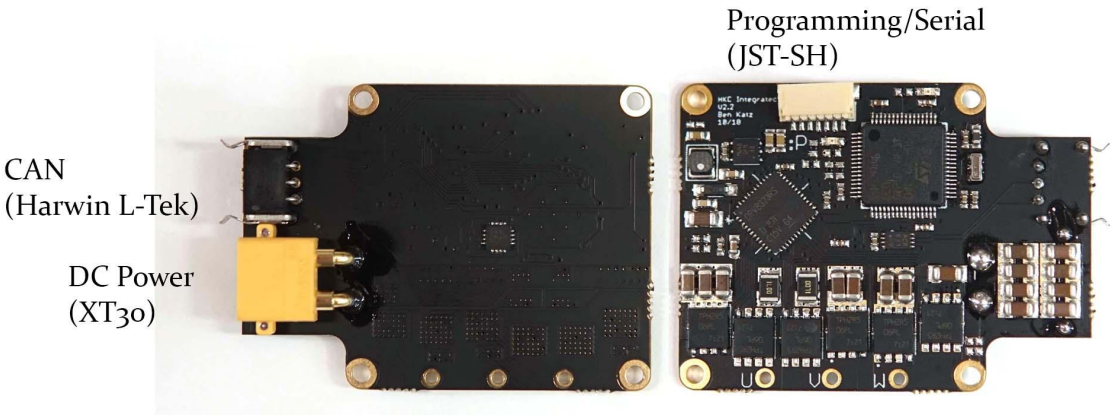
Ben Katz
Motor Drive Documentation
6/29/2019
Last updated 4/15/2021

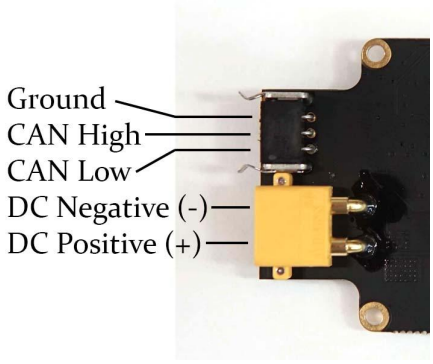
Connectors

The motor drive has two connectors which will be used during normal operation: An XT30 connector for power, and a Harwin Datamate L-Tek High-reliability connector for CAN bus communication. On the top side of the board, there is a JST-SH connector which contains the programming pins and a serial port which can be used for configuring the drive.

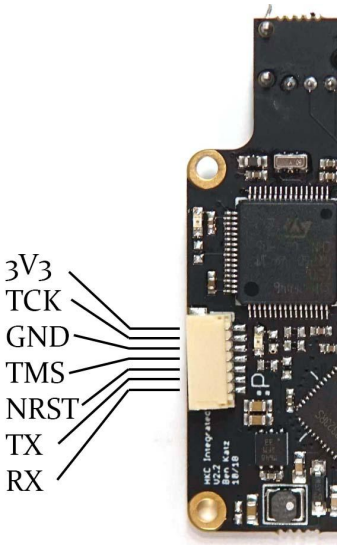


The part number for the CAN connector is M80-8420342. The mating connector is a M80-8990305. You can buy pre-crimped leads for the mating connector, which are part number M80-9110099.

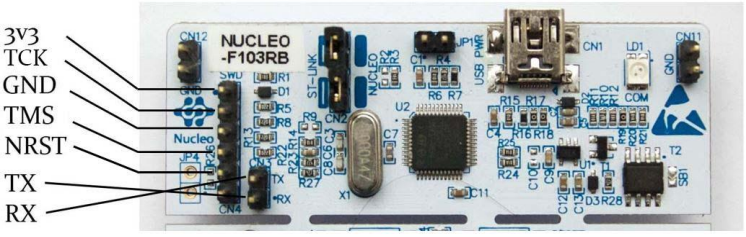
The pinouts of the CAN and XT30 connectors are shown in the image below. The XT30 pinout is standard and should match the markings molded into any mating XT30 connector.



