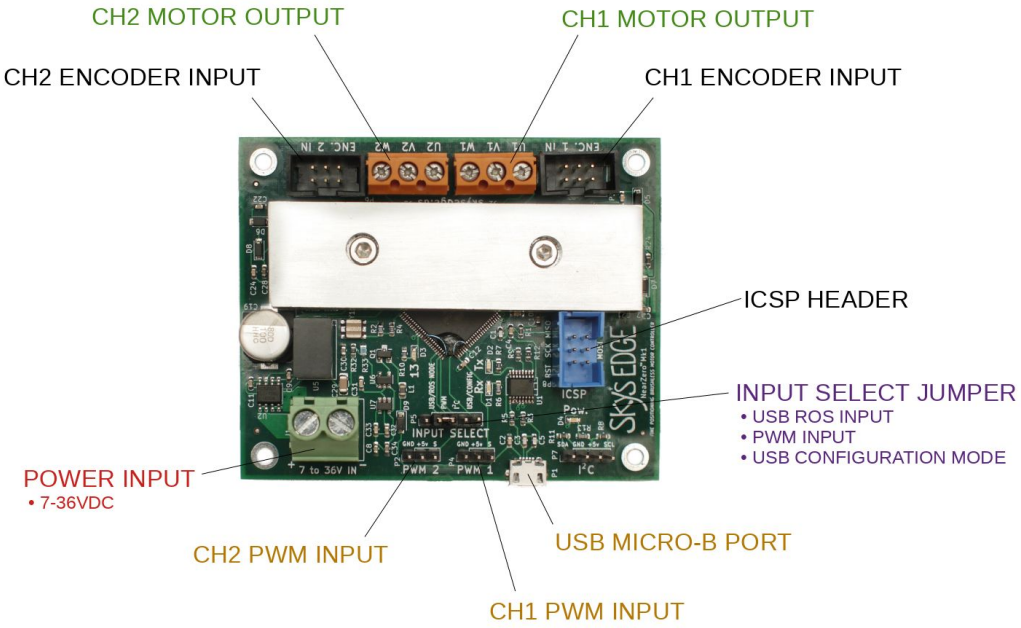
# NearZero™

*Brushless Motor Controller for Robotics*

www.skysedge.us




## Quickstart

1. Connect motor(s)
   - Use the three 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 PWM control use standard servo cable for channel 1, 2, or both. The silk screen indicates GND and Sense positions to inform correct cable orientation.
   - For direct Robot Operating System control, use the USB port.
3. Set jumper for desired command input type
   - Position the mode-select jumper for either PWM or ROS input, or for Configuration.
   - 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 power source can supply enough current for your configuration.

NOTE: Setup via the configuration console (5), explained below, is not considered optional. Out-of-the box functionality defaults to velocity-command at 300mA with 200mA resting current for both channels. This is meant to be a relatively conservative default state for initial "sanity check" testing and is unlikely to be well suited for real-world use.

## Overview

The NearZero (NZ) controller allows fine, slow motion, or positioning control of all 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 0 and 3A and between 7 and 36V will work with this controller, though the quality of slow motion control depends on the motor used (see section 1). An experimental feature included in the configuration console (see section 5) allows the selection of asymmetrical non-sine commutation waveforms that may reduce cogging on motors not intended for smooth positioning operation, but this feature will require extensive trial and error on the part of the user.

Whereas typical brushless motor controllers require hall sensor or back-EMF (sensorless) feedback from the motor in order to trigger the commutation sequence, the fine positioning control for which the NearZero controller is intended is accomplished via open loop sine wave commutation. 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, configuration parameters, 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 as demanded by the application. The encoder 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, but it should be noted that feedback from this signal is not used to trigger commutation directly. Rather, these inputs allow encoder ticks to be reported to the ROS for odometry, which in addition to navigation could be used within ROS to slow the velocity or position commands sent to the NearZero controller to prevent stalls.

# Detail

## 1. Motor Selection and Connections

The phases for each motor output terminal block are marked on the silkscreen (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 the wrong way you can reverse any two of the three leads. Alternatively, the configuration console (see section 5) 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 5) 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.

The power (in Watts) an individual motor draws is a function both of the configured current limit (see section 5) and of the supply voltage according to the electrical power law $P = IV$.
As the board doesn't monitor the supply voltage, determination of the correct current to obtain the desired power is left to the engineer/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

Whether using the PWM inputs or ROS directly via the USB port, motors can be commanded by either velocity or position. The command type is configured for each channel separately using the configuration console, detailed below in section 5. When configured for position command the motor's behavior is servo-like, moving to a set position proportionate to the commanded input. When configured for velocity command, command input sets velocity. In either case the gain of the command inputs can also be set in the configuration console.

When using the PWM input source the NearZero board can be used with a PWM-based control source, like an RC receiver, robotics control board, drone/autopilot hardware, servo-tester, arduino, 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, keep in mind that control must be switched from velocity command to position command in the configuration console. Note that unlike a true servo motor, position control is not closed-loop; the motor current/power must be configured to be high enough that the motor won't stall or skip under load.

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, keep in mind that 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.

