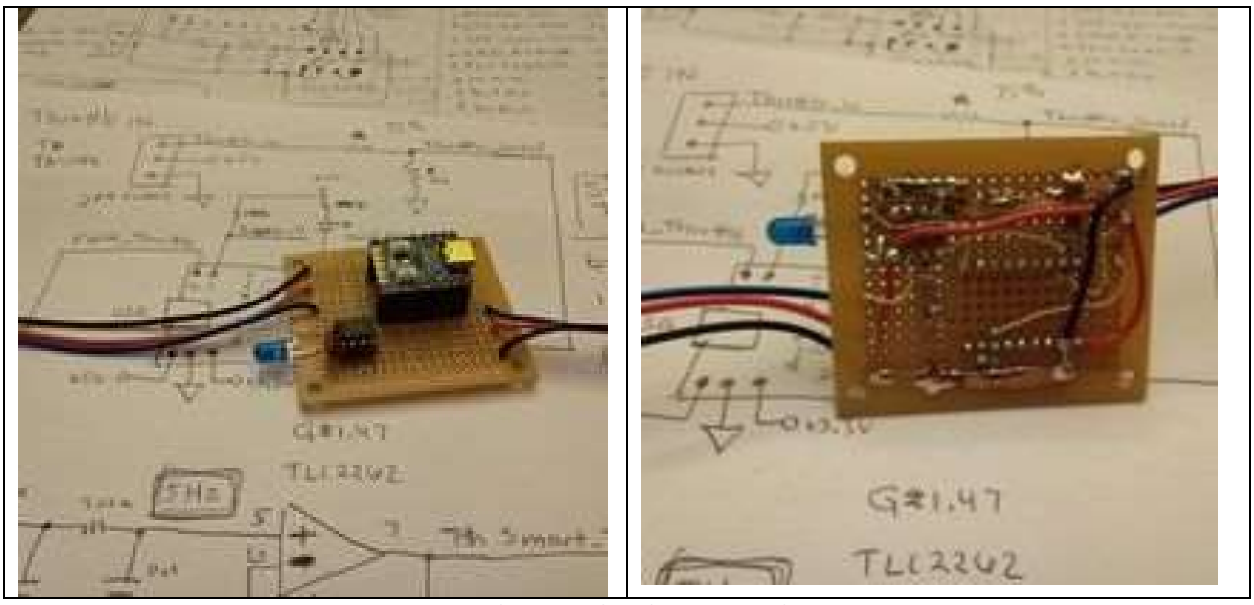


Figure 2- e-Bike Smart Throttle Brass Board Prototype.

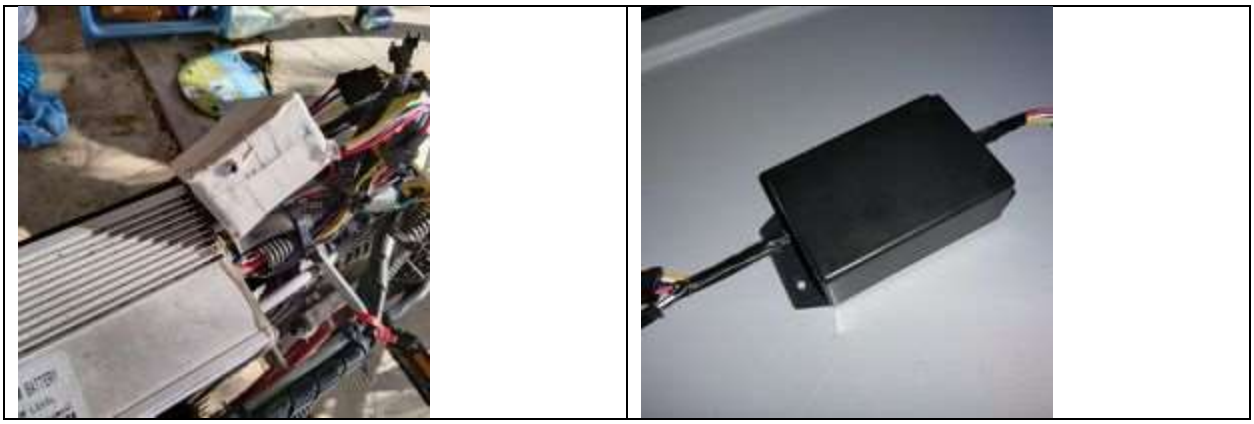


Figure 3- e-Bike Smart Throttle Enclosed

Test Equipment

Development: The following equipment is used to develop the project, but it won't be needed once the brass prototype has been developed.

- E-Bike throttle (conventional, analog)
- 0-5V power supply with current limiting
- Oscilloscope w/ test leads
- Digital Multi-meter (DMM) w/ test leads
- Computer with Thonny IDE
- USB type C cable

Prototype: The brass prototype shall be a road-test worthy and weather resistant.

- E-Bike (test vehicle)
- E-Bike throttle (conventional, analog)
- Weather-tight enclosure w/ grommets
- Prototyping board + hook-up wire

Throttle Response Analysis

Refer to the following diagrams when analyzing the throttle response. For a PWM-based ESC, the duty cycle typically ranges from 1000 to 2000 µS, corresponding to a speed range of 0% to 100%. In this design, I initially used percentages to control the duty cycle, though in practice, the timing (in microseconds) is more critical than the percentage itself. Historically, I referenced 5-10% of a 50 Hz PWM signal for this control, though this terminology is technically incorrect. The underlying math remains consistent, but it's important to focus on the time intervals rather than the percentage.

It's important to note that no two e-bike throttles can ever be expected to have the same range of output voltage; due to imperfections in the potentiometer. For this reason, it becomes important to develop a math model to make the program less rigid. By doing this we can employ an algorithm so that the throttle can self-calibrate using general function rather than hard numbers (constants).

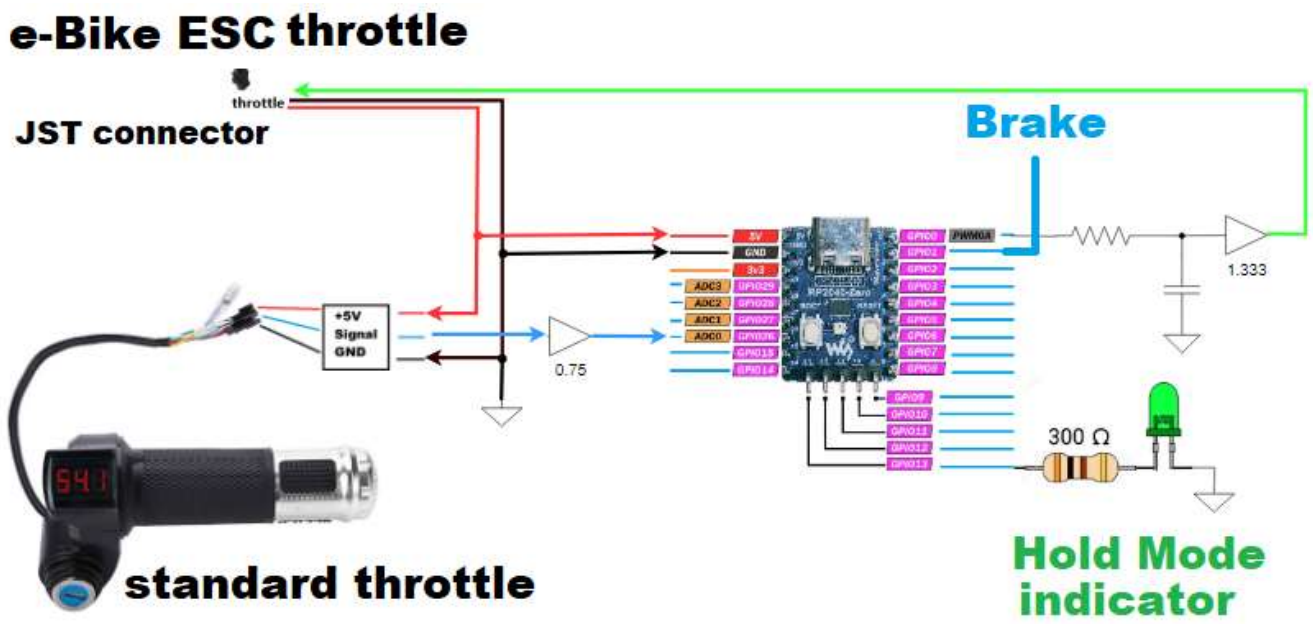


Figure 4- e-Bike Smart Throttle Schematic Diagram.

Throttle Voltage

Above we see the circuit used for this design. It is implemented using a resistor divider network to first reduce the input signal to approximately 0.75 of its original value. The purpose of this is because the RP2040 uses an internal +3.3V reference voltage for the ADC reading the throttle voltage. The typical throttle voltage exceeds 3.3V.

In this diagram a low-pass RC filter is used before a buffer. In actuality an active-filter design is used. It should be noted that this section is not required for a PWM-type ESC. The purpose of this filter and the amplifier with gain 1.333 is to undo the reduction done by the resistor divider network. The low-pass filter us used to convert the PWM signal back into an analog signal for a low-cost ESC found on most e-Bikes. This helps to make it a viable commercial product.

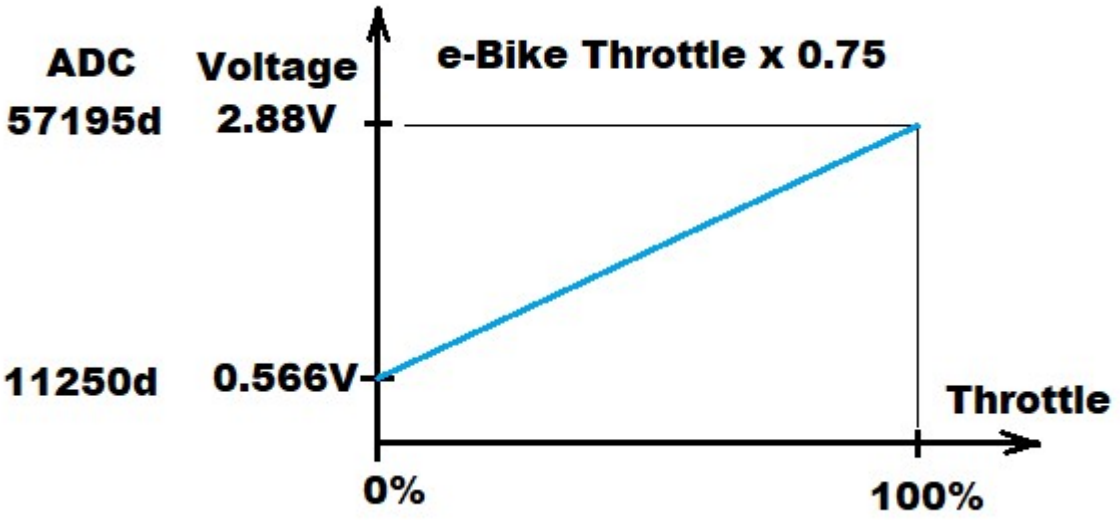


Figure 5- e-Bike Throttle Signal Response (scaled to 75%)

Above shows the analog response from a standard e-bike throttle, reduced to 75% scale with a resistor divider network. The value 0.75 was chosen because it was easy enough to employ with 4 resistors of the same value; 3 resistors in parallel give us 1/3 the value of one of the.

