

NearZero™ NZ1

Brushless Motor Controller for Robotics
www.skysedge.us

Quickstart

1. Connect motor(s)

- Connect the motor's main power wires using the motor output screw terminals for channel 1, 2, or both. Unconnected channels are ignored after power up.
- Wire order doesn't matter, but if the motor spins the wrong way you can reverse any two of the three leads.

2. Connect control cable(s)

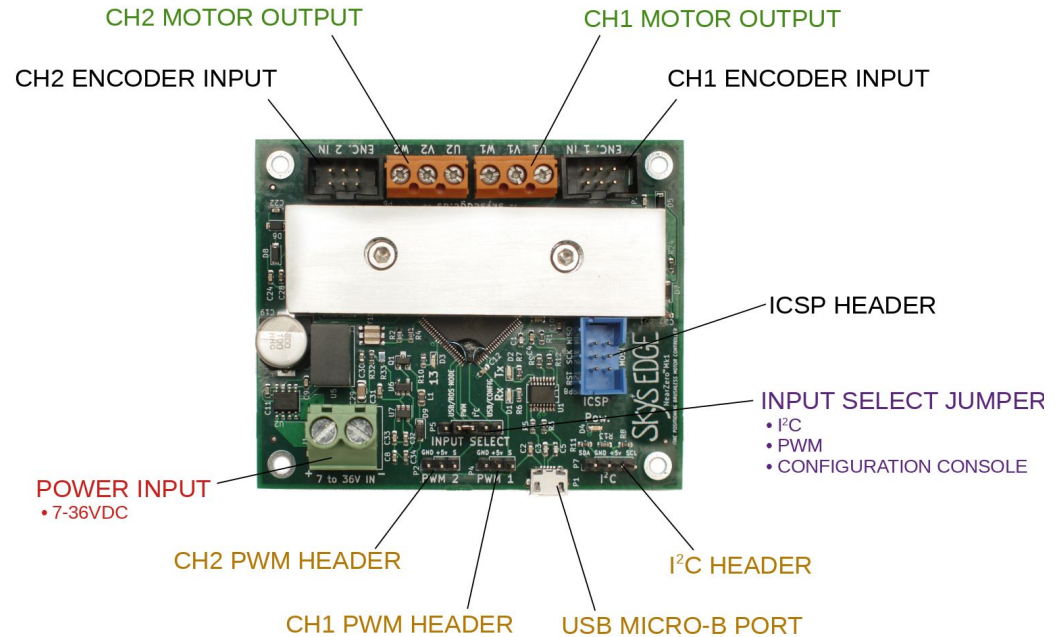
- For I²C control, use the included cable or your own 4-pin header connector. Get up and running with the Python examples (Section 6).
- For PWM control use a standard servo cable for channel 1, 2, or both.

3. Set jumper for desired command input type

- Position the mode-select jumper for either PWM or I²C input, or for configuration (via USB). For control from a single-board computer (e.g. Raspberry Pi) use I²C is the clear choice
- The board will switch to the selected mode upon power up or reset.

4. Connect power

- Any DC power source from 7 to 36V (e.g. Lithium, Lead Acid, a DC wall adapter, etc)
- Be sure the power source can supply enough current for your application.
- The board is ready to go when the amber LED turns off.



NOTE: The configuration console explained below (section 8) is required for setting the I²C address and all settings relating to PWM control.

Overview

The NearZero (NZ) controller allows fine (arbitrarily slow in either direction) or positioning control of brushless motors for direct drive applications. Intended usage includes controlling hub motors for domestic robots, self-balancing devices, actuators for manipulators and robotic arms, and motorized or stabilized gimbals and mounts. Any brushless motor requiring between 7 and 36V will work with this controller, though the quality of slow motion control depends on the motor used (see section 1). The maximum continuous current is 1A per channel with active cooling, though excursions peaking at 3A are permissible. When controlled by PWM the running and resting currents are set in the configuration console according to section 8. Alternatively, control by I²C permits continuous real time current control. The 25kHz switching frequency puts the running tone well out of the human hearing range, resulting in totally silent "squeal-free" operation.

Whereas typical brushless motor controllers require hall sensor or back-EMF (sensorless) feedback from the motor in order to trigger the commutation sequence, the NearZero's fine-positioning is accomplished via a sine commutation sequence that is not informed by a hall sensor. A caveat is that while conventional controllers can run a motor at almost arbitrarily high speed, if a certain speed is surpassed with the NearZero controller (which depends on the load, current setting, supply voltage, and on the motor itself) the motor will stall. A stall condition is not harmful to the motor or controller, but it is the engineer's responsibility to avoid this condition with appropriate power configuration and speed envelopes as demanded by the application, or by implementing active feedback using encoder data. The encoder and hall input connectors can be used with a motor having either a built in quadrature encoder or with an external encoder suitably mounted to the motor. These inputs can be used to report encoder or hall ticks over I²C for external closed-loop control, or for internal closed-loop control in the "servo" positioning mode.