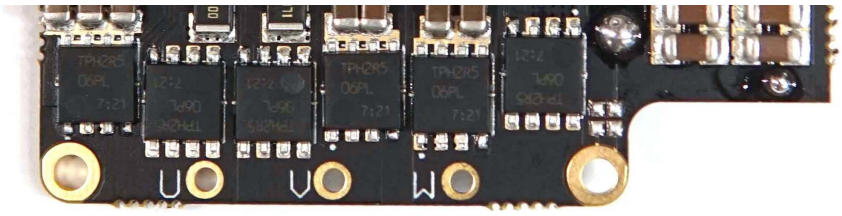
The programming/serial header is a JST SM07B-SRSS-TB connector. The mating connector is a SHR-07V-S-B, and the pre-crimped leads are ASSHSSH28K305. The pinout of the programming connector is shown in the image below. The pin ordering matches that of the ST-Link headers found on a Nucleo development board.



To connect with a Nucleo ST-Link, the pinout is as follows. Remember to swap TX and RX:



The motor terminals get soldered to the three terminals on the board labeled U,V, and W. The order the motor wires are soldered in doesn't matter, as during the calibration procedure the driver will swap them internally so that any motor will spin the same direction with a positive torque or speed command.



Power supply requirements

The motor drive will run on 10-35V DC. All internal components are rated for at least 50V, so 35V is not a strict upper bound, but you should leave a healthy threshold for safety. Peak current is 40A, if you set the torque limit to the maximum.

VERY IMPORTANT

If the motor needs to do significant negative work, it must be connected to a power supply capable of absorbing power, like a battery. The motor will back-feed the power supply (i.e. "regenerative braking"), converting the work to electricity. If the power supply is not designed for this, this energy will go into charging up the capacitors on the power supply and motor driver. This will cause a very large voltage spike on the DC bus, which can damage either your power supply or the motor driver.

You can also prevent this issue by using a shunt regulator (example) to absorb regen power.

AlSO VERY IMPORTANT

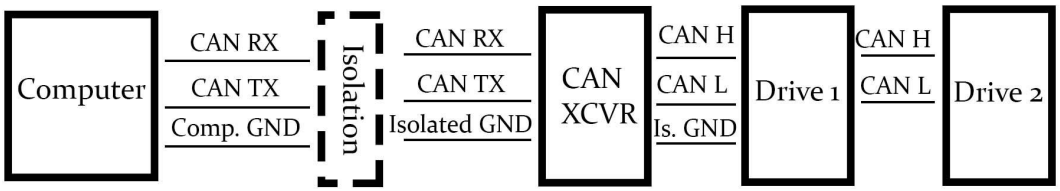
A pre-charge circuit should always be used to power on the motor drive! Plugging straight into a battery or power supply causes high currents as the capacitors on the board charge. The inductance in the wires going between the power supply and drive store significant energy at high currents, and this energy goes into charging up the capacitors on the board. This will cause a voltage spike when you plug the board in, which can be up to 2X the input voltage, and can cause components on the board to fail. Minimize the length of the wires between the drive and the power source as well, and twist them tightly. Recommended precharge resistance is 10-100 ohms. Make sure one LED turns on during precharge. If using a power supply for testing, you can just ramp up the voltage.

Wiring/ground loops

DC input wires should be roughly 14-18 AWG, depending on how much RMS power you expect. I recommend Teflon or Silicone insulated wires, as they can be run at very high temperatures. DC input wires should be as short as possible and tightly twisted together.