Likewise, it's easy to make a 1.33333 gain using four equal value resistors. The choice was made to change the divider to use 10K and 33K resistors for simplicity. Now the math has changed and that has forced a re-write of the original code. This is the main motivating purpose for creating this document.

In the diagram above a DMM was used to measure the throttle voltage, as measured by the ADC (using a program ADC.py, shown below). To the left of the voltage a decimal value is shown representing the ADC output. From the values shown, it appears the reference voltage is 3.299953 volts, which is very close to the theoretical value of 3.3 volts; 0.001% error in this case. For this reason, we can presume the ADC to be accurate and precise enough for our measurements. For this reason, we can develop using theoretical values for the ADC.

There may be some small margin each end of the response. Should the throttle exceed the expected range, but the software will clip the range so that it doesn't exceed the ESC's capability.

Analog to PWM Throttle Details

We are using a frequency of 50 Hz, which has a period of 0.02 seconds or $20{,}000\ \mu S$. For a $2{,}000\ \mu S$ pulse, the duty cycle needs to be: $\frac{2{,}000\ \mu S}{2{,}000\ \mu S} = 0.1\ or\ 10\%$. The duty cycle for $1{,}000\ \mu S$ would be 5%.

PWM Output:

The RP2040 Zero has a 16-bit PWM, therefore;

10% of 65535 = 6553.5 which is approximately 6554

5% of 65535 = 3277 (approximately)

ΔPWM = 6554-3277 = 3277 100% - 50% = 50% This is for the output.

ADC:

The ADC is 16-bits with a 3.3V reference voltage:

$$ADC_{step} = \frac{3.3}{2^{16} - 1}\left(\frac{\mu V}{bit}\right) = \frac{3.3}{65535} = 50.3548\left(\frac{\mu V}{bit}\right)$$

ADC: 50.355 uV/step

Scaled Down Throttle:

ADC_{VMax} = Full speed = 2.88 volts, ADC value is 57195d

ADC_{VMin} = Min speed = 0.566 volts, ADC value is 11240

$\Delta ADC = ADC_{VMax} - ADC_{VMax} = 57195 - 11240 = 45955$ This is the range of the throttle.

$ADC_{VMax} = 57195d$

$ADC_{VMin} = 11250d$

$\Delta ADC = 45955d$ This is the full range of the throttle.

Table 1 - Throttle Voltages

Throttle ID	Vmin (@ 0% speed)	Vmax (@ 100% speed)	Scale Used	ADC_Vmin (@ 0% speed)	ADC_Vmax (@ 100% speed)
Throttle No. 1	0.755	3.84	3/4	0.566	2.88
Throttle No. 2	0.830	4.128	3/4	0.612	3.068

Note: $ADC_{step} = 50.355\left(\frac{\mu V}{bit}\right), V_{Ref} = 3.3 \pm 0.03\ V$

Table 2 - Throttle ADC Values

Throttle ID	**Expected ADC value (@ 0% speed)**	**Expected ADC value (@ 100% speed)**	**Actual ADC value (@ 0% speed)**	**Actual ADC value (@ 100% speed)**	Ideal $\boldsymbol{\Delta ADC}$	**Actual $\boldsymbol{\Delta ADC}$**	**Ideal Response slope (m)** $\frac{\Delta\text{PWM}}{\Delta\text{ADC}}$	**Actual Response slope (m)** $\frac{\Delta\text{PWM}}{\Delta\text{ADC}}$
Throttle No. 1	11240	57194	11250	57195	45954	45945	0.0713	0.0713
Throttle No. 2	12154	60927	12225 averaged	61435 averaged	48773	49210	0.06719	0.06659

Note: $\Delta PWM = 3277$. This is the change in PWM output duty cycle. $5\%\ of\ (2^{16} - 1) = 3276.75$

Output Option 1: PWM

Full throttle range corresponds to PWM range from 5% to 10% (50Hz, 1mS to 2mS)

The throttle response is linear, so given the general slope equation: $y = m * x + b$, where y is the PWM output, x is the ADC value, m is the response slope and b is a PWM offset:

$$\text{PWM} = m * ADC_{VMin} + b$$

$\text{PWM}_{\text{Min}} = m * ADC_{VMin} + b$, where m is slope and b is offset

PWM_{Min} is the PWM we expect for the minimum voltage: 5% or 3277. The math is explained below.

In general, PWM is the output signal. When it is divided by $(2^{16} - 1)$, or 65535, this value is the actual duty cycle that controls the ESC. This value ranges from 5% to 10% when using a frequency of 50 Hz.

In practice, the PWM ESC can work up to 400 Hz, which has a period of $2500\ \mu S$. This means the duty cycle will range from:

$$\frac{1000}{20000} = 0.05 = 5\% \quad to \quad \frac{2000}{20000} = 0.1 = 10\%$$

$$0.05 * 65535 = 3277 \quad and \quad 0.1 * 65535 = 6554$$

$$\Delta\text{PWM} = 3277 \text{ @ frequency} = 50 \text{ Hz}$$

The corresponding PWM value for 40-80% are:

$$0.4 * 65535 = 26214 \quad and \quad 0.8 * 65535 = 52428$$

$$\Delta\text{PWM} = 26214 \text{ @ frequency} = 400 \text{ Hz}$$

In general, the slope and offset are found by:

$$m = \left(\frac{\Delta\text{PWM}}{\Delta ADC}\right), \qquad b = y - m * x$$

Throttle Samples

Two sample of throttles were available for testing. These calculations model their responses. The general formula for the throttle response is: $\mathrm{PWM} = m * ADC_{VMin} + b$.

Throttle ID #1

Slope:

$$\mathrm{m} = \frac{\Delta \mathrm{PWM}}{\Delta ADC} = \frac{3277}{45945} \approx 0.071324 \approx 1/14.020$$

Offset:

$$b = y - m * x = \mathrm{PWM}_{\mathrm{Min}} - \left(\frac{\Delta \mathrm{PWM}}{\Delta ADC}\right) ADC_{VMin}$$

$$b = 3227 - (0.071324) * 11250 = 3277 - 802.4 = 2474.6 \approx 2475$$

Throttle ID #1:	$\mathbf{m \approx 0.071324 \approx \frac{1}{14.02}}$,	$\boldsymbol{b \approx 2475}$

Throttle ID #2

Slope:

$$\mathrm{m} = \frac{\Delta \mathrm{PWM}}{\Delta ADC} = \frac{3277}{49210} \approx 0.06659 \approx 1/15.017$$

Offset:

$$b = y - m * x = \mathrm{PWM}_{\mathrm{Min}} - \left(\frac{\Delta \mathrm{PWM}}{\Delta ADC}\right) ADC_{VMin}$$

$$b = 3227 - (0.06659) * 12225 = 3277 - 814.063 = 2462.94 \approx 2463$$

Throttle ID #2:	$\mathbf{m \approx 0.06659 \approx \frac{1}{15.017}}$,	$\boldsymbol{b \approx 2463}$

Table 3 - Throttle Response Summary

Throttle ID	Actual ADC value (@ 0% speed)	Actual ADC value (@ 100% speed)	Actual ΔADC_{MAX}	Actual Response slope (m)	Actual Response offset (b)
Throttle No. 1	11240	57195	45955	0.0713	2475
Throttle No. 2	12225 Averaged	61435 averaged	49210	0.06659	2463

Output Option 2: Analog

This is for an analog output using a low-pass RC filter.

Slope & Offset

$\boldsymbol{m = 1,\ b = 0}$

The gain and offset for the conversion from ADC to PWM is a direct conversion. Offset = 0 and the slope is a perfect 1.0. This is an example showing this:

$$V_{ADC} = 2.88\ (V), \qquad ADC_{step} = 50.355\ \left(\frac{\mu V}{bit}\right)$$

$$ADC_{value} = \frac{2.88}{50.355}\left(\frac{V}{\frac{\mu V}{bit}}\right) = \frac{2.88}{50.355\mu}\left(\frac{V}{\frac{V}{bit}}\right) = \mathbf{57194}\ (bit)$$

Our PWM value is 57194 bits. If we convert this value to PWM, we get:

$$Voltage = DutyCycle * V_{ref}, \qquad where\ V_{ref} = 3.3\ V$$

$$DutyCycle = \frac{PWM}{65535} = \frac{57194}{65535} = 87.2724\%$$

$$Voltage = DutyCycle * V_{ref} = 0.872724 * 3.3 = \mathbf{2.87999\ V}, which\ is\ very\ close\ to\ 2.88\ V$$

The voltage is then fed into an amplifier with a gain inverse of the divider. This makes the output close to the original DC value from the throttle. We can see that: $\frac{4}{3} * \frac{3}{4} = 1$.

Low-Pass Filter

We do have to design our low-pass RC filter so that our signal is much higher in frequency than our cutoff frequency. The higher the PWM frequency, the lower we can make our RC time constant, and therefore use smaller capacitors. We do not have this restriction for this Analog output option since we do not have to restrict our pulse widths between $1{,}000\ to\ 2{,}000\ \mu S$. At $2{,}500\ \mu S$ the frequency is 400 Hz. This would be the maximum frequency we can use for Option 1.

$$f_{cuttoff} \leq \frac{1}{R * C} \ll 50\ Hz$$

Therefore, we will use $f_{cuttoff} = 5\ Hz$. Rearranging the formula, we find that,

$$C \leq \frac{1}{R *\ f_{cuttoff}} = \frac{1}{R *\ 5}$$

We'll try a large value for R; $R = 10^6$

$$C \leq \frac{1}{10^6 *\ 5} = 0.2 * 10^{-6} = 0.2\ \mu F$$

$$\boldsymbol{R = 1\ M\ \Omega}, \qquad \boldsymbol{C = 0.2\ \mu F}$$

Programming Constants

At this point we have derived the necessary equations to characterize any given e-bike throttle and employ their response characteristics as numerical values for use in the embedded microcontroller that makes this a "smart throttle". This is the purpose for writing this document.

