CubeMars (TMotor) Control Method

This tutorial will walk through an alternative method to using the Dephy actuators, namely using the AK-series actuators provided by CubeMars (Associated with TMotor). These actuators come with a driver chip that manages the low level motor control, which can be commanded over a CAN bus or Serial Port using our open-source python library: TMotorCANControl. This library has been tested with the AK80-9, and should work with all AK-series TMotors. Sadly, it was named "TMotorCANControl" before the serial mode was available, and before we knew the subtle difference between TMotor and CubeMars.

In addition to the motor, some peripheral devices can be connected to the Raspberry Pi, such as an IMU, loadcell strain gauge amplifier, and external encoders. The sensor data from these can be accessed at the same time as controlling the AK80-9 motors from our opensourceleg python library (still currently under development).

As with the Dephy tutorial, the control examples below can be thought of as the basic building blocks of a higher level controller, serving as example code to realize your control strategy.

Helpful Links

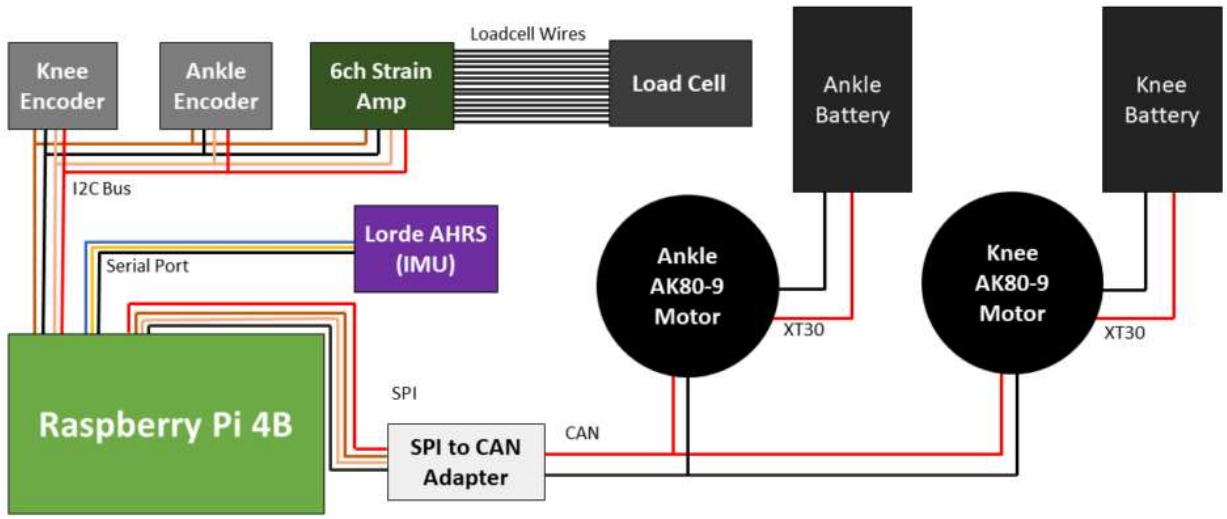
- TMotorCANControl GitHub Repository
  - This is the source code for the TMotorCANControl python package. You can also find demo scripts here.
- TMotorCANControl API documentation on ReadTheDocs
  - This is detailed documentation for each class, method, and variable in the API.
  - The most important parts are the TMotorManager_mit_can, TMotorManager_servo_can, and TMotorManager_servo_serial classes.
- TMotorCANControl PyPi (pip) Page
  - This is where the latest version of the software will be available for download and easy installation with pip, the python package manager.
- CubeMars AK80-series motor manual
  - This explains the usage of the CubeMars GUI and the communication protocols for each mode.

Overall OSL Electronics

Because the AK80-9 is just a motor (with onboard encoder), the peripheral electronics will need to be connected to the Raspberry Pi directly, rather than to the Dephy ActPack. You can see the diagram below for the AK80-9 in either control mode (more information about the different control modes of the AK80-9 Actuator is provided below, but in general we recommend using it over the CAN Bus in “Servo Mode.”