1. Motor Selection and Connections

The phases for each motor output terminal block are marked on the silkscreen as U1, V1, W1 for channel 1 and U2, V2, W2 for channel 2. Wire order for 3-phase (brushless) motors doesn't matter but if the motor spins opposite the desired direction for a given command input, any two of the three leads can be reversed. Alternatively, the configuration console (see section 8) gives one the option to set the direction for each channel without having to reset wires.

The quality of slow motion control depends heavily on the motor used. Many motors will exhibit cogging (torque ripple), whereby they visibly detent into the pole positions at low speed. The smoothest motion is obtained with motors intended for such operation, like hub motors and gimbal motors. In general such motors have a high pole count and relatively high winding resistance. An experimental feature included in the configuration console (see section 8) allows the selection of asymmetrical non-sine commutation waveforms that may reduce cogging on motors not normally intended for smooth positioning operation, but this feature will require extensive trial and error on the part of the user.

Hot plugging motors is not supported mainly due to an initialization routine that the board performs at start time, described in section 4. Channels are automatically disabled that don't have a motor connected during power up. Conversely, swapping one motor with another without restarting the board may result in an undetected, potentially dangerous over-current condition.

2. Controlling the Board

Two input types are available: I²C for control from single-board computers (detailed in section 6) and PWM for control from the R/C ecosystem (detailed in section 5). In either case motors can be commanded by *velocity*, *position*, or *servo* (closed-loop-position):

Velocity -- Command inputs vary motor velocity.

Position -- Command inputs vary motor position.

Servo -- Like *position* mode except that an encoder (if installed) is used for on-board closed-loop position feedback, analogous to a servo motor.

I²C input is the recommended control method for robotics applications not only because it allows more precise control than PWM inputs, but also because it allows real-time control of the current (power) setting and instantaneous switching between velocity/position/servo modes. What's more, I²C cabling is well suited to chassis wiring situations (as opposed to the would-be awkwardness of running a USB cable to each controller board) and is intended for extensibility (one can run dozens or even hundreds of NearZero boards with addresses on one or more I²C busses). Encoder ticks, when an encoder is installed, are also reported via the I²C bus, thus the roboticist is empowered to implement effort control and odometry. Extremely accessible Python example files are provided for out-of-the-box functionality with a Raspberry Pi or Nvidia Jetson Nano, and the I²C address is set via the configuration console (see section 8).

PWM input allows the NearZero to be used with a PWM-based control source, like an RC receiver, drone/autopilot hardware, servo-tester, arduino, robotics control board, or any device capable of generating a PWM signal. The PWM inputs are standard servo-style 3-pin .1" headers. For a motor to operate in a positioning fashion in place of a conventional servo motor, control must be switched from *velocity* command to either *position* or *servo* command

In the configuration console. Also be mindful that the motor current/power must be configured to be high enough that the motor won't stall or skip under load, as PWM input does not give real-time control over current. When using PWM input the command type, resting current, and running current is configured for each channel separately using the configuration console, detailed in section 8.

NOTE: The +5V pins on NearZero's PWM headers are energized, meaning the board will power attached 5V PWM devices eliminating the need for a separate 5V power source like a BEC when using control sources like an RC receiver or servo tester. If an attached PWM device is powered separately in addition to being attached to the NearZero, the 5V rails of both devices will be tied. If this is undesirable the +5V (center) wire on the PWM cable can be omitted (or cut) without sacrificing functionality.

3. Setting Control Input Type

Although general configuration is done within the configuration console described in section 8, changing the command input source is sufficiently fundamental to have warranted a hardware jumper, called the *input select* jumper. It is here that one switches between I²C input, PWM input, or the USB port for use with the configuration console.

The "RESET" position on the *input select* jumper initiates a software reset when touched momentarily with the jumper if the NearZero is in either I²C or PWM mode. The jumper should not be left in this position. With the NearZero in *any* mode (including the configuration console), the board can also be reset by plugging (or replugging) it into a computer via the USB cable and initiating a serial connection (described in section 8).

4. Supplying Power

The NearZero will run with any power source supplying between 7 and 36V and can be set to power motors from 0A to 1A continuous current with 1mA resolution, and up to 3A peak for short durations when commanded from I²C. The board will power on as soon as voltage is supplied to the input power terminal block in Figure 1, and will enter whichever input mode is set by the *input select* jumper. Connecting a USB cable will energize the board for the purpose of using the configuration console, but in general a motor cannot be powered by USB power alone.