Below is a table with code constants used to support Throttle ID #1. This section of the document shows how these values are calculated.

Table 4 – Code Constants (Throttle ID #1)

It was shown above that $ADC_{step} = 50.355\left(\frac{\mu V}{bit}\right)$. We can presume that $V_{Ref} = 3.3\,V$.	
STOPPED = 11240 # Throttle min. voltage = 0.566V (measured with DMM)	$V_{min} = 0.566V$ The ADC value should be: $\frac{0.566(V)}{50.355\left(\frac{\mu V}{bit}\right)} = 11240\ (bits)$
FULLSPEED = 57194 # Throttle min. voltage = 2.88V (measured with DMM)	$V_{max} = 2.88V$ The ADC value should be: $\frac{2.88(V)}{50.355\left(\frac{\mu V}{bit}\right)} = 57194\ (bits)$
DELTAADC_MAX = 45954 # ADC span	$\Delta ADC_{Max} = ADC_{Max} - ADC_{Min} = 57194 - 11240 = 45954$
DELTA1MPH = 1641 # ADC threshold for 1 MPH change	28 mph is the speed range of e-bike. $\frac{\Delta ADC}{28\ mph} = 1641.21 \approx 1641$
1_MPH = 12891 # ADC value for 1 MPH	$1MPH = STOPPED + \Delta 1MPH = 11240 + 1641 = 12881$
3_MPH = 16163 # ADC value for 3 MPH	$STOPPED + 3 * \Delta 1MPH = 11240 + 3 * 1641 = 16163$
MINPWM = 3277 # PWM set to 5%	$5\%\ x\ (2^{16} - 1) = 3276.75 \approx 3277$
MAXPWM = 6554 # PWM set to 10%	$10\%\ x\ (2^{16} - 1) = 6553.5 \approx 6554$
SLOPE = 0.071324 # Throttle response slope for PWM output.	$\mathrm{m} = \frac{\Delta \mathrm{PWM}}{\Delta ADC_{Max}} = \frac{3277}{45945} \approx 0.071324 \approx 1/14.020$
OFFSET = 2475	$b = \mathrm{PWM_{Min}} - \left(\frac{\Delta \mathrm{PWM}}{\Delta ADC}\right) ADC_{VMin}$ $= 3227 - (0.071324) * 11240$ $= 3277 - 801.7 = 2475.3 \approx 2475$

Table 5 – Code Constants (Throttle ID #2) Ideal Reference Presumed.

It was shown above that $ADC_{step} = 50.355\left(\frac{\mu V}{bit}\right)$. We can presume that $V_{Ref} = 3.3\,V$.	
These steps presume there is no error.	
STOPPED = 12153 # Throttle min. voltage = 0.612V (measured with DMM)	$V_{min} = 0.830V$ The ADC value should be: $\frac{0.612(V)}{50.355\left(\frac{\mu V}{bit}\right)} = 12153\ (bits)$
FULLSPEED = 57194 # Throttle min. voltage = 3.068V (measured with DMM)	$V_{max} = 3.068V$ The ADC value should be: $\frac{3.068(V)}{50.355\left(\frac{\mu V}{bit}\right)} = 60928\ (bits)$
DELTAADC_MAX = 48775 # ADC span	$\Delta ADC_{Max} = ADC_{Max} - ADC_{Min} = 60928 - 12153 = 48775$
DELTA1MPH = 1742 # ADC threshold for 1 MPH change	28 mph is the speed range of e-bike. $\frac{\Delta ADC_{Max}}{28\ mph} = \frac{48775}{28\ mph} = 1741.96 \approx 1742$
1_MPH = 13895 # ADC value for 1 MPH	$1MPH = STOPPED + \Delta 1MPH = 12153 + 1742 = 13895$
3_MPH = 16163 # ADC value for 3 MPH	$STOPPED + 3 * \Delta 1MPH = 12153 + 3 * 1742 = 17379$
MINPWM = 3277 # PWM set to 5%	$5\%\ x\ (2^{16} - 1) = 3276.75 \approx 3277$
MAXPWM = 6554 # PWM set to 10%	$10\%\ x\ (2^{16} - 1) = 6553.5 \approx 6554$
SLOPE = 0.067186 # Throttle response slope for PWM output.	$\mathrm{m} = \frac{\Delta \mathrm{PWM}}{\Delta ADC_{Max}} = \frac{3277}{48775} \approx 0.067186 \approx 1/14.884$
OFFSET = 2461	$b = \mathrm{PWM_{Min}} - m * ADC_{VMin}$ $= 3227 - 0.067186 * 12153$ $= 3277 - 816.511 = 2460.489 \approx 2461$

Table 6 – Code Constants (Throttle ID #2) Measured Reference Values.

It was shown above that $ADC_{step} = 50.355\left(\frac{\mu V}{bit}\right)$. We can presume that $V_{Ref} = 3.3\,V$.	
We will not make that presumption in this table. The ADC values were observed on a terminal.	
STOPPED = 12225 # Throttle min. ADC value averaged	$V_{min} = 0.612V$ (measured) The ADC voltage per bit is: $ADC_{step} = \frac{0.612(V)}{12225\ (bits)} = 50.061\left(\frac{\mu V}{bit}\right)$
$V_{Ref} = 3.28$ # Volts (calculated) measured 3.32V with DMM.	$V_{Ref} = (2^{16} - 1) * ADC_{step} = 65535\ (bit) * 50.061\left(\frac{\mu V}{bit}\right) = 3.28\ (V)$
	$V_{min} = 3.068V$ (measured) The ADC voltage per bit is: $ADC_{step} = \frac{3.068(V)}{61435\ (bits)} = 49.939\left(\frac{\mu V}{bit}\right)$
$V_{Ref} = 3.27$ # Volts (calculated) measured 3.32V with DMM.	$V_{Ref} = (2^{16} - 1) * ADC_{step} = 65535\ (bit) * 49.939\left(\frac{\mu V}{bit}\right) = 3.27\ (V)$
$V_{Ref} = 3.28$ # We will use this value.	
FULLSPEED = 61435 # Throttle max. ADC value averaged	$V_{max} = 3.068V$ The ADC value should be: $\frac{3.068(V)}{61435\ (bits)} = 49.939\left(\frac{\mu V}{bit}\right)$
DELTAADC_MAX = 49210 # ADC span	$\Delta ADC_{Max} = ADC_{Max} - ADC_{Min} = 61435 - 12225 = 49210$
DELTA1MPH = 1756 # ADC threshold for 1 MPH change	28 mph is the speed range of e-bike. $\frac{\Delta ADC_{Max}}{28\ mph} = \frac{49210}{28\ mph} = 1757.5 \approx 1756$
1_MPH = 13981 # ADC value for 1 MPH	$1MPH = STOPPED + \Delta 1MPH = 12225 + 1756 = 13981$
3_MPH = 17493 # ADC value for 3 MPH	$STOPPED + 3 * \Delta 1MPH = 12225 + 3 * 1756 = 17493$
MINPWM = 3277 # PWM set to 5%	$5\%\ x\ (2^{16} - 1) = 3276.75 \approx 3277$
MAXPWM = 6554 # PWM set to 10%	$10\%\ x\ (2^{16} - 1) = 6553.5 \approx 6554$
SLOPE = 0.06659 # Throttle response slope for PWM output.	$\mathrm{m} = \frac{\Delta \mathrm{PWM}}{\Delta ADC_{Max}} = \frac{3277}{49210} \approx 0.066592 \approx 1/15.017$
OFFSET = 2463 # Throttle response offset for PWM output.	$b = \mathrm{PWM_{Min}} - m * ADC_{VMin} = 3227 - 0.066592 * 12225 = 3277 - 814.077 = 2462.923 \approx 2463$

Table 7 –Throttle Code Constants Compared

Variable	Throttle #2 (measured)	Throttle #2 (ideal)	Throttle #1
STOPPED	12225	12153	11240
FULLSPEED	61435	57194	57194
DELTAADC_MAX	49210	48775	45954
DELTA1MPH	1756	1742	1641
1_MPH	13981	13895	12891
3_MPH	17493	16163	16163
SLOPE	0.06659	0.067186	0.071324
OFFSET	2463	2461	2475

Smart Throttle Features

The previous sections help us build a basic PWM Throttle. We can even convert the output back to analog to support low-cost ESCs. The section describes features that make this a smarter product.

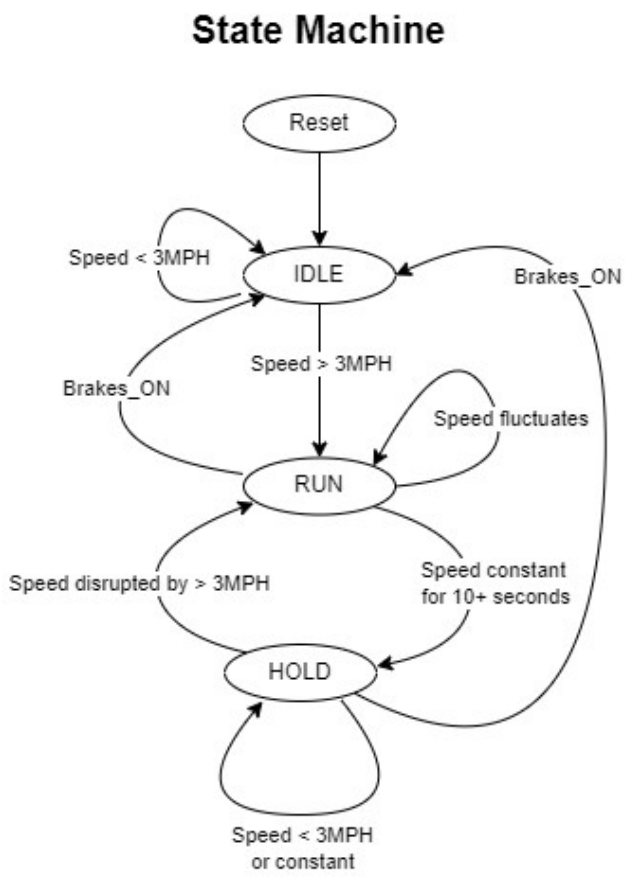
Cruise Control

To develop cruise control, we use a top-down approach using a state-chart diagram. This state-chart describes the operation of the throttle at a very high level. These images were draw using a web application at the URL: https://app.diagrams.net/#

From the state-chart we develop a flowchart that describes the behavior in simple English or pseudocode.