▶ Serial (Servo Mode only)

▼ CAN (Servo or MIT Mode)



Electronics Configuration in CAN mode

To use the OSL with the above configuration, some additional peripherals that are compatible with our opensourceleg library are:

1. AS5048B-TS_EK_AB 14 bit magnetic encoders.
2. 6ch Strain Amp From Dephy Inc.
3. Lord Microstrain AHRS (an IMU)

To set up and control 1 AK80-9 motor, you will need:

1. 1x CubeMars AK80-9 actuator.
2. 1x Raspberry Pi 4
3. 1x big PiCAN2 Hat or 1x CAN Bus plus RS485 Hat (The CAN bus plus RS485 hat is a bit sm cheaper, and more compact, but the PICAN2 hat is sized for a normal Raspberry pi)

4. 1x serial to USB converter of some sort, to configure the motor.
    - The RLink device from CubeMars will work, and allow you to test the motor over CAN from their GUI. Recommended if you expect to try out the different modes.
    - A cheaper FTDI converter will let you set it up, but you can't test it over CAN from the GUI
5. 1x power supply capable of supplying between 24V and 48V. This could be a battery, a desktop power supply, an AC-DC wall transformer, etc.
6. 1x 3by1 UART connectors and 1x 4by1 CAN connector for the driver chip. A set should come with the motor, but in case you need others, this kit would let you build your own.
7. 1x Male XT30 LiPo battery style connector for the driver chip. Again, one should come with the motor, but this kit would allow you to build your own.

AK80-9 Control Modes

The Motor can be controlled in any one of 3 possible modes, which will each require slightly different setup steps and will provide different options and limitations. The controller operates in either "MIT Mode," which is based on the MIT Mini Cheetah controller, or "Servo Mode," which exposes some more traditional motor control options. Additionally, in Servo Mode, the motor can be controller over the CAN bus or the Serial Port, while in MIT Mode it can only be controlled over the CAN Bus. Some specifications for each mode for the AK80-9 are provided below to highlight the differences. Essentially, CAN communication is faster, and serial communication provides more feedback data. The MIT Mode has a lower current limit but on-board impedance controller, and the Servo mode has a higher current limit but no on-board impedance controller.

In most cases, we recommend using Servo Mode over the CAN port, due to the high maximum current and update frequency. If more data is needed, Servo Mode over the Serial port could be used, and if an on-board impedance controller is required, then the MIT mini cheetah controller provided in MIT mode will work well, but at lower peak output torques.

Specification	MIT Mode (Over CAN Bus)	Servo Mode (Over CAN Bus)	Servo Mode (Over Serial Port)
Max Current (or Torque) Command	18 Nm (about 16.6A q-axis)	60 A (q-axis)	60 A (q-axis)
Recommended peak current	24 A	24 A	24 A
Communication Rate	1 MBps	1 MBPs	962100 $B_{\text{ł}}$

Maximum update frequency	1000 Hz	1000Hz	50Hz
Needed Peripherals	CAN to USB converter	CAN to USB converter	Serial to USB converter
Current Control	Yes	Yes	Yes
Duty Cycle (Voltage) Control	No	Yes	Yes
On Board Impedance Control	Yes (MIT – Mini Cheetah Controller)	No	No
On Board Position Control with Trapezoidal Velocity Planning	No	Yes	Yes
Velocity Control	Yes (Via impedance controller)	Yes (Via PID controller)	Yes (Via PID controller)
Reports motor Position, Velocity, q-axis Current, Temperature, and Error Code	Yes	Yes	Yes
Reports q-axis voltage, d-axis current & voltage, and input current and voltage, and more detailed error code	No	No	Yes

Table of Parameters for Each control Mode

Calibrating and Configuring the Motor

Before the motor can be used in operation, it must first be set up over the serial port. For video tutorials walking through the process, see the videos Yoyo Liu's YouTube channel. Select the mode you plan to use below to see the configuration tutorial.