To reduce manufacturing cost the NearZero employs a single-channel current sensing circuit. Consequently, instead of monitoring (and regulating) the current draw of each channel continuously during operation, each channel's power is ramped up to .5A one at a time after the board is energized. This initialization routine lasts about 10 seconds and is necessary for the NearZero to determine the correct duty cycle needed for each channel to draw the configured degree of current; a relationship which is unique to the specific motors used. If a motor is not connected at start-time, the corresponding channel will be left off. Also, as the NearZero has no means to maintain a fixed output power to compensate for supply voltage variation, voltage should not be allowed to vary more than would be expected of a discharging battery. This initialization routine starts automatically as soon as the board is energized and is indicated by the amber "13" LED. The board is ready to go when that LED turns off.

Active cooling (e.g. an external fan) or more rigorous heat sinking should be considered when the board's total continuous current draw approaches 1A and is considered necessary if each channel will be drawing 1A (2A total).

5. Control by PWM

In PWM mode the NearZero can be paired with an R/C receiver or similar device as shown in Figure 2. Standard 3-wire “servo cables” connect to the PWM headers, where the GND, +5, and Sense positions are clearly marked on the NearZero’s silk screen. In this mode, out-of-the box functionality defaults to *velocity-command* at 300mA with 200mA resting current with sensor type set to *encoder* for both channels. This is meant to be a relatively conservative default state for initial “sanity check” testing. All PWM-specific settings are made through the configuration console, described in section 9. If PWM control results in a vibrating motor or a motor that can’t be stopped from turning, the PWM center OFFSET parameter may need to be changed.

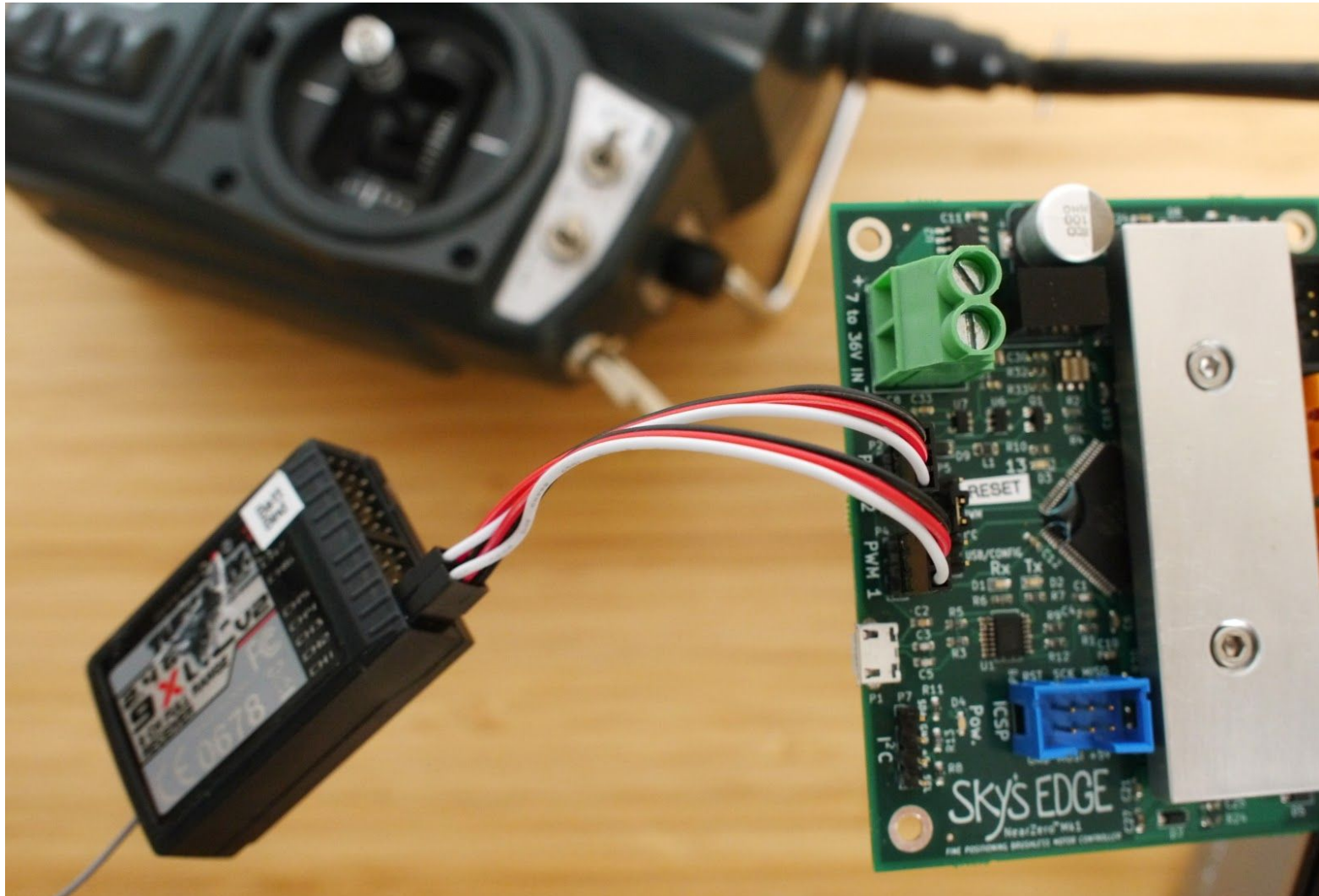


Figure 2

6. Control by I²C

Making the connections:

