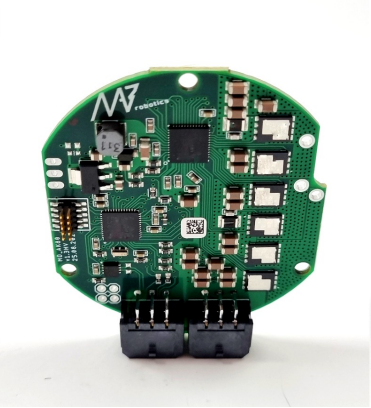
MD x CANdle Manual

Contents


Quick Start Guide

Before first use

Skip to main content

1. Always stay cautious when dealing with actuators. Even though they don't seem like it, they may severely hurt you when unintentional movement occurs. It's recommended to fix the actuator to the workbench.
2. Get accustomed to the safety limits section of this document. While developing your application be sure to keep the limits low, and only if you are sure you know what you're doing - increase the limits.
3. We recommend using power supply sources that can work in two quadrants - meaning they can supply and dissipate some of the energy produced by the motor in case it works as a generator. Old trafo-based power supplies usually block current coming into the power supply, causing overvoltage events on the MD80s. The best choice is to use LiPo batteries or at least SMPS power supplies.

Video tutorials

Check out our video tutorials if you prefer this way of presentation:

1. Guide on how to set up a new MD series motor controller to work with a motor of your choice: MD x CANdle - Brushless Motor Setup Guide
2. Quick startup guide - watch when your motor is already configured and you'd like to try the examples: MD x CANdle - Getting Started Tutorial
3. MDtool guide - a short introduction to MDtool and its basic commands (at this point the MDtool has evolved, however, the main principles are the same) MD x CANdle - MDtool
4. Motion modes guide - an introduction to motion modes available on MD series motor controller MD x CANdle - motion modes
5. ROS/ROS2 startup guide - an introduction to ROS/ROS2 drivers for MD series motor controller MD x CANdle - ROS/ROS2 startup guide

Configuring the MD series controller for a new motor

MDtool can be used to set up a new motor to work with the MD series motor controller controller. This approach simplifies the configuration process so that the end user can reconfigure the MD series controller driver to work with almost any brushless motor.

Warning

These steps are universal between the controller hardware versions, however, be sure to always check the maximum ratings before applying voltage to the controller.

- First, make sure the MD controller can work with your motor. A vast majority of hobby motors will be suitable, although too big motors in terms of power and gimbal motors (with high phase resistance) might not work as expected. Be sure to contact us before you proceed with a gimbal or high-power motor (over 500W peak power).
- Place the diametrically magnetized magnet on the motor shaft and mount the MD controller firmly, centered above the magnet.

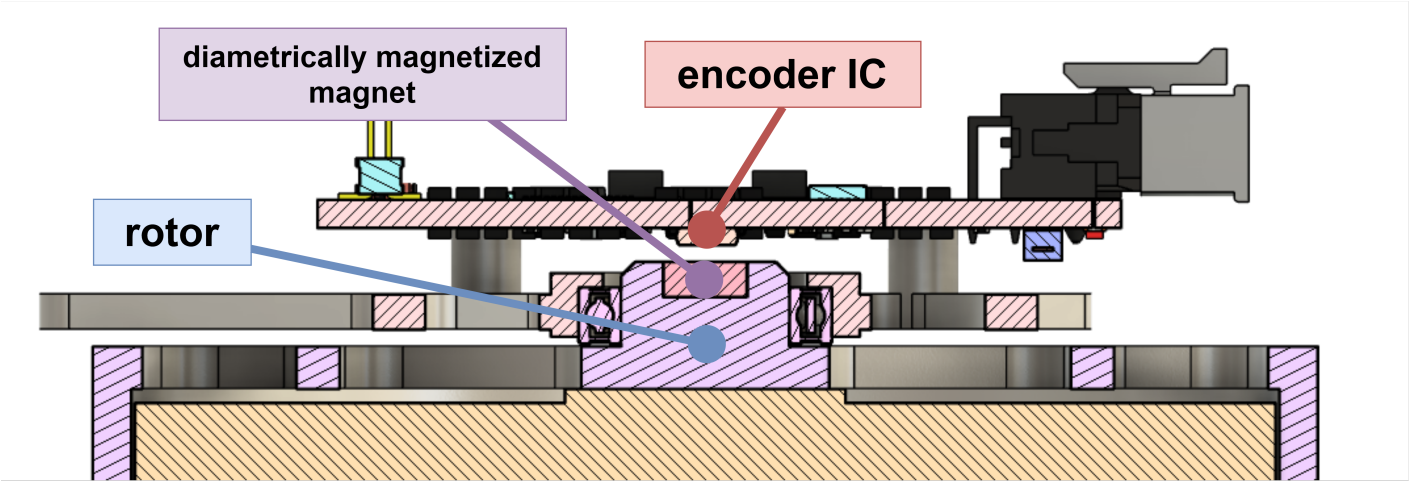