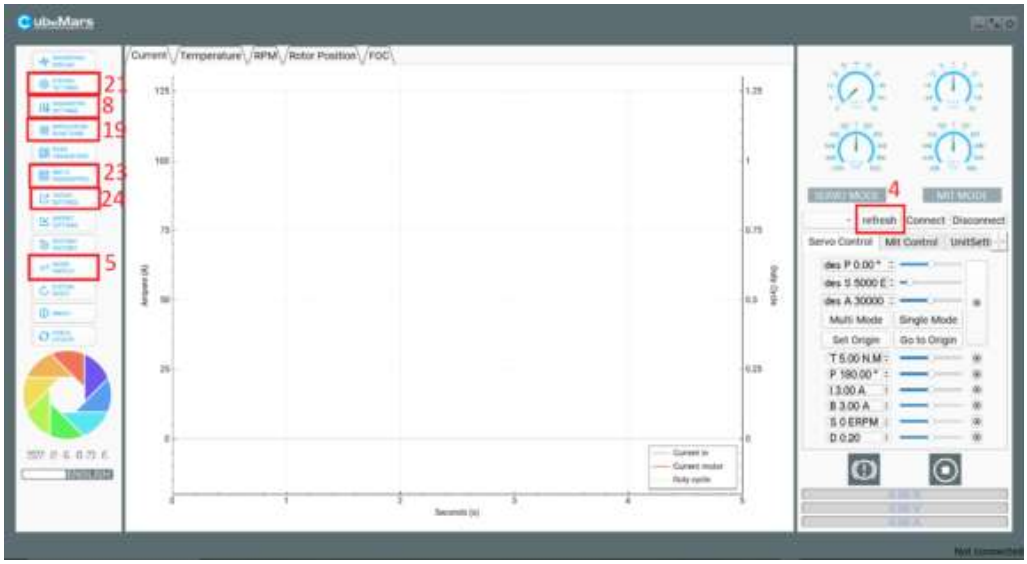
▶ MIT Mode (CAN Only)

▼ Servo Mode (CAN or Serial)

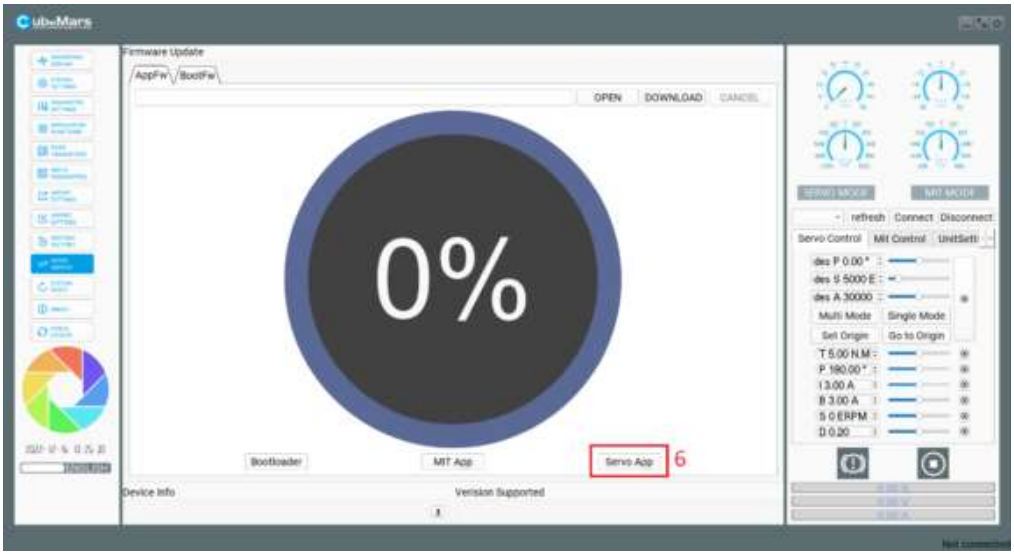
1. On a Windows computer, download the R-Link program at the bottom of this page on the TM website. (It's called the "Upper Computer"). The program will work even if you are using a basic

FTDI connector instead of the Rubik Link device.