The flowchart pseudocode is easy to convert to any coding language.

State-Chart Diagram



This State Machine describes the operation of the cruise control feature.

Upon reset the device remains in Idle mode until it senses a speed greater than 3 MPH on the throttle.

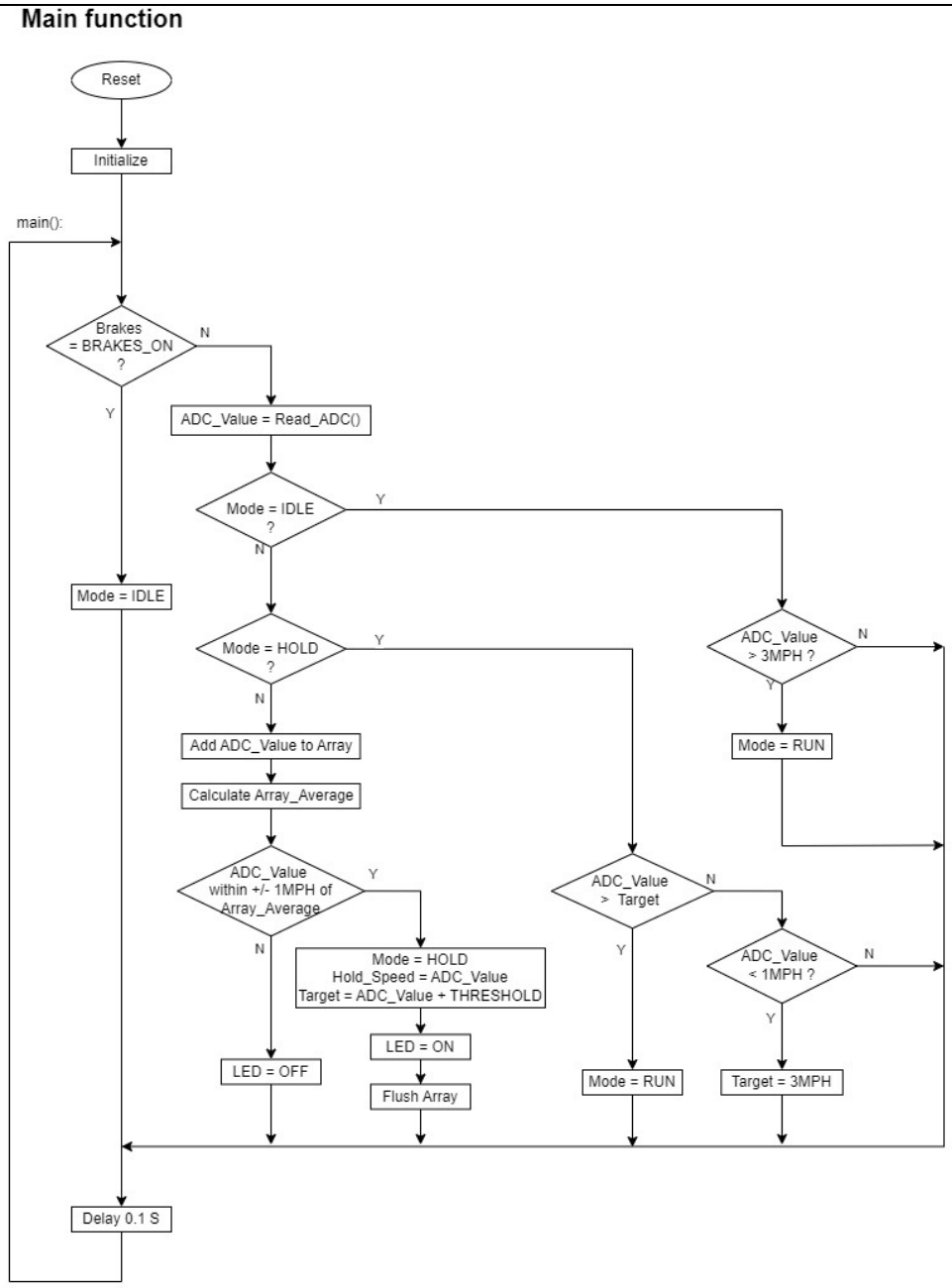
In Run mode the throttle is continuously converted and updated as a PWM output.

Hold Mode is automatically activated when the rider maintains a stead speed for a period of 10 seconds. The rider may let go of the throttle and the speed will be held constant.

Hold mode can be disrupted by applying the brakes, or by increasing the speed by more than 3 MPH.

Figure 6- e-Bike Smart Throttle State Machine

Flowchart/Pseudocode (main)



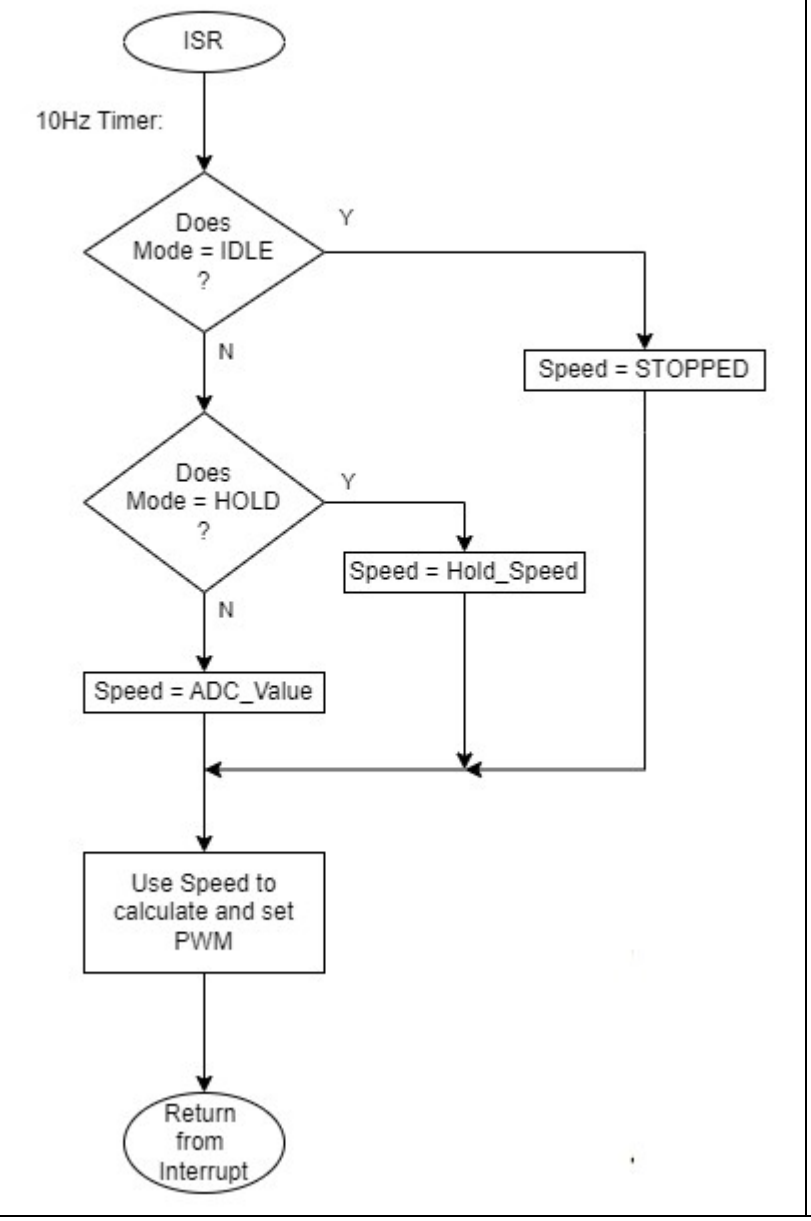
This is the main loop for the program. This is the function that reads the ADC and operates on it according to the mode it is in. This flowchart complies with the logic of the State Machine Chart. A short delay is used in the loop to help provide stability.

The Initialize block sets up the device and initiates a 10 Hz timer interrupt.
100 sample in an array are used to gather 10 seconds worth of data.

The bulk of the operations are done in an interrupt service routine.

Figure 7- e-Bike Smart Throttle Main() Algorithm

Flowchart/Pseudocode (Interrupt Service Routine)



This interrupt service routine is called every 0.1 seconds.

It continuously monitors for the Mode state (Idle, Run or Hold) and sets the speed accordingly.

It then converts the speed value into a PWM output.

Figure 8- e-Bike Smart Throttle ISR() Algorithm

Utility Functions

At this time of this writing application ideas for this product came to mind. Here is a growing list of functions that can be supported by the current design, followed by a description of how to implement this in the design.

- Speedometer
- Joule Meter
- Tripometer/Odometer
- Keycode lockout

Speedometer

We can use one of the hall effect sensors to measure the wheel rotation. For the particular test motor used in these calculation, there are 7 pulses per revolution. Here is the math that describes how we can measure the speed in RPM (revolutions per minute).

The test e-Bike has a diameter of $\emptyset = 26\ (inches)$. The length of travel for 1 revolution is given by:

$$L = 26\,\pi\ (\cancel{inches}) * \frac{1}{12}\left(\frac{foot}{\cancel{inches}}\right) = 6.807\ (feet)$$

For this test vehicle motor, the number of pulses per revolution for each hall-effect sensor are given by:

$$PULSPERREV\ =\ 7\ \left(\frac{pulses}{revolution}\right)$$

It is known that the vehicle can reach a speed of 28 mph. It can be shown that:

$$28\ \left(\frac{miles}{hour}\right) = 41.066\ \left(\frac{feet}{second}\right), \qquad 1.4667\ \left(\frac{feet}{second}\right) = 1\ \left(\frac{mile}{hour}\right)$$

Let us relate the speed to the wheel geometry. For a given speed in MPH, let us find the revolutions per second.

$$\frac{1\ (MPH)}{L} = \frac{1.4667\ \left(\frac{\cancel{feet}}{second}\right)}{6.807\ (\cancel{feet})} = 0.2155\ \left(\frac{revolutions}{second}\right)$$

We can express the revolution speed in terms of hall-effect clock pulses using following at $1\ (mph)$:

$$0.2155\ \left(\frac{rev.}{sec.}\right) * PULSPERREV\ =\ 0.2155\ \left(\frac{\cancel{rev.}}{sec.}\right) * 7\ \left(\frac{pulses}{\cancel{rev.}}\right) = 1.508\left(\frac{pulses}{sec.}\right)$$

$$At\ 1\ \left(\frac{mile}{hour}\right) \rightarrow\ \ 1.508\left(\frac{pulses}{sec.}\right) or\ 0.663\ \left(\frac{sec.}{pulse}\right)$$

$$At\ 28\ \left(\frac{miles}{hour}\right) \rightarrow\ \ 42.24\left(\frac{pulses}{sec.}\right) or\ 23.73\ \left(\frac{\mu\ sec.}{pulse}\right)$$

Here we have arrived at the values that will allow us to design a speedometer function. We see that at 1 mph we have a relatively long period of time; 663 mS. At 28 mph we only have 24 μS between pulses.