Control by I²C is easy using a single-board computer. The Python examples at <http://skysedge.us/robotics/nearzero/python> provide a starting point for controlling the NearZero on a Raspberry Pi or NVIDIA Jetson, and implementation on any other computer sporting I²C pins should be straightforward. The connections can be made with one's own cabling or the supplied cable parts according to Figure 3 (note that the Raspberry Pi and NVIDIA Jetson Nano both use the same pin mapping on the GPIO header). The nice thing about I²C is that up to 119 devices can be connected to a single SCL/SDA pin pair, so connecting more than one NearZero is as easy as making the connections in parallel. An example of this is shown in supplementary Figure S1, where three NearZero I2C cables connect to a perfboard where they're bussed to a single I2C header on an NVIDIA Jetson Nano.

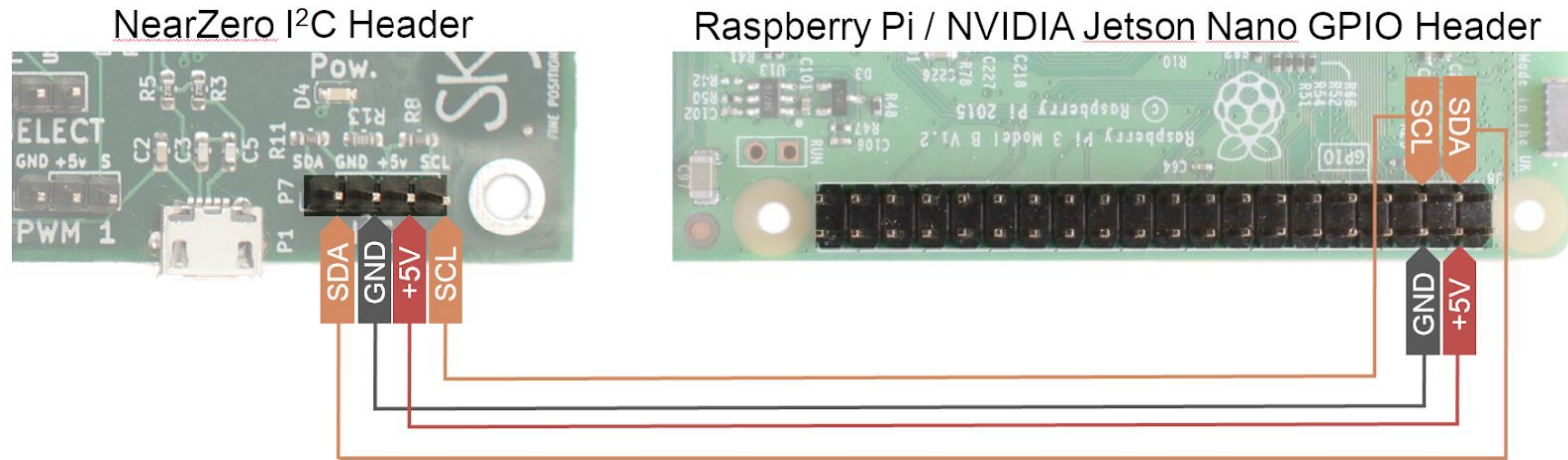


Figure 3

The supplied connectors are IDC (insulation displacement) connectors as shown in Figure 4. In lieu of an expensive specialty crimping tool these can be terminated using a flat-head screwdriver to press the (unstripped) wire in place and a needle-nose pliers to bend the holding tabs over, after cutting the ribbon cable to the desired length. Figure 5 depicts this cable being used to connect a NearZero to a Raspberry Pi.



Figure 4



Figure 5

Install the Python examples:

Several example Python files are available at https://github.com/jhaupt/nz1_python. It is assumed that you have a working single-board computer with Python installed:

1. Copy the files onto the single-board computer: From the directory of your choice run `git clone https://github.com/jhaupt/nz1_python.git`
2. Install the libi2c Python library by amaork: Run `git clone https://github.com/amaork/libi2c.git` and from within the libi2c directory do `sudo python setup.py install`
3. Enable the I²C interface:
 - a. On a Raspberry Pi run `sudo raspi-config` and navigate to Interfacing Options > I2C > Yes
 - b. On an NVIDIA Jetson Nano run `sudo pip install Jetson.GPIO`
4. Once the NearZero(s) are connected it's a good idea to run `i2cdetect -r -y 1` to display all connected I²C devices. The NearZero(s) should be present in the "grid" display at the assigned address(es)

Using the Python examples:

There are four python scripts to play with, steal code from, use as boilerplate, or whatever:

<i>supersimple_example.py</i>	-- The absolute minimum code needed to make the NearZero do something.
<i>simple_example.py</i>	-- A more advanced but still simple example that defines a class and method to control the NearZero.
<i>odometry.py</i>	-- Prints encoder ticks. It can be run at the same time as any of the other examples because unlike USB, many programs can access the same I2C bus simultaneously.
<i>teleop_keyboard.py</i>	-- A simple implementation of real time keyboard control.