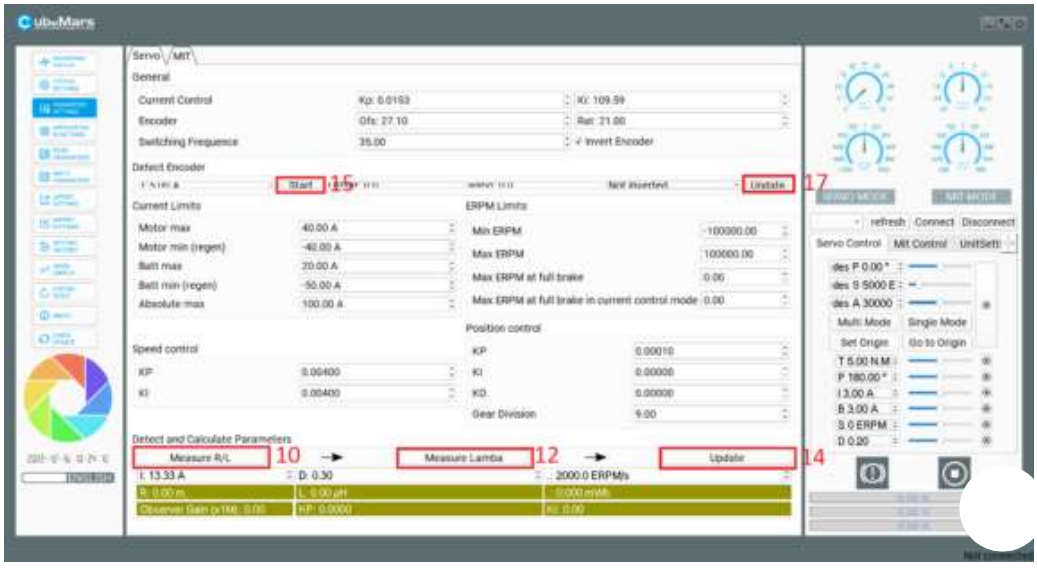
2. With the motor powered off, connect the USB to serial device to the UART port on the driver chip and to your computer.
3. Open the R-Link program. You can switch the language to English at the bottom left side of the screen.
4. Press the **"refresh"** button. You should see your serial port appear in the menu to the left of this button (ie, COM6).
5. Press the **"Mode Switch"** button to reveal the Mode Switching Menu. The motor comes pre-configured for MIT mode, so we will need to switch to the Servo Mode firmware.
6. Press the **"Servo App"** button on the left sidebar.



Locations of the menu items, labeled by tutorial step.



Mode Switch Button Location



Location of steps to update parameters. Use these parameters for your motor as a baseline.

7. Wait for a few seconds for it to switch. A popup menu should appear saying that you have switched to Servo Mode. If this hangs or does not ever appear, contact CubeMars to verify that your motor came with the servo mode enabled firmware.
8. Press the **"Parameter Settings"** button on the left sidebar. We will need to calibrate the motor. This will open up the calibration and parameter setting page.
9. Make sure the Motor is free to rotate.
10. Press the **"Measure R/L"** button, and hit okay on the popup that appears.
11. Wait for it to make some ticking sounds. When it's done a popup will appear–press okay.
12. Press the **"Measure Lambda"** button, and hit okay on the popup that appears.
13. Wait for it to spin up to a constant velocity for a few seconds. When it's done, a popup will appear–press okay.
14. Press the **"Update"** button (in the lower right). This will write the measured parameters to the motor.
15. In the "Detect Encoder" area, press the "Start" button. On smaller screens, this section may not render fully, but the buttons will be in the same place as shown.
16. Wait for it to spin around. There is a high-pitched whine, which is annoying, but it will be over soon!
17. Press **"Update"** button (in the upper right).
18. Verify the gains, limits, and other parameters on this screen are as you would like. (The values shown are what we use, but you can always come back to update them). Note that the values pertaining to the current control, encoder, and the measurements at the bottom might be different from ours, and don't need to change.
19. Press the **"Application Functions"** button on the left sidebar.
20. Set the controller ID to your desired number, set the Baud rate to "BAUD_RATE_1M". Optionally, check the "Send status over CAN" button and set a rate, and set a timeout time in ms (should be longer than the update delay you plan to use with the motor).
21. Press the **"System Settings"** button on the left sidebar.
22. Verify that the settings match the defaults for the AK80-9, shown below.
23. Press the **"Write Parameters"** button on the left sidebar. If successful, a green "Parameter write OK" message should appear at the very bottom right of the window.
24. Press the **"Export Settings"** button on the left sidebar to save the settings you have configured for later re-use. It will prompt you to save a MCParams and an AppParams file.
25. You can now power off the motor, or test it in the **"Servo Control"** tab on the right side panel of the GUI.
26. Later, you can recover these settings by pressing **"Import Settings"** and selecting your saved files.