Per Nyquist's theorem we should sample at least 2x as fast, so we actually have a period of only 12 μS between pulses. The longer period is 56x as long as the shortest. We see that the number of bits we would need for a counter should be at least:

$$log_2(56) = \frac{\ln(56)}{\ln(2)} = \frac{4.025352}{0.6931472} = 5.81, so\ we\ use\ 6\ bits$$

One way to implement a speedometer is to count the number of pulses that occur in 1 second. But at 1 mph, this would be only 1.5 pules per second. We would like to be able to track speed more accurately at lower speed and this makes it difficult.

There is another approach that can be taken; we can measure the time between pulses. However, we are again confronted with a challenge as the time between pulses can be as low as 24 μS. This would require a timer with a frequency of 83.33 KHz to comply with Nyquist.

Joule Meter

It is not difficult to measure the e-bike's battery current using a hall-effect sensor, such as the WCS1700, or a LEM module. The WCS1700 has a voltage range of 5V and a sensitivity of 33 mV/amp sensed. It is not difficult to implement a 4-20 mA transducer based upon this element. This would reduce the wire count to 2 and allow for long length.

Joules are a measure of energy, which is a product of power and time. A rider may consume more or less power depending how they ride, affecting battery life and range. The amount of power drawn can be measured in real time. The sample period is the measure of time at that given power level; the resulting joules can be accumulated in a variable that represents the total amount of joules consumed by the ride.

The usefulness of this information is that, when combined with the speedometer, gives a true measure of battery performance in real time. This data can be logged for later analysis and/or used to provide the rider with an accurate estimate of mileage available in the battery, given that we start from a know fully charged status. We can accumulate these joules to subtract from a value that approximates a fully charged battery capacity. We can later determine the what these values are by examining the log data and/or by experimentation.

Tripometer/Odometer

This feature can be easily implemented to help the rider estimate their riding performance: Average speed, maximum speed, length of trip, overall mileage. A Flash memory device with sufficient memory can be used to implement this function without wearing out the memory of the RP2040.

Keycode lockout

A weather-resistant keypad and LED indicator can be used to allow the rider to enter a code. The LED notifying the rider of the key presses, and flashes to notify rider of ready to use. A slow blink to indicate ready to use, or a fast set of pulses notifying the rider of the incorrect entry. The bike can be disable without the correct code. However, this feature can be bypassed by removing this product and re-

connecting the original throttle connectors. This feature would be most useful should the product be fully integrated into the throttle body, thereby preventing this tampering. The benefit of this feature is that the vehicle is secure without needing a key.

Calibration Functions

At this time of this writing, it has been determined that a calibration function can be avoided by using a potentiometer in the hardware. This affects the final gain, but there is a small offset error not accounted for. In testing offset was not significant enough to cause any inadvertent speed commands or loss of speed range.

However, should the need arise, this section describes a method to achieve calibration without any additional hardware.

The consequences of not calibrating this device may result in a portion of the usable range of the throttle may not be in the active region of the throttle. Or the active range may be too far offset to be usable.

Auto Calibrate

One method to calibrate is to use a GPIO button to sense the throttle upon reset. Fully activate the throttle upon startup, it will use the throttle position to sense this maximum value. The indicator light will indicate Calibration function is active. User holds the throttle until the indicator starts blinking. When the user lets go of the throttle it will sense the minimum position to complete the calibration function.

The calibration values are saved in Flash memory. If they are zero, this means the device has never been calibrated and the default values should be used, or they were corrupted. They are zeroed out should something go wrong with the calibration function. In this case, the default values are used.

Factory Reset

To do a factory reset, fully activate the throttle and brake upon power up (or reset). The defaults will be used to over write the calibration values. This is how the factory reset is implemented on an electric scooter, like the *Phantomgogo brand Model A8* 300W electric scooter.

Testing

Testing will consist of laboratory experiments to verify that the desired signal ranges (PWM or analog) have been achieved. First, we study the Code Functionality, then we examine the PWM conversion to analog to verify that the original analog signal from the basic throttle has been reproduced with the conversion.

Code Functionality

The following test conditions were all tested by observing the Hold LED and using an LED as an indication of PWM duty cycle, while operating the throttle or brake, as indicated by the appropriate column heading.

An oscilloscope was also used to verify that the pulse-width was within the desired range. For debugging, printf() code statements were used to send status information; Mode, ADC voltage, average array voltage as needed. The final code product has them commented out.

Table 8 – Operational Test Matrix

Test Condition	**Mode Under Test**	**Brake**	**Throttle**	**Hold LED**	**Comments**
Upon start-up	Idle	OFF	0 MPH	Flash ON-OFF	Flash notifies rider ready to use.
Upon start-up	Idle	OFF	< 3 MPH	OFF	Remains IDLE Throttle > 3 mph.
Not in Idle	Run	OFF	>= 1 MPH	OFF	Normal running operation.
Brake activated	Idle	ON	ANY	OFF	Overrides all over conditions.
Throttle stable 10+ seconds	Hold	OFF	>1 MPH	ON	Auto-Cruise Control activates into Hold mode.
While cruise-control active	Hold	OFF	< SET SPEED	ON	Auto-Cruise Control de-activates to Run mode.
While cruise-control active	Hold	OFF	1MPH to SET SPEED	ON	Auto-Cruise Control remains active between SET SPEED > Throttle > 1MPH
While cruise-control active	Hold	OFF	1MPH to SET SPEED	ON	Once Throttle < 1MPH, Auto-Cruise Control remains active. New threshold is 3MPH.
Rider had previously let go of the throttle.	Hold	OFF	> 3MPH	OFF	Auto-Cruise Control de-activates to Run mode after the threshold was exceeded.
While cruise-control active	Hold	OFF	> SET SPEED + 1MPH	OFF	Auto-Cruise Control de-activates to Run mode.
While cruise-control active	Hold	ON	ANY	OFF	Auto-Cruise Control de-activates to Idle mode

PWM to Analog Conversion

In analog mode the PWM signal output is converted to analog using a low-pass filter with a cutoff frequency of <5 Hz. A suitable RC time constant was created using 330 kΩ and 0.1 μF.

Ideally the ADC value used for the duty cycle should reproduce the analog signal with minimal error. However, initial testing has shown this to not be true. We will tabulate the data to study the reasons why.

The ADC.py test function will be used to observe the values as well as using a DMM to measure the voltage. The Oscilloscope will be used to measure the Duty Cycle. Printf() statements will be used to send the PWM values to compare against the actual duty cycle.

Table 9 – ADC Test Values

ADC (voltage)	ADC value (count)	Duty Cycle (%)	PWM (count)	Throttle Setting
V	(17146)	%		10%
V	(24528)	%		25%
V	(36830)	%		50%
V	(49133)	%		75%
V	(56514)	%		90%

(value) = ideal value from calculations. The real values may be approximated.

Table 10 – RP2040 Voltages

RP2040 Zero Signal	Measured Voltage
PWM max V	
PWM min V	
3.3 V	
5V	

Signal Scale-Down/Scale-Up

The analog voltage from the throttle exceeds 3.3V when at full speed. This exceeds the ADC reference voltage (3.3 volts), so a voltage divider is used to reduce the signal to 75% of its original value.

Once the signal is processed and reconstructed as an analog signal, it needs to be amplified to match the original analog signal level. The gain of this amplifier, ideally, should be the inverse of the ration used in the resistor divider network.

To demonstrate what his means, we used a resistor divider network with a gain of $\frac{3}{4}$. To undo this reduction, we simply need to amplify by a factor of $\frac{4}{3}$; $\frac{3}{4} * \frac{4}{3} = 1$.

Prototype Schematic

Below shows a copy of the hand-drawn schematic for this design. A preliminary BOM is shown in the image. Below the schematic is a wiring diagram used to construct the rugged prototype. This design is the same one used for the breadboard design.

This schematic is needed to generate a proper BOM to determine the board cost. The product will need an enclosure and grommets.