These files are succinct and are commented so as to be self-explanatory. Still, some notes are warranted.

In both ***supersimple_example.py*** and ***odometry.py*** the following code block should be included for each connected NearZero. An intuitive name is defined (in this case the NearZero is used to control two differential drive hub motors, so we call it "wheels") and the I²C address is set (in this case 0x40). Nothing aside from the name and address needs to be changed when adding another device.

```
#-----CONNECT TO NZ AT GIVEN ADDRESS (e.g. 0x40) AND GIVE IT A NAME (e.g. "wheels")-----
wheels = pylibi2c.I2CDevice('/dev/i2c-1', 0x40)
wheels.delay = 10
wheels.page_bytes = 16
wheels.flags = pylibi2c.I2C_M_IGNORE_NAK
```

In ***supersimple_example.py***, writing commands to a NearZero called "wheels" is as easy as:

```
wheels.write(0x0, '1v+00100c00200') #Drive channel 1 at velocity=100 with 200mA of current
wheels.write(0x0, '2v+00250c00320') #Drive channel 2 at velocity=250 with 320mA of current
```

Where the command syntax is as follows:

```
wheels.write(0x0, '2v+00250c00320')
```

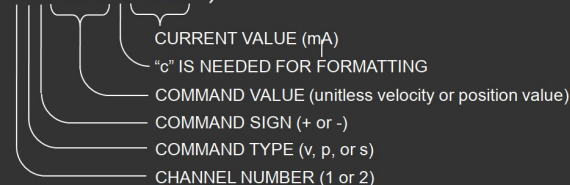


Diagram illustrating the command syntax breakdown for the example command: `wheels.write(0x0, '2v+00250c00320')`

- CHANNEL NUMBER (1 or 2)
- COMMAND TYPE (v, p, or s)
- COMMAND SIGN (+ or -)
- COMMAND VALUE (unitless velocity or position value)
- 'c' IS NEEDED FOR FORMATTING
- CURRENT VALUE (mA)

In ***simple_example.py***, things are even more straightforward with the implementation of a class called "nz". In the "Instantiate the joints" section, simply define each motor by giving it a name and specifying the channel and address of the NearZero it's connected to like so:

```
#-----INSTANTIATE THE JOINTS-----
LeftWheel = nz("LeftWheel",0x40,"1")
RightWheel = nz("RightWheel",0x40,"2")
```

In this way thinking of thinking about the NearZero boards, we think of each motor as being identified by the I²C address of the NearZero it's attached to and its channel number.

To send commands to a motor we then use the *write* method which takes three arguments: the command type (*v* for velocity, *p* for position, or *s* for servo), the command value (an integer), and the current value in mA (an integer). The following code from ***simple_example.py*** sets the velocity of both LeftWheel and RightWheel to 300 (a unitless number) and the current to 200mA:

```
#-----MAKE THE NEARZERO DO THINGS-----
vel1 = 300 #define velocity [unitless]
vel2 = 300
I1 = 200 #define current [mA]
I2 = 200
LeftWheel.write('v',vel1,I1) #write the commands
RightWheel.write('v',vel2,I2)
```

From this it should be easy to imagine building up more advanced code which interfaces with a navigation stack either with stand-alone code or via the ROS. The file ***teleop_keyboard.py*** is based on ***simple_example.py*** and allows direct keyboard teleoperation of one or more NearZeros.

If you've connected a quadrature encoder or hall sensor as described in section 7, obtaining the "ticks" is shown in ***odometry.py*** and is as easy as:

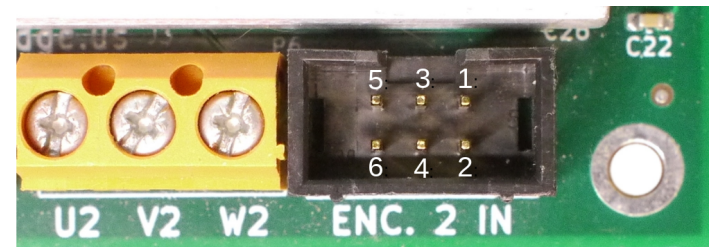
```
data1 = wheels.read(0x0,12)
print(data1)
```

Encoder ticks are reported as 12 integers where the first 6 represent channel 1 and the second 6 represent channel 2. Digits 1 and 7 indicate the sign for channels 1 and 2 respectively, where "0" = positive and "-" = negative. For example, 012523-00427 indicates that encoder 1 is reporting 12523 ticks and encoder 2 is reporting -427 ticks. These values range from -32,768 to 32,767 and count up or down with each increment or decrement of a quadrature encoder or hall sensor, starting at 0 when the NearZero is powered on. The count resets back at the other extreme upon overflow.