Installing the TMotorCANControl Library

1. Prepare a functioning Raspberry Pi 4 following the instructions here. Preparing the Pi in this way will make connecting over SSH much easier.
2. Please ensure that you can connect to the Pi over SSH.
3. Once connected to the Pi, type **"pip install TMotorCANControl"** to install our library that will be used to communicate with the motor. If this step fails for some reason, you can see the repository on gitub here and can fork it or download the code directly.
4. The above command should also install the NeuroLocoMiddleware library, the python-can library, and the pyserial library. If it doesn't, then type **"pip install NeuroLocoMiddleware"**, **"pip install python-can"**, and **"pip install pyserial"** to get the dependencies.
5. The example scripts used in this tutorial can be found in the github repository in the demos folder, or can be downloaded here: TMotorCANControlDemos

Configuring the Raspberry Pi for Communication

▶ Serial Communication (Servo Mode Only)

▼ CAN Communication (Servo Mode or MIT Mode)

1. The PiCAN hat comes with a 120 Ohm termination resistor that can be connected by shorting the leads labeled **"JP3"**. Solder a 2-pin header to these leads and connect a jumper to enable the resistor. If using the CAN Bus plus RS485 Hat, this resistor is already wired in.
2. With the Pi powered off, connect the PiCAN Hat to the Raspberry Pi's GIPO headers.
3. Follow the instructions on this page to set up the Raspberry Pi to be able to connect to the PiCAN hat, or on this page for the CAN Bus plus RS485 hat. Note that the parts where they test the connection are geared toward connecting to a car's CAN bus, so these exact codes are not relevant for us. If you run into problems, check their troubleshooting guide.
4. Now, connect the CAN high and CAN low lines from the driver board on the motor to the corresponding screw terminals on the CAN hat. Note that in the user manual, the order of pins in the CAN port is given as Low, High, High, Low. Some motors have a different order, and if it does it will be labeled on the chassis of the motor, and you should use that ordering. The ground

terminal on the CAN hat should be connected either to the ground line of the UART port on the driver chip, or to the ground of your power supply setup, to ensure a common ground.

5. To test the CAN connection, navigate to the folder where you saved the test scripts, and enter the “servo_can” subfolder or “mit_can” subfolder, depending on which control mode you are using.
6. Type the command **“nano check_motor_connection_mit_can.py”** or **“nano check_motor_connection_servo_can.py”**, depending on which mode you are using. This will open a text editor for you to edit the script.
7. Change the variables **“ID”** and **“Type”** to have the values that match the CAN ID of your motor and the type of motor. You will need to do this for every example script, unless you happen to be using an AK80-9 motor with CAN ID 1.
8. To exit nano, press **ctrl+X**. Press **y** and then **ENTER** to save your work.
9. Power the motor on.
10. Type **“python3 check_motor_connection_mit_can.py”** or **“python3 check_motor_connection_servo_can.py”** to run the test program. If you see it output “motor is successfully connected!” then the setup is complete and you can begin writing controllers! You can see the example scripts below as a jumping-off point, or dive into the library documentation if you’d like.
11. If the connection was not sucessful:
    1. Double check that the motor is on and that you see a blue light.
    2. Double check that the CAN ID and motor type variables in **“check_motor_connection.py”** are correct.
    3. Double check the wiring of the circuit.
    4. Double check the termination resistor is connection to JP3 onthe PiCAN hat.
    5. Double check that the PiCAN hat is set up correctly by using “feedback mode” as described here.
    6. Double check that the python modules are all installed.
    7. If this still doesn’t work, then please log an issue on the github repository and we will see what we can do to help.