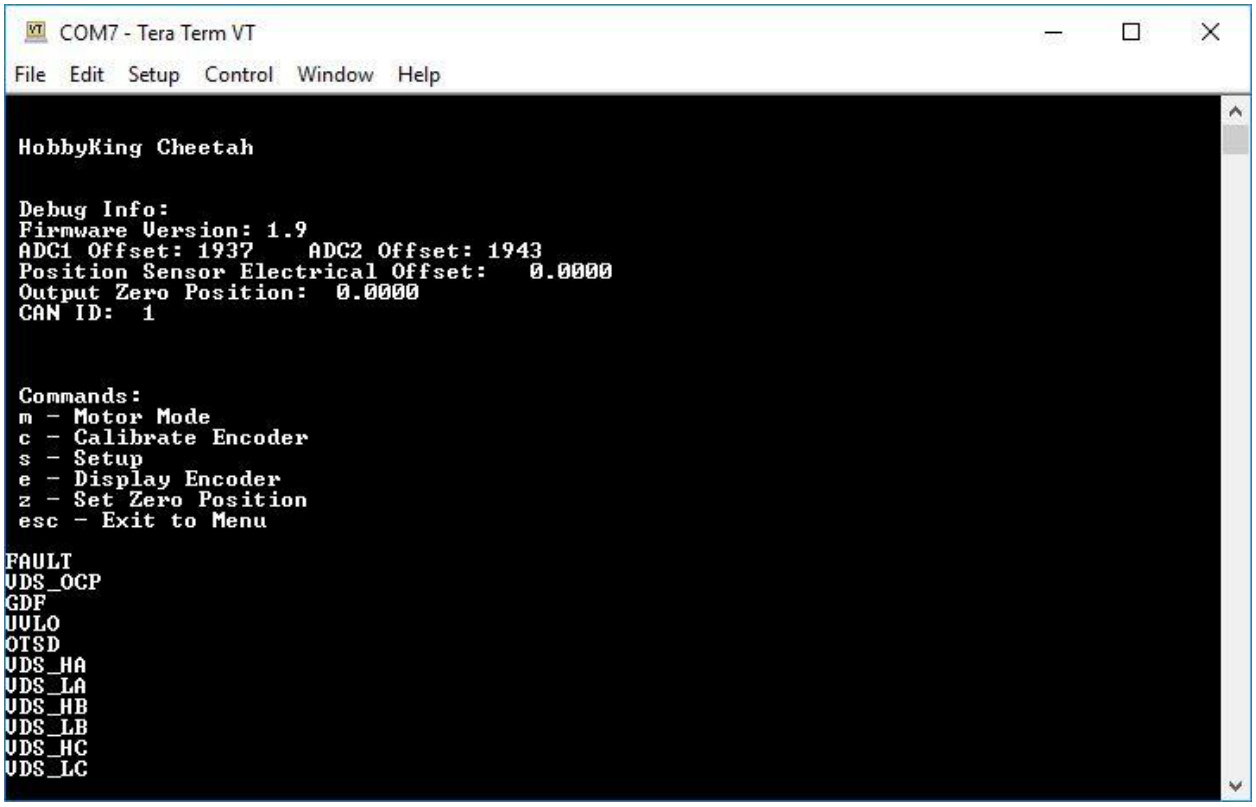
VERY IMPORTANT

On the CAN connector, DO NOT CONNECT GROUND PINS BETWEEN MOTOR DRIVES. This will cause a ground loop, and likely something will blow up at high currents. The power and logic grounds on the boards are connected, and CAN is differential, so no additional ground reference is needed. When daisy-chaining multiple motor drives, only connect the CAN-High and CAN-Low pins between drives. Next to the last drive in the chain, put a 120 Ohm resistor between CAN-H and CAN-L. Only use the ground pin for connecting to systems with a ground isolated from the power ground. I typically have isolation between the control computers and the motor drive, like the following diagram.



Configuration

The motor drives can be configured over a serial interface on a computer. The serial terminal should be configured for 921600 baud, 8 bits, 1 stop bit, no parity bits. Upon power-on, the driver will print some debug information to the terminal, and display a menu of options which can be navigated by typing in characters to the terminal. New versions will also print a list of faults once. If the faults print multiple times, something is wrong with the drive.



Enter a mode by typing the character displayed before each mode.

m - Motor Mode. In this mode, the motor will listen to position/velocity/torque commands over CAN, and respond with actual position, velocity, and current.

c - Calibrate Encoder. Make sure you run once the motor drive is attached to the motor. If you re-mount the motor drive, re-run the calibration. This process calibrates the position sensor on the rotor, and compensates for non-linearity in the position sensor output. This process cannot be interrupted by keyboard commands. If you need to interrupt it, you can do so by removing DC power.

s - Setup. This menu lets you change configuration settings.

Current control bandwidth - higher bandwidth means faster torque response, but causes some audible noise. I usually set to be slightly higher than the sample rate of the device controlling the motor, usually 1000 Hz.

CAN ID - the ID number on the CAN bus the motor will listen to. Each motor on the bus should have a different CAN ID.