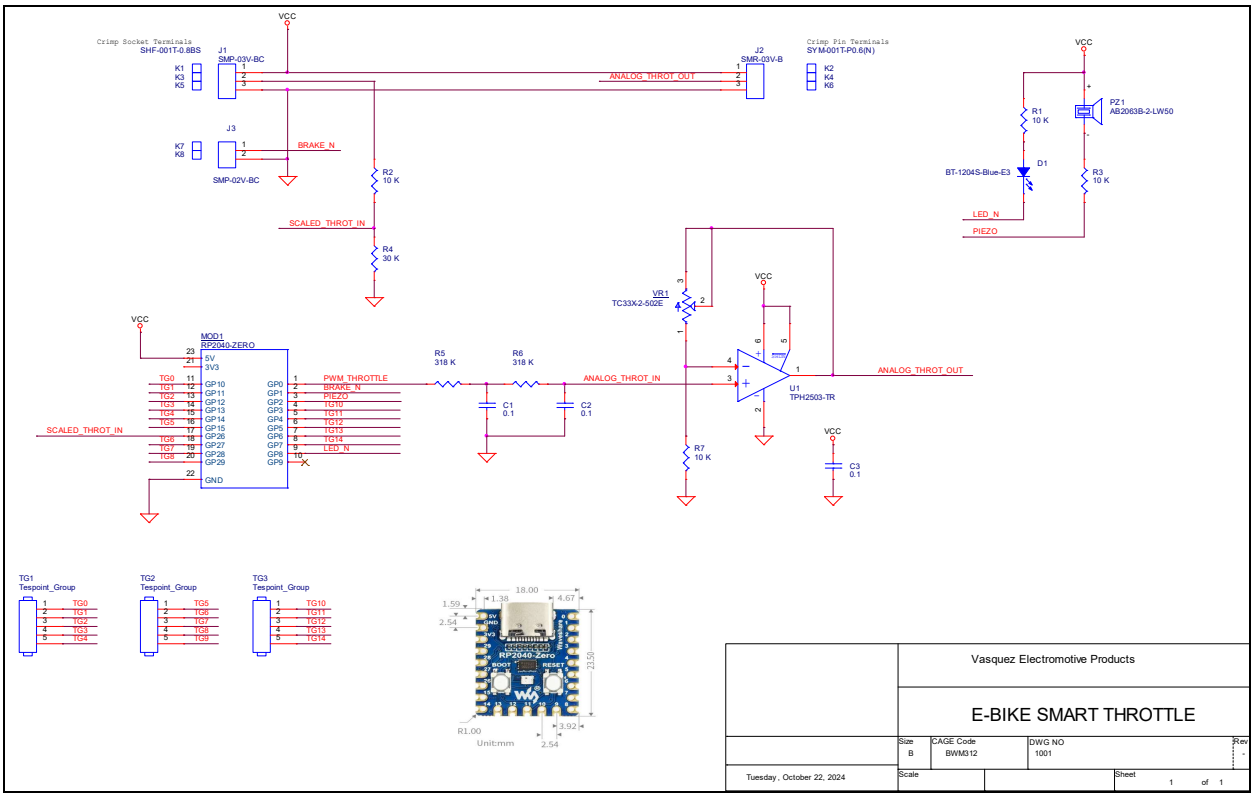
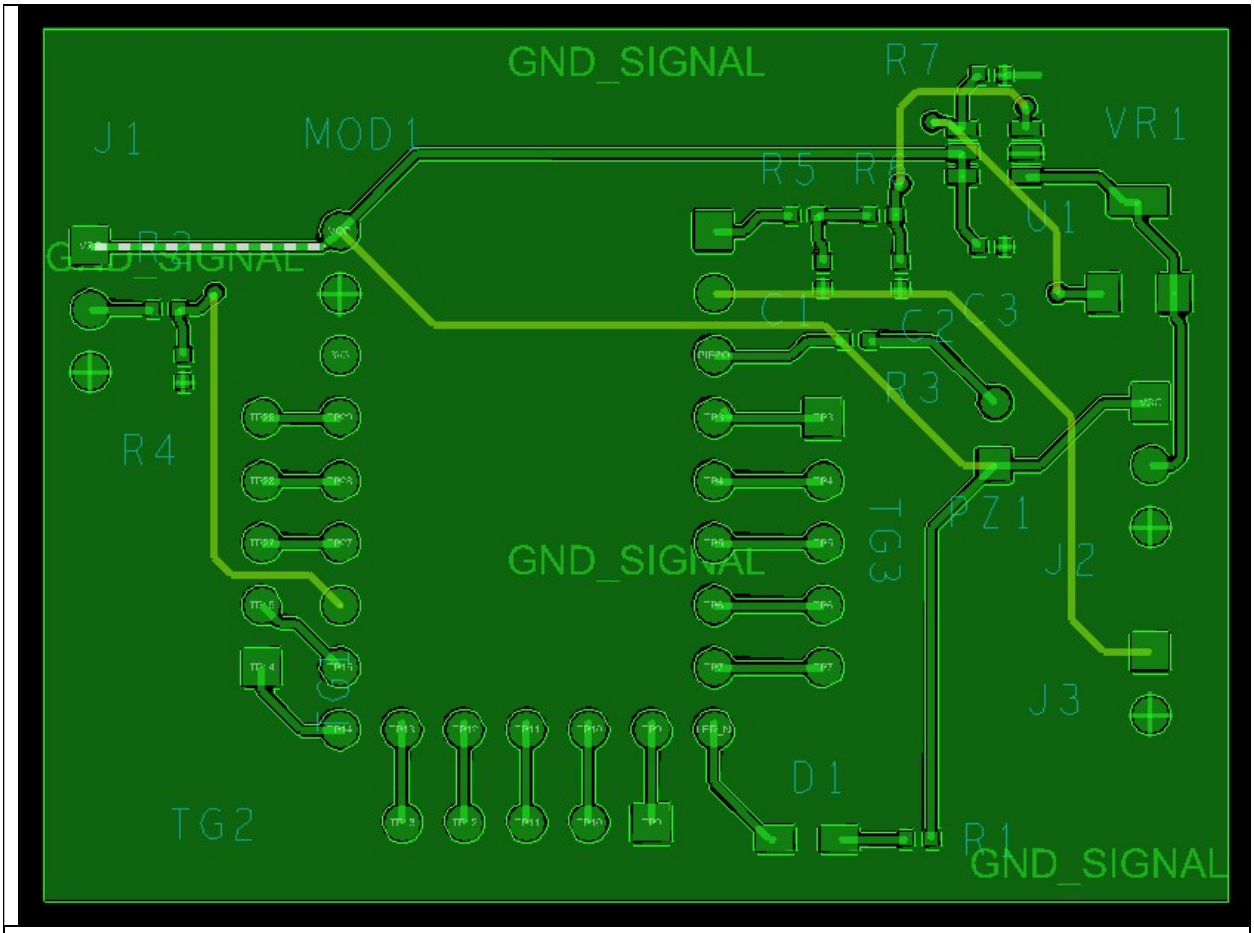


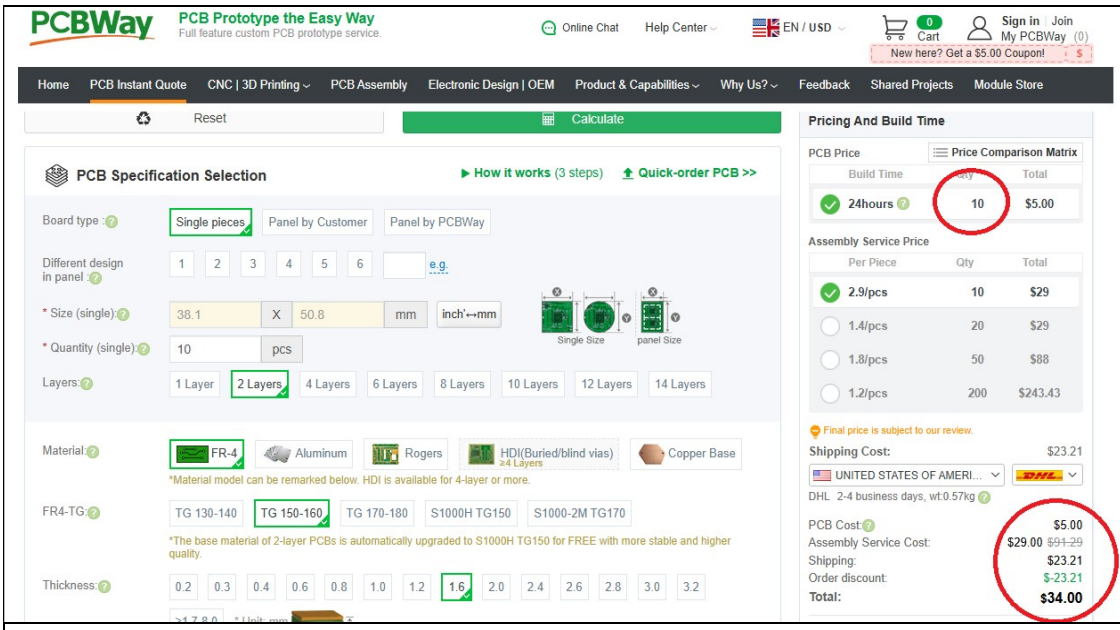
Figure 9- e-Bike Smart Throttle Schematic

PCB Layout



This is a first draft of a PCB layout that would work very well.

Figure 10- PCB Layout



I believe this price includes the bare boards.

Figure 11- PCB Costs Estimate

Table 11 – Costs Summary

Parts Total:	$13.57
PCB + Assembly	$3.40
Total:	**$16.97**

(This is a per board cost)

It seems we have challenges to bring the cost under $10. This product would have to sell for $51 to make any money. Note that this does not include the packaging or labor cost for programming.

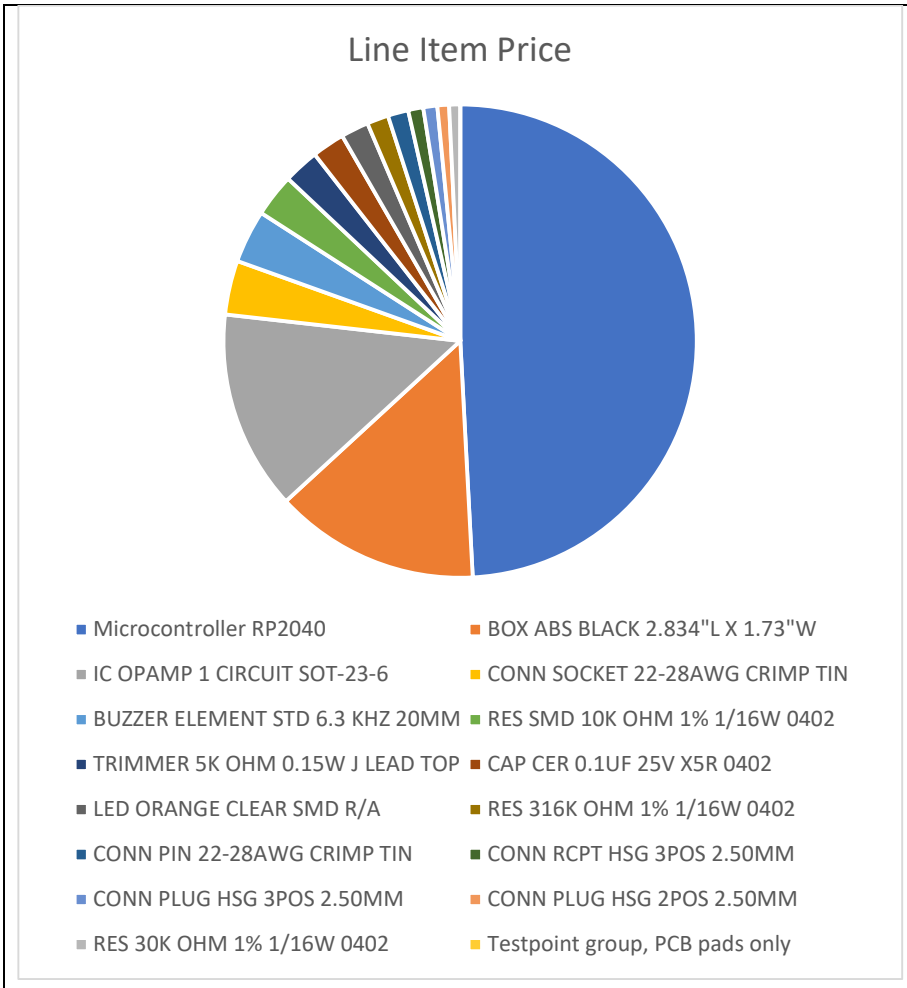
Production

We have a working prototype, but to make this a production-worthy product a proper schematic and PCB layout of the design are needed. This section shows the actual schematic details.