TMotorCANControl Examples

The following examples and demos involve reading data from actuators and commanding actuators with different control laws. You can change and experiment with these scripts as you see fit–they are m

be a jumping off point for your future endeavors. You can also check out the github readme for more information about how the library works. In order to run each of the examples, follow the steps below:

1. If you haven't downloaded the demos yet, then they can be found in the github repository in the demos folder, or can be downloaded here: TMotorCANControlDemos
2. Navigate to the folder where you installed the demos. Enter the subfolder relating to your chosen communication and control mode (MIT or Servo, CAN or Serial).
3. Type **"dir"** to view the scripts in this folder, or consult the lists below to identify which script you would like to run.
4. For each of the scripts below, you may need to edit the parameters to match your motor's ID, port, type, etc. If you are using an AK80-9 with the default ID, then you should be good to go. To edit these parameters, follow the bulleted steps below:
   - Type: **"nano <script_name>.py"** into the console to view the script using the default text editor of the Raspberry Pi. Of course, if you prefer a different text editor feel free to use that!
   - When you open the script to change this information, you can also take a look at the rest of the code to see how it works.
   - To exit nano, press **ctrl+X**. Press **y** and then **ENTER** to save your work.
5. Turn on the power to the motor.
6. To run the script, type **"python3 <script_name>.py"**.
7. Finally, to exit the program simply press **ctrl+C**. This will cease the program operation and safely send the power down command to the motor.

▶ mit_can

▶ servo_serial

▼ servo_can

1. **check_motor_connection_servo_can.py**: As described in the setup tutorial above, this script will tell you if the motor is properly connected or not. If it is not, follow the above debugging steps.
2. **demo_idle_servo_can.py:** This script will run the motor in read-only mode, printing the motor's state information to the console. Note that the temperature of the chip may be unavailable and simply read 0.
3. **demo_position_step_servo_can.py:** This script will send a constant position command to the motor using an impedance controller. You can change the position value as well as the impedance gains to see what different controllers feel like.
4. **demo_current_step_servo_can.py:** This script will send a constant current command to the motor. You can play with this value to change the acceleration induced on the motor. Just be

careful that nothing gets in the motor's way, as it will spin at a high speed.

5. **demo_duty_step_servo_can.py:** This script will send a constant duty cycle command to the motor. You can play with this value to change the speed of the motor.
6. **demo_velocity_servo_can.py:** This script will send a constant speed command to the motor. Speed control is unlikely to be very useful for controlling a prosthetic leg, but the functionality exists in the motor driver chip, so it is an option available to you.
7. **demo_position_tracking_servo_can.py:** This script is an example of how to track a desired position trajectory, in this case a simple sinusoidal path. The motor will oscillate about its desired path.
8. **demo_PD_duty_servo_can.py:** This script will show an example of a PD controller that uses the duty cycle of the motor to drive it to a desired position.
9. **demo_PD_current_servo_can.py:** This script will show an example of a PD controller that uses the current of the motor to drive it to a desired position. This is essentially impedance control, since current is proportional to torque.
10. **demo_current_chirp_servo_can.py:** This script will send an oscillating torque command that will cause the motor to play a chirping sound.
11. **demo_two_DOF_servo_can.py:** This script will set one motor to the angle of the other, so you can turn it remotely! You can use this structure as a model for writing code to control multiple motors in the same control loop.

Implementation Details

▼ General

1. **Units:** The quantities reported by the TMotorCANControl API are in SI units, namely position in rad, velocity in rad/s, acceleration in rad/s/s, current in Amps, and torque in Nm/A. We provide functions dealing both with "output" angle and torque and "motor-side" angle and torque. In this case, the output quantity refers to its value after the gear reduction, and the motor-side quantity refers to its value before the gear reduction. The current given is an estimate for the q-axis current (see note 4 below).

▼ CAN Bus

1. **Python Version:** The python-can library used to mange the CAN bus requires python to be at least version 3.7. This dependency is fundamental to the operation of the TMotorCANControl API, and as such to use TMotorCANControl you must upgrade to at least python 3.7.
2. **Communication Rate:** Our fastest successful communication rate with one motor was 2000Hz. However, at such a fast communication rate, the program was running almost constantly, so we recommend slower communication rates of 1000Hz or less to guarantee stability, especially when working with additional sensors and actuators in the control loop.