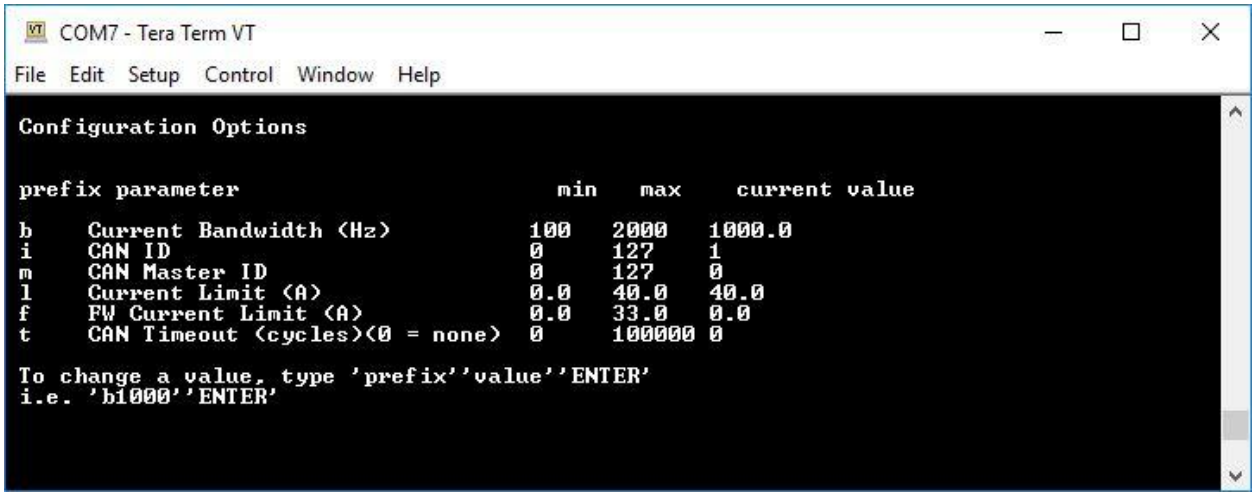
CAN Master - the ID number attached output CAN message. This should match the ID of your higher-level controller.

Current Limit - maximum peak phase current, in Amps

FW Current Limit - Maximum field weakening current. Defaults to zero. The motor drive can do closed-loop field weakening on Surface PM motors. Maximum should be less than your motor's continuous current rating, for safe continuous operation at high speeds. This feature will enable higher speed operation than would usually be possible for a given drive voltage. Achievable speed increase depends on the ratio of field weakening current times inductance to flux linkage.
CAN timeout - After this many loop cycles (40 kHz loop), the torque command will become 0, for safety (i.e. wire came unplugged, etc.)

e - Display Encoder. This mode prints out the mechanical angle of the motor, in radians, and raw encoder count.

z - Zero. Sets the mechanical zero position to the current encoder position



IMPORTANT
Power cycle the motor drive between changing settings and running the motor. Some internal settings are calculated from the user configurable parameters, and this only happens on power-on.

CAN protocol and example

In Motor Mode, the driver uses the following packet structures to send and receive data. The driver uses standard frame format (11 identifier bits) not extended.

An example program for communicating with a motor using a Nucleo can be found here: https://os.mbed.com/users/benkatz/code/CanMasterTest/

To run, it requires a CAN transceiver to be connected to pins PB_8 and PB_9.

The CAN bus runs at 1 MBaud. The motor drive receives 8 bytes of commands within the data field of the CAN packet, and sends back 6 bytes of data.

Command Packet Structure

The driver uses one packet to combine 5 commands. The commands are:

- 16 bit position command, scaled between P_MIN and P_MAX in CAN_COM.cpp
- 12 bit velocity command, scaled V_MIN and V_MAX in CAN_COM.cpp
- 12 bit Kp
- 12 bit Kd
- 12 bit Feed-Forward Current

On the motor drive, the commands will all be summed to achieve the final torque command.

In response, the motor drive will send back the following data:

- 8 bit motor ID
- 16 bit position, scaled between P_MIN and P_MAX in CAN_COM.cpp
- 12 bit velocity, between 0 and 4095, scaled V_MIN and V_MAX
- 12 bit current, between 0 and 4095, scaled to -40 and 40 Amps, corresponding to peak phase current.