Table 12 – Bill of Material (BOM)

Item	Qty	Reference	Value	Manufacturer_PN	Mfr	Description
1	3	C1,C2,C3	0.1	CC0402KRX5R8BB104	YAGEO	CAP CER 0.1UF 25V X5R 0402
2	1	D1	APA2106SECK	APA2106SECK	Kingbright	LED ORANGE CLEAR SMD R/A
3	1	ENC1	CU-1952	CU-1952	Bud Industries	BOX ABS BLACK 2.834"L X 1.73"W
4	1	J1	SMP-03V-BC	SMP-03V-BC	JST Sales America Inc.	CONN PLUG HSG 3POS 2.50MM
5	1	J2	SMR-03V-B	SMR-03V-B	JST Sales America Inc.	CONN RCPT HSG 3POS 2.50MM
6	1	J3	SMP-02V-BC	SMP-02V-BC	JST Sales America Inc.	CONN PLUG HSG 2POS 2.50MM
7	5	K1,K3,K5,K7,K8	SHF-001T-0.8BS	SHF-001T-0.8BS	JST Sales America Inc.	CONN SOCKET 22-28AWG CRIMP TIN
8	3	K2,K4,K6	SYM-001T-P0.6(N)	SYM-001T-P0.6(N)	JST Sales America Inc.	CONN PIN 22-28AWG CRIMP TIN
9	1	MOD1	RP2040-ZERO	RP2040-ZERO	Waveshare Electronics	Microcontroller RP2040
10	1	PZ1	AB2063B-2-LW50	AB2063B-2-LW50	PUI Audio, Inc.	BUZZER ELEMENT STD 6.3 KHZ 20MM
11	4	R1,R2,R3,R7	10 K	RC1005F103CS	Samsung Electro-Mechanics	RES SMD 10K OHM 1% 1/16W 0402
12	1	R4	30 K	RC0402FR-0730KL	YAGEO	RES 30K OHM 1% 1/16W 0402
13	2	R5,R6	318 K	RC0402FR-07316KL	YAGEO	RES 316K OHM 1% 1/16W 0402

14	3	TG1,TG2,TG3	Tespoint_Group	N/A	N/A	Testpoint group, PCB pads only
15	1	U1	TPH2503-TR	TPH2503-TR	3Peak	IC OPAMP 1 CIRCUIT SOT-23-6
16	1	VR1	TC33X-2-502E	TC33X-2-502E	Bourns Inc.	TRIMMER 5K OHM 0.15W J LEAD TOP



The microcontroller cost is the single biggest cost. It is also one the easiest to improve depending upon the required performance.

The enclosure is the 2nd most expensive item. This cost cannot be significantly reduced without making a bulk purchase at a good price.

The op-amp is rather expensive at $1.85. This can be reduced to under $0.50.

JST Crimp pins are more expensive than the connector housings. A lower-cost vendor than Digikey can help reduce these costs. The JST SM series is the type of connectors that all e-Bikes use.

Figure 12- Smart Throttle Parts Cost Pie Chart

Stand & Road Testing

The brass unit was installed on the test vehicle and static tests were done to verify correct performance. Initial testing proved problematic as the voltage range exceed the range of the ESC. The motor would cut out when at full-throttle.

A calibration potentiometer was added to the final output stage. This worked well. The throttle was able to achieve the full range of speed. The brake feature was not connected.

Road Test #1

Actual road testing went well. The first test was to "crawl" at a slow speed. The cruise control feature worked perfectly. All ranges of speed were tested without problems. The same set of test conditions were done on the road. Full speed was not practical to achieve in a residential zone as stops were preventing long stretches of riding.

Important Braking Lesson

A potentially serious almost arose. The need to brake became suddenly required, but the first instinct was not to activate the throttle to disengage it. This was only a momentary mishap that never became a real problem, but the test driver has inside knowledge of this device, thereby preventing a problem. Knowledge overtook instinct to activate the throttle, disengaging the cruise control function.

Had the test driver not had this special knowledge this could potentially have led to an accident as the brakes had no effect on disengaging the cruise control. This "non-accident" served as reminder of the brake feature already implanted in both hardware and software. It was missing the brake connector.

The brake connector was added, completing this design 100%. The feature worked perfectly. The instinct to activate the brakes overrides the throttle in every case, as design.

Marketing

This product originated from a personal challenge faced by the author during long e-bike rides—carpal tunnel discomfort caused by prolonged use of thumb or twist throttles. Holding the throttle for extended periods not only leads to an uncomfortable ride but also raises the risk of throttle failure, potentially leaving riders stranded. This design was inspired by features found in e-scooters, which are not typically available in off-the-shelf e-bike throttles unless purchased as part of a complete e-bike.

Driven by necessity rather than market trends, the author sought to create a solution when no standalone product existed. After extensive research, it became evident that this feature could only be found in full e-bike systems, highlighting a significant gap in the market. A formal marketing study would be essential to demonstrate the viability and cost-effectiveness of this product to potential investors.

The Raspberry Pi RP2040 Zero module was selected for its low cost—$6.67 per unit at the time of this writing—and powerful capabilities, running MicroPython for rapid development. ChatGPT aided in quickly crafting the initial code, streamlining the development process. While the RP2040 is the most expensive component, priced at $6.67, the total component cost is estimated at around $14, with the enclosure priced at $3 and the remaining parts at approximately $4.

To further reduce production costs, future iterations could utilize a more economical microcontroller, such as a PIC from Microchip, potentially lowering the microcontroller cost to under $2. This could bring the total cost below $10, creating a highly profitable product. A $30 retail price would be both reasonable and attractive to consumers—this author would have gladly paid it to avoid the challenges of developing the product. Currently the price per unit is just under $18, but no cost reduction efforts were taken.

Conclusions

Summary

The goal of this project was to develop a smart throttle capable of providing automatic cruise control functionality. The key features of the design include:

- Automatically engagement
- Intuitive operation
- Improved ergonomics to reduce carpal tunnel strain
- Adds value to existing products
- Low-cost
- Upgradable and expandable for future enhancements

A breadboard prototype was first implemented using 2%-tolerance thru-hole resistors, proving the concept worked effectively. To ruggedize the design, a hand-wired prototype was assembled with surface-mount components of 1%-tolerance and housed in an enclosure for road testing.

For final testing, the throttle was installed on the test vehicle (the "Phoenix Mark III" e-bike). Initially, the throttle did not perform correctly, as the signal exceeded the expected range, causing the ESC to shut down. Further lab testing revealed that the output voltage was too high. I replaced the final gain resistor (4.99 kΩ) with a 5 kΩ potentiometer, and recalculated the ideal gain to be 1.33. After tuning the gain, the circuit functioned perfectly.

While a potentiometer was used for fine-tuning during testing, future designs can use fixed resistors, as the losses observed with 2%-tolerance components were minimal with 1%-tolerance parts. The project successfully met its objectives and is ready for further development.

The design can be built for about $17 per unit using turnkey services. PCB Assembly is one of the largest costs, but it is offset by the inclusion of the bare board. The assembly costs are about $3.4 of that $17. The bare board itself would only cost $2.5 without turnkey services, so we don’t save much by doing assembling them in-house.

In order of costs, from highest to lowest, here are the top 5 most expensive parts, and a target price to reduce the overall costs:

Table 13 – Part Cost Reduction Targets

Description	Line Item Price	Target Item Price
Microcontroller RP2040	$6.67	$1.67
BOX ABS BLACK 2.834"L X 1.73"W	$1.90	$1.90
IC OPAMP 1 CIRCUIT SOT-23-6	$1.85	$0.50
CONN SOCKET 22-28AWG CRIMP TIN	$0.50	$0.20
BUZZER ELEMENT STD 6.3 KHZ 20MM	$0.49	$0.25
Total:	$11.41	$4.52

We can reduce the costs by this much: $\$11.41 - \$4.52 = \$6.89$

This brings down the total costs to: $\$16.97 \ - \ \$6.89 \ = \ \$10.08$

This shows that we can potentially approach the target **unit price of $10.08**.

Appendix A

Python Code

These are some useful code snippets to help us develop out ***Smart Throttle*** features. A first draft of the Smart Throttle code is shown below. This code is targeted to the Raspberry Pi RP2040 Zero.

ADC Function

ADC program used to read ADC from throttle and print to the terminal.

Table 14 – ADC Function

```
# ADC.py
# By: Richard R. Vasquez
# Date: 10/15/24

import machine
import utime

sensor_temp = machine.ADC(26)
while True:
    reading = sensor_temp.read_u16()
    print("ADC: ", reading)
    utime.sleep(0.2)
```

Analog to PWM Throttle

This is the original version of the program that converts analog throttle to PWM.