▼ Serial Port

1. **Python Version:** The pyserial library used to mange the Serial Bus works with both Python 2 and Python 3!
2. **Operating System:** Because it doesn't rely on a particular Linux CAN driver, you could actually use the TMotorCANControl library to drive a motor from a windows laptop, or any other serial interface compatible with pyserial.
3. **Communication Rate:** Our library can handle communication of up to 1000Hz without difficulty. However, at such a fast communication rate, the TMotor started grouping packets together, and the effective update frequency was more like 100Hz. To avoid this issue, in Serial mode it is recommended to use only 50Hz until the issue is resolved.

▼ Servo Mode

1. **Peak Torque**: In Servo mode, with a 60A current limit, the peak torque could be quite high. The highest we have measured is 34 Nm, but this was at only 30A, so we expect it can get higher if you are willing to push the motor more. The higher peak current you use, the less time you can use it for before it overheats of course, so beware.

▼ MIT Mode

1. **Zeroing the Motor Angle:** The TMotor controller for the AK-series TMotors offers built-in functionality to "zero" the position of the motor. The motor will remember this zero position as long as it has power. This can be done by calling the "set_zero_position()" function in the TMotorCANControl API. After sending this command, the motor will be unresponsive for abo second, as it collects encoder measurements to mark its new zero position. Please be awa

this fact when calling this function, as you may receive a runtime warning if you do not wait for a long enough period before sending new commands.

2. **Peak Torque:** In MIT Mode, the peak steady-state torque we have measured with the AK80-9 is 14Nm, significantly less than the Dephy Act Packs are capable of. For applications where higher torque is required, we recommend using one of the higher-torque TMotors, such as the AK80-64.
3. **Q-Axis Current and Torque Constant Representation: I**n the TMotorCANControl API, the current given is the q-axis current. This refers to current seen in a DC motor producing the same amount of torque as our BLDC motor, and allows us to consider one current value rather than 3 (one per phase). The Dephy Act Packs report q-axis current, and it is generally useful for writing higher level-controllers. However, rather than directly reporting q-axis current, the TMotor controller for all the AK-series TMotors reports an estimated torque value, which is higher than the torques that we have been able to measure on a benchtop test. We have estimated the q-axis torque constant of the AK80-9 to be 0.115Nm/A, and have adjusted the current value reported by the TMotorCANControl API to report the q-axis current (in Amps) that would give the correct torque. Note that on the TMotor website, the AK80-9's torque constant is listed at 0.091Nm/A, which is a line-to-line torque constant that corresponds with q-axis torque constant of between 0.11 and 0.12 Nm/A, right around the range we have measured. While we have not yet been able to test the other TMotors, we expect they will function similarly, and have provided estimated torque constants in the API implementation. If you measure something different, please let us know!
4. **Operation Near Limits:** In MIT Mode, all of the AK-Series TMotors have position limits of ±12.5rad, imposed by the encoder resolution. Additionally, each motor has a different velocity and current/torque limit. For the best operation, we recommend not operating too close to these limits. When any of these values moves past the limit, the TMotor will simply wrap the value back to the opposite bound (so if moving to an angle more positive than 12.5rad, it will wrap to -12.5rad, and vice versa). This could be confusing, so the TMotorCANControl API takes care of compensating for this wraparound effect, allowing you to measure angles beyond 12.5rad and velocities above their respective limits. Note that you still cannot command a position or velocity outside of this range, as the driver will not know how to accept the value. Our API will simply smooth the data if you are operating near the limits, so that the motor's state won't appear to rapidly oscillate. Also, the current is squashed to be limited to the maximum/minimum value, because it can change so quickly that if we tried to expand its range things could spiral out of control. So, we still recommend not trying to instantly switch any value from its upper limit to its lower limit, but the API should be able to handle it if you need to do so.
5. **Torque Fluctuation:** In MIT Mode, we have also observed a torque dip of about 20% that occurs twice per rotation through one magnetic angle (42 times per rotation of the rotor, or 378 times per rotation of the output shaft). Perhaps our mootr was damaged in testing, or improperly callibrated when this occurred.

This page was last updated on 16 Dec 2022.