7. Connecting Encoders and Hall Sensors

The encoder input connectors are 6-pin shrouded headers for use with a 6-pin keyed IDC connector, like Amphenol P/N 71600-006LF available from Digi-Key, Mouser Electronics, and most fine electronics parts suppliers. The connector can be used to attach either a quadrature encoder or a hall sensor needing 5V power, but not both simultaneously. The pinouts are described in Figure 6.

For the *servo* command mode to work, the encoder or hall sensor must be wired so that ticks increment when the motor receives a positive position command and decrement when the motor receives a negative position command. If *servo* mode doesn't behave correctly, any two of the motor's three power wires may be swapped to switch it's direction, thus making it accord with the encoder's sense of positive and negative.



PIN 5: HALL C	PIN 3: ENCODER A / HALL A	PIN 1: GND
PIN 6: NOT CONNECTED	PIN 4: ENCODER B / HALL B	PIN 2: 5V OUT

8. Configuration

Configuration is done using a serial console which provides a menu system for setting all of the NearZero's parameters.

To connect to the configuration console:

1. With the Input Select jumper in the USB/CONFIG position, connect NearZero to a computer using a USB cable. Configuration can be done using USB power alone.
2. Connect to the serial console. Users familiar with serial consoles (either over USB or RS232) can skip this and connect with their accustomed-to method at 9600 baud and typical default settings. Otherwise, use one of the following methods:
 - (a) Using the **screen** command (Linux or MacOS):
 - o In a terminal, run `screen /dev/ttyUSB0` replacing `/dev/ttyUSB0` with the correct USB dev assignment (but there's a good chance it'll be `/dev/ttyUSB0`).
 - o Once connected, use Ctrl+J to send commands.
 - o To exit **screen**, do Ctrl+A followed by K (or sometimes, Ctrl+A followed by Ctrl+K).
 - (b) Using PuTTY (Any OS):
 - o Download, install, and run PuTTY.
 - o Under Session settings set the connection type to "Serial", the Serial Line to the correct USB dev assignment (like `/dev/ttyUSB0`), and Speed to 9600.
 - o Under the Connection>Serial settings everything should be left as default (speed = 9600, which was already covered under the Session settings, Data bits = 8, Stop bits = 1, Parity = None, and Flow control = XON/XOFF).
 - o Click "Open".
 - o Once connected, use Ctrl+J to send commands.
 - o To exit, close the window.
 - (c) Using the Arduino IDE (Any OS):
 - o Run the IDE.
 - o Go to Tools > Serial Monitor
 - o Once connected, use ENTER to send commands.
 - o To exit, close the window.

Using the configuration console:

When the connection is first established the NearZero will summarize all current configuration settings starting with the I²C address, followed by channel 1 settings, and then channel 2 settings. After this there will be a long pause while the controller detects whether any motors are connected and performs a calibration if so. If external power is not applied, the NearZero will report that no motors are connected whether or not they are.

After this the Main Menu appears. Type the number of the option you want and press either Ctrl+J or ENTER to send it (depending on how you connected, see above). All options should be self-explanatory and sufficiently explained within the menu system, but a few configuration examples are provided here:

```
-----Welcome to the NearZero Configuration Terminal-----
      Firmware version 2.0 -- by J. Haupt
      CHANGES TAKE EFFECT AFTER CYCLING POWER
                        or RESETTING

1: Channel 1 configuration
2: Channel 2 configuration
3: Set I2C Address
4: List current settings
5: Reset factory defaults
q: Quit/restart (use after changing INPUT SELECT jumper as desired)

>>
```

Some configuration examples:

- To set the I²C address of the board to 98 (hex value = 0x62): Send "3", then "98".
- To set the Channel 1 PWM running current to 1A and resting current to .25A: Send "1", then "6", then "1000", then "250".
- To set PWM command gain on Channel 2 to 90 (on a scale of 1-1000): Send "2", then "8", then "90".

Complete list of configuration settings:

GENERAL SETTINGS

- Set I2C Address** -- Set the I²C address of the board in decimal (not hex) form as an integer between 3 and 119.
List current settings -- Displays all current configuration settings. This is the same readout that appears automatically every time the board is started and a serial console is open.
Reset factory defaults -- Returns all settings to their original values.
Quit/restart -- May be used after setting the *input select* jumper to restart the board into one of the control modes (I²C or PWM) without needing to cycle power.