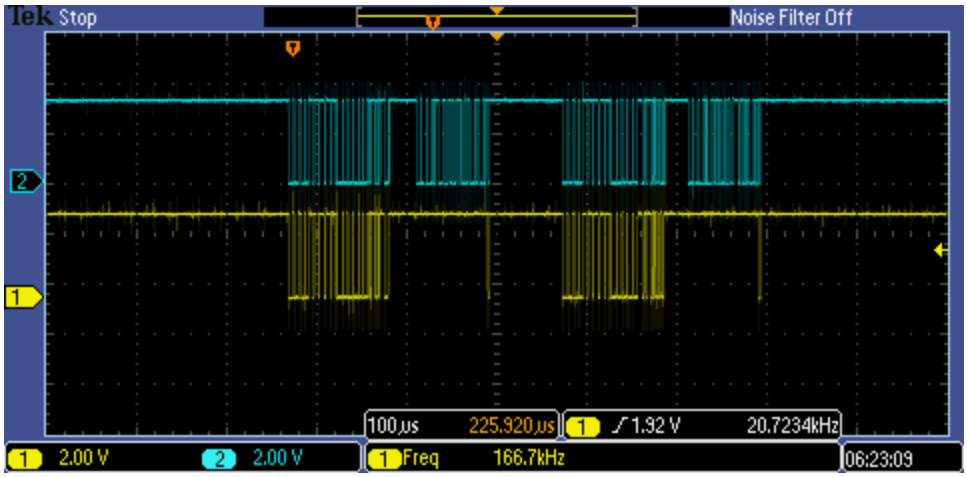
The 3 signals are packed into the 6 bytes in the following structure:
Byte 0: Motor ID
Byte 1: Position bits 15-8
Byte 2: Position bits 7-0
Byte 3: Velocity bits 11-4
Byte 4: Velocity bits 3-0: current bits 11-8
Byte 5: Current bits 7-0

Special Commands:

Enter Motor Mode
[0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFC]
Exit Motor Mode
[0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFD]
Zero Position Sensor - sets the mechanical position to zero.
[0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFE]

Timing

Each send and receive transaction takes about 220 µs. After receiving a message the motor module has a 15 µs delay before replying. Here's what send/receive from two motors looks like, with 300 µs delay between the starts of the two packets. Blue is the CAN TX pin on the microcontroller, yellow is the CAN RX pin:



USB-CAN Adapter, Python API

Board design and BOM for a USB to CAN board can be found here:
https://github.com/bgkatz/USBtoCAN/tree/master/eagle%20files

Firmware to interface with a Python API is here:
https://os.mbed.com/users/benkatz/code/CanMaster/

The Python API is in the same repository as the board design files. The python library will be able to send/receive commands at around 100 Hz, depending on your computer and OS:
https://github.com/bgkatz/USBtoCAN/tree/master/python%20library

If you want to run the motors entirely from a microcontroller, an easy-to-expand example is here.
https://os.mbed.com/users/benkatz/code/MotorModuleExample/

Firmware

Firmware is available here:
https://os.mbed.com/users/benkatz/code/HKC_MiniCheetah/

The exact variant and revision used on the newest Mini Cheetahs is here:
https://os.mbed.com/users/benkatz/code/HKC_MiniCheetah/rev/fe5056ac6740/

A stable revision which has been tested with the Python API and microcontroller example is here, in case future versions break features:
https://os.mbed.com/users/benkatz/code/Hobbyking_Cheetah/rev/59575833d16f/

New firmware based on ST's CubeMX setup code rather than MBED. Let me know (or open a git issue) if you have problems. https://github.com/bgkatz/motorcontrol

Updating the firmware on both the motor drives and USB-CAN board requires an STLink or Nucleo Dev Board in STLink mode. Both use the same JST-SH 7-pin connector.

Binary versions of the firmware are here for both the motor drive and the USB-CAN Adapter, if you don't want to deal with compiling:
https://drive.google.com/drive/folders/16Q1ty0vweEwSLUS5kKFIQCJXj0VAnlm3?usp=sharing

Eagle Schematic,board layout, Gerbers, and BOM are available here
https://github.com/bgkatz/3phase_integrated/tree/master/New%20Version/Cheetah%20Driver%20Integrated%20V4%20connector