Table 15 – Basic Analog to PWM Throttle Program

```
# PWM_Throttle.py
# By: Richard R. Vasquez
# Date: 10/15/2024

# Map the analog value to the PWM duty cycle (0-65535)
# Assuming the analog range is 0-3.3V (for 3.3V ADC reference)

# 10% of 65535 = 6553.5 approximately 6554
# 5% of 3277 (approximately)
# Observation of throttle: Full speed = 3.84 volts, Min speed = 0.76 volts
# Used 0.75 scale down, since 3.8 > 3.3 range of ADC reference voltage.
# Scaled down 0.75x Throttle: Full speed = 2.88 volts, Min speed = 0.566 volts
# ADC: 50.354 uV/step
# Delta steps: 57195-11250 = 45945   This is the range of the throttle.
# Delta PWM = 6554-3277 = 3277   100% - 50% = 50%   This is for the output.
#
# OPTION 1: This is for a PWM output.
# Full throttle range corresponds to PWM range from 5% to 10% (50Hz, 1mS to 2mS)
# Slope: m = 3227/45945 = 0.070237 = 1/14.2377
```

```
# Offset: b = y - m*x = 3227 - (11250/14.2377) = 3277 - 790.156 = 2436.84
#
# OPTION 2: This is for an analog output using a low-pass RC filter.
# Full throttle range corresponds to ADC value: 57195 - 11250 = 45945
# MinPWM = 11121        65535 * 0.566/3.3 = 65535 * 0.169697 = 11240
# MaxPWM = 57194        65535 * 2.88/3.3  = 65535 * 0.872727 = 57194
# Delta PWM = 57194 - 11240 = 45954     87% - 17% = 70%   This is for the output.

# Slope_A: m = 45954/45945 = 0.070237 = 1.0002
# Offet_A: b = y - m*x = 57194 - (11250 * 1.0002) = 57194 - 11216 = 45942

import machine
import utime

# Configure ADC0 on GPIO pin GP26 (Channel 26)
adc = machine.ADC(machine.Pin('GP26'))

# Configure PWM0 on GPIO pin GP0 (PWM0A)
pwm = machine.PWM(machine.Pin('GP0'))

# Set PWM frequency to 50 Hz (20 ms period)
pwm.freq(50)
sensor_temp = machine.ADC(26)

while True:
    # Read analog value from ADC0
    analog_value = sensor_temp.read_u16()

    duty_cycle = int(((analog_value)/14.2377) + 2436.84)

    # Check if duty_cycle is below 5% and limit it to 5%
    if duty_cycle < 3277:
        duty_cycle = 3277

    # Check if duty_cycle is above 10% and limit it to 10%
    if duty_cycle > 6554:
        duty_cycle = 6554

    #print(duty_cycle)

    # Set PWM duty cycle
    pwm.duty_u16(duty_cycle)

    # Wait for a short time to update the PWM output (adjust as needed)
    utime.sleep_ms(100)
```

Smart e-Bike Throttle Code

This is the original this is the first version of the Smart Throttle, employing a cruise-control feature.

Table 16 – Smart Throttle Code

```
# File: Smart_Throttle_Analog_output_rx3.py
# By: Richard R. Vasquez
# Date: 10/19/2024
#
# Notes: This version is nearly perfected and works well enough for road testing.
# It seems that it goes into Hold mode too easily, as if the Array average is too close.
# This version aims to tackle that problem.
#
# Works well with PWM and Analog modes now.
# Analog output now works. The MINPWM and MAXPWM were clipping the signal from 5 to 10%
# The Brake signal is fixed now.
# Commented out all print() instructions.
#
# For use with Throttle ID #2
# The ISR and main() were swapped because it made more sense to have the more complex
# function in main() and the simple update function in the ISR.
#
# Operation:
# Map the analog value to the PWM duty cycle (0-65535)
# Assuming the analog range is 0-3.3V (for 3.3V ADC reference)
#
# Observation of throttle: Full speed = 3.84 volts, Min speed = 0.76 volts
# Used 0.75 scale down, since 3.8 > 3.3 range of ADC reference voltage.
# Scaled down 0.75x Throttle: Full speed = 2.88 volts, Min speed = 0.566 volts
# ADC: 50.355 uV/step
# Delta steps: 57195-11250 = 45945   This is the range of the throttle.
# Delta PWM = 6554-3277 = 3277   100% - 50% = 50%   This is for the output.
#
# OPTION 1: This is for a PWM output.
# 10% of 65535 = 6553.5 approximately 6554
# 5% of 3277 (approximately)
# Full throttle range corresponds to PWM range from 5% to 10% (50Hz, 1mS to 2mS)
# Slope: m = 3227/45945 = 0.070237 = 1/14.2377
# Offet: b = y - m*x = 3227 - (11250/14.2377) = 3277 - 790.156 = 2436.84
#
# OPTION 2: This is for an analog output to support low-cost ESC's
# We can easily convert PWM to analog using a low-pass RC filter. Fc = 5Hz, R = 1Mohm, C = 0.2uF
# We just feed the ADC value to the PWM with no changes to recreate the analog singnal.
# Slope_A: m = 1 and Offet_A: b = 0

import machine
import utime
import array
```

```
# Predefine constants
STOPPED = 12153         # Throttle min. voltage = 0.612V
FULLSPEED = 60927       # Throttle min. voltage = 3.068V
ONE_MPH = 13895         # ADC value for 1 MPH
THREE_MPH = 16163       # ADC value for 3 MPH
DELTA_1MPH = 1742       # ADC threshold for 1 MPH change
DELTA_HALFMPH = 871     # ADC threshold for 0.5 MPH change
LED_ON = 0              # LED is active low, On
LED_OFF = 1             # LED is active low, Off
BRAKE_ON = 0            # Brake input: active low

# For Analog Output
MINPWM = STOPPED        # PWM set to ADC @ min throttle (analog mode)
MAXPWM = FULLSPEED      # PWM set to ADC @ full throttle (analog mode)
SLOPE = 1               # Throttle response slope for Analog output. This doesn't work right!
OFFSET = 0              # Throttle response offset for Analog output.

# For PWM Output
#MINPWM = 3277          # PWM set to 5%    (PWM mode)
#MAXPWM = 6554          # PWM set to 10%   (PWM mode)
#SLOPE = 1 / 14.884     # Throttle response slope for PWM output.
#OFFSET = 2461          # Throttle response offset for PWM output.

# Program stability constants
PWMFREQ = 50            # PWM frequency = 50Hz
ISR_FREQ = 10           # ISR frequency = 10Hz
ARRAY_SIZE = 100        # 10 seconds * ISR_FREQ = 100 samples

# Mode constants
IDLE = 0                # Idle mode
RUN = 1                 # Run mode
HOLD = 2                # Hold mode

# RP2040 Zero pin assignments
ADC_PIN = 26            # ADC on GP26 (Throttle Input)
PWM_PIN = 0             # PWM on GP0 (Throttle Output)
BRAKE_PIN = 1           # Brake sense on GP1 (Brake connector signal)
LED_PIN = 8             # GPIO 8 for LED when mode = HOLD

# Predefine devices
adc = machine.ADC(machine.Pin(ADC_PIN))  # ADC on GP26
pwm = machine.PWM(machine.Pin(PWM_PIN))  # PWM on GP0 (Throttle Output)
pwm.freq(PWMFREQ)     # Set PWM frequency to 50 Hz (20ms period)

# Initialize array and variables
adc_array = array.array('i', [0]*ARRAY_SIZE)  # Circular array to hold 10 ADC samples
index = 0           # Array index (0-9)
```

```
avg = 0             # Average value of array elements
mode = IDLE         # Start in IDLE mode9
speed = STOPPED     # Default speed
hold_speed = STOPPED  # Default hold speed
adc_value = 0       # Latest ADC value
duty_cycle = MINPWM   # Initial duty cycle

# Brake input pin on GP1 with internal pull-up resistor
brake_pin = machine.Pin(BRAKE_PIN, machine.Pin.IN, machine.Pin.PULL_UP)

# LED pin configuration on GPIO 8
led_pin = machine.Pin(LED_PIN, machine.Pin.OUT)

# Function to read the ADC value (Throttle input)
def read_adc():
    return adc.read_u16()

# Function to update the PWM output
def update_pwm(duty_cycle):
    pwm.duty_u16(duty_cycle)

# Function to flush the array (reset all values to 0)
def flush_array():
    global adc_array
    adc_array = array.array('i', [0]*ARRAY_SIZE)  # Reset array to all 0's

# Function to calculate average of the array
def calculate_average():
    return sum(adc_array) // len(adc_array)

# New ISR (formerly the main loop's speed and PWM updates)
def timer_isr(timer):
    global mode, speed, duty_cycle

    # Update speed and PWM duty cycle based on mode
    if mode == IDLE:  # IDLE mode
        speed = STOPPED
        led_pin.value(LED_OFF)  # Turn off the LED to start
    elif mode == HOLD:  # HOLD mode
        speed = hold_speed
    else:  # RUN mode
        speed = adc_value

    # Convert speed to PWM duty cycle
    duty_cycle = int(speed * SLOPE + OFFSET)
    #print("Duty Cycle: ", duty_cycle)

    # Constrain duty cycle to the 5% - 10% range for PWM, or ADC_value min/max
```

```
    if duty_cycle < MINPWM:
        duty_cycle = MINPWM
    elif duty_cycle > MAXPWM:
        duty_cycle = MAXPWM

    # Update PWM output
    update_pwm(duty_cycle)

# New main loop (formerly the ISR logic, but now loops forever)
def main():
    global mode, index, hold_speed, avg, target, adc_value

    # Main loop with 0.1S delay
    led_pin.value(LED_ON)  # Turn off the LED to start
    while True:
        # If brake is applied, go to IDLE mode
        utime.sleep(0.1)
        adc_value = read_adc()  # Continuously update the ADC value
        # print("ADC: ", adc_value)
        if brake_pin.value() == BRAKE_ON:  # Brake is active low (IDLE mode)
            mode = IDLE
            #print("IDLE mode")
        else:
            # Mode transitions and ADC handling
            if mode == IDLE and adc_value > THREE_MPH:  # Exiting IDLE mode
                mode = RUN  # Switch to RUN mode
                #print("RUN mode")
            elif mode == HOLD and adc_value > target:  # Exiting HOLD mode
                mode = RUN  # Switch back to RUN mode
                #print("RUN mode")
                led_pin.value(LED_OFF)  # Turn off the LED when leaving HOLD mode
                flush_array()
            elif mode == HOLD and adc_value < ONE_MPH:
                target = THREE_MPH
            else:
                # Must be in RUN mode by default
                adc_array[index] = adc_value  # Add current ADC value to array
                index = (index + 1) % ARRAY_SIZE # Circular buffer logic
                avg = calculate_average()  # Calculate average

                # Check if speed stabilizes within +/- 0.5 MPH to enter HOLD mode
                if (abs(adc_value - avg) < DELTA_HALFMPH) and (avg > ONE_MPH):
                    mode = HOLD  # Enter HOLD mode
                    #print("HOLD mode, Avg: ", avg, "ADC: ", adc_value)
                    led_pin.value(LED_ON)  # Turn on the LED when entering HOLD mode
                    hold_speed = adc_value
                    target = adc_value + ONE_MPH
                    flush_array()
```

```
# Timer to run the new ISR
timer = machine.Timer()
timer.init(freq=ISR_FREQ, mode=machine.Timer.PERIODIC, callback=timer_isr)

# Start the main loop
main()
```