The USB port is available for direct control with the Robot Operating System (ROS). Instructions for using ROS to control the NearZero board will be found in section 6. Naturally, the NearZero can also be used in a ROS robot in the PWM input mode described above with the use of a separate robotics controller that has PWM outputs. This may be preferred for robots that already make use of PWM-controlled drive systems wherein the NearZero board can serve as a drop-in replacement to, for example, use hub motors on a Turtlebot.

## 3. Changing Control Input Type

Although all configuration is done within the configuration console described in section 5, changing the command input source is sufficiently fundamental to have warranted a configuration jumper. It is with this jumper that one switches between PWM input, the USB port for use with the ROS, or the USB port for use with the configuration console. The usable jumper positions are indicated on the silkscreen and are USB/ROS, PWM, $I^2C$, and USB/CONFIG. Note that $I^2C$ is not available at this time but may be enabled in a future software update.

It is safest to change the Input Select Jumper when the board is not powered. The selected input mode will then be entered upon power up.

## 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 3A with 1mA resolution. Resting current can also be set for each channel allowing for lower power consumption when a motor is not moving. Each channel is configured independently and these changes are made via the configuration console detailed in section 5.

Connecting a USB cable will energize the board for the purpose of using the configuration console or debugging ROS communication, but a motor cannot be powered by USB power alone. Also, when removing the main power source for the purpose of connecting a new motor or enacting configuration changes, USB power **must** also be removed to allow the logic circuitry to power down. *Continued.*

# 4. Supplying Power (Continued):

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 one at a time after the board is energized. This initialization routine lasts a few seconds and is necessary for the NearZero to determine the correct duty cycle needed for each channel to draw the configured degree of current. This relationship is specific to the motors that are connected. 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.

The 3-phase bridges used on the NearZero have a published maximum current limit of 5A each, however the NearZero firmware limits the maximum current to 3A as higher current draws may overheat the board if active cooling or a larger heatsink are not supplied. Because the firmware is open source, it would not be difficult for a cognizant user to set a higher maximum current limit. See section 7.

# 5. [SECTION NOT FINISHED] 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. If external power is applied, it must be removed before making the USB connection to allow the board to reset.
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:
    (a) Using the *screen* command (Linux or MacOS): 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**). Once connected, use Ctrl+J to send commands. To exit **screen**, do Ctrl+A followed by K (or sometimes, Ctrl+A followed by Ctrl+K).

    (b) Using PuTTY (Any OS):

    (c) Using the Arduino IDE (Any OS):

```
ROS topic for command input: /cmd_vel
Wheel base scale factor: 50
Wheel diameter scale factor: 50

Channel 1 Settings:
   ROS command mode: DIFFERENTIAL DRIVE
   PWM command mode: VELOCITY
   ROS sensor type: HALL
   ROS topic for hall ticks: nzticks1
   Current: 0.30A.
   Rest current: 0.20A.
   Directionality: NORMAL
   PWM command gain: 60
   Acceleration: 3000.00
   Torque smoothing: OFF

Channel 2 Settings:
   ROS command mode: DIFFERENTIAL DRIVE
   PWM command mode: VELOCITY
   ROS sensor type: HALL
   ROS topic for hall ticks: nzticks1
   Current: 0.30A.
   Rest current: 0.20A.
   Directionality: NORMAL
   PWM command gain: 60
   Acceleration: 3000.00
   Torque smoothing: OFF
```

# 6. [SECTION NOT FINISHED] Controlling via USB using the Robot Operating System (ROS):

The ROS offers tremendous flexibility in how something like an actual physical hardware device (i.e. NearZero) can be controlled, which has the unfortunate side effect that the learning curve is steep. Basic knowledge of how one interacts-with and develops-for ROS will be assumed

In addition to normal configuration settings in the configuration console, certain things need to be configured specifically for operation with ROS.

Set ROS topic by Channel > blah > blah

a. Turn on the ROS master computer (probably an SBC).

NOTE: Always be sure you're on the machine you think you're on (local or remote). Good to dedicate one workspace to the laptop and one to the Pi/SBC.

b. SSH into the master machine (the robot, e.g. ssh pi@rostest, p=ss), run roscore and then use roslaunch to run the bringup script relevant for the master machine. For Tachi this is $ roslaunch tachi_bringup tachi_robot.launch.

NOTE: Bringup scripts launch all the actual/specific launch scripts for various devices. Instead of running a bringup script you could also run each individual launch script separately, I think? Bringup scripts are located at ~/catkin_ws/src/turtlebot3/turtlebot3_bringup/launch

c. Also on the master machine, after launching the bringup script, if there are any additional nodes that needs to be run that weren't included in the bringup, we can run those. For example, serial_node, which lets Arduino-like devices publish and subscribe, can be run with rosrun rosserial_python serial_node.py /dev/ttyUSB0. The USB /dev number might not alsways be ttyUSB0. When it's running properly, $rosnode list should include a result for /serial_node. NOTE: This is now built into the bringup script for Tachi.

d. Next, on the remote machine (the laptop) use roslaunch to run the bringup script relevant to the remote machine. For Tachi this is $ roslaunch tachi_bringup tachi_remote.launch.

# 7. 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.

While the NearZero hardware design isn't open source, the firmware is, and is written in a way that makes it accessible to novice programmers. The code is easy to read, well commented, and is decidedly *not* object-oriented. 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 www.sysedge.us/products/nz1/firmware.