CHANNEL-SPECIFIC SETTINGS

- Set SENSOR type** -- If no sensor is connected this can be ignored. Switch to *encoder* to use a quadrature encoder or *hall* to use a hall sensor. Encoder and hall sensor “ticks” are accumulated internally and reported over the I²C bus so that an external computer can handle closed loop control of the NearZero as informed by a robot’s navigation stack or similar. Additionally, the *servo* command mode makes use of the encoder input to allow the NearZero to function as a stand-alone closed loop position controller.
- Set max position/servo SLEW RATE** -- When a motor is used in *position* or *servo* mode this sets the rate at which the NearZero will drive the motor to match the commanded position.
- Set position/servo ACCELERATION** -- Sets the rate at which a motor accelerates to the SLEW RATE from a stop.
- Set directionality** -- Inverts the direction for *velocity*, *position*, or *servo* commands. If an encoder or hall sensor is connected and *servo* mode behaves incorrectly, changing the *directionality* parameter will NOT fix it. In this case any two of the motor’s three power wires must be reversed to make the motor’s directionality accord with that of the encoder.
- Set waveform TORQUE SMOOTHING** -- An experimental feature which adds a two-term Fourier series to the commutation waveform to negate torque ripple (cogging) on certain motors.
- Set PWM running/resting CURRENT** -- Set the running and resting current limit in mA (up to 3A) when running in PWM mode.
- Set PWM command type** -- Set whether PWM command inputs are interpreted as *velocity*, *position*, or *servo*. When using *servo* command mode, if the motor “runs away” instead of holding position, it’s likely that the encoder sign is reversed from the motor direction sign. To fix this either reverse two of the motor power wires or the two encoder sense wires.
- Set PWM command GAIN** -- Sets the gain of PWM input signals. That is, this sets how fast (or far) a motor will turn for a given PWM signal.
- Set PWM center offset** -- Sets the PWM pulse width that the NearZero considers to be neutral. The default settings assume a standard servo PWM-source is providing inputs, but a non-standard PWM source may require that a different pulse-width be considered as zero velocity. If this is troublesome the AUTO mode is available as a catch-all and works by taking the initially-detected PWM value as neutral.

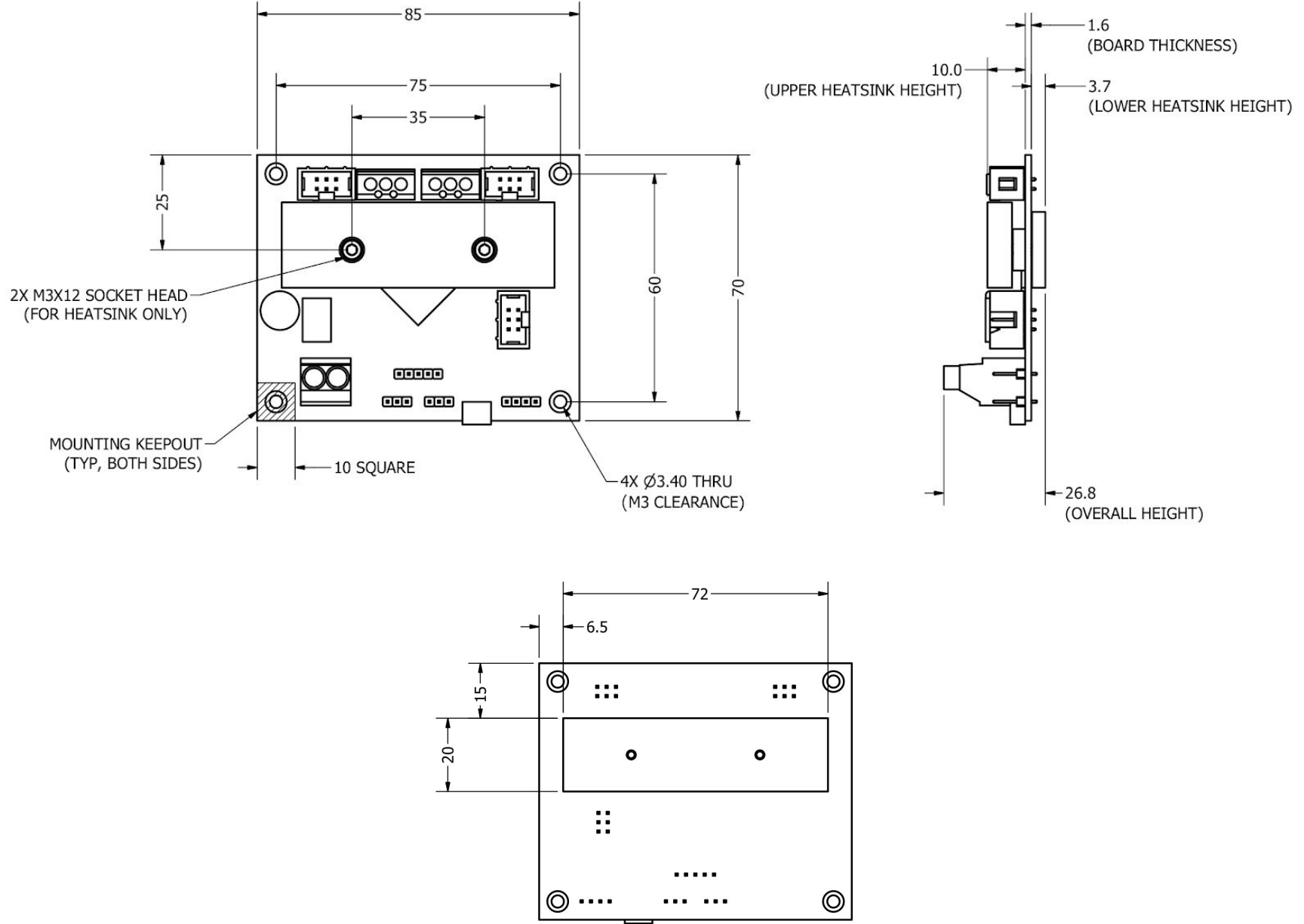
9. Board Architecture, Firmware Modification, and Firmware Updates

The NearZero uses an ATmega2560 microcontroller and is programmed using the Arduino IDE as an Arduino Mega 2560. Unlike typical Arduino-like boards which can be programmed either via the USB port or an In-System Programmer (ISP), the NearZero *must* be programmed using an ISP via the ICSP header. I use the OLIMEX AVR-ISP-MK2 which goes for about \$23 from Digi-Key or Mouser. Those unaccustomed to Arduino programming in this way may quickly find they prefer the ISP to the USB port for several reasons: (a) The port assignment (e.g. /dev/tty/USB0, COM1, etc) doesn’t need to be set. (b) The ISP always just works. The board will never fail to respond or hang up, as sometimes happens in confusing ways with the USB connection. (c) The hot-key for uploading a sketch via ISP is just Ctrl+Shift+U instead of Ctrl+U. (d) If a serial connection is open over the USB port for debugging, flashing firmware with an ISP won’t interrupt that connection. This makes rapidly debugging code using a serial console much more efficient.

The NearZero firmware is written in a way that makes it accessible to novice programmers. The code attempts to be readable and well commented. The stock firmware attempts to foresee every possible usage-case and presents a well-featured configuration console to provide flexibility. Even so, if a user/engineer finds that the stock firmware cannot be made to meet their needs, I would encourage making the firmware “your own” with the tacit understanding that Sky’s Edge cannot be held responsible for damaged hardware. The latest version of the official NearZero firmware is available at <https://github.com/jhaupt/NearZero1>.

10. Mechanical Layout

All dimensions are "reference". All units are mm:



11. Supplementary Material

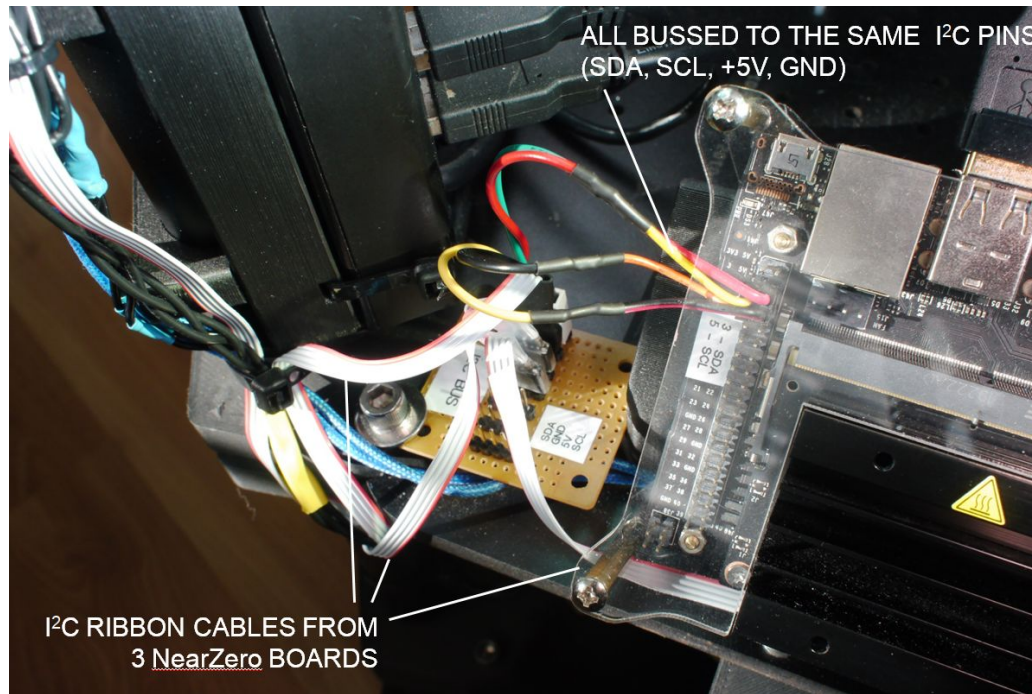


Figure S1: Three NearZero I2C cables connected to a perfboard where they're bussed to a single I2C header on an NVIDIA Jetson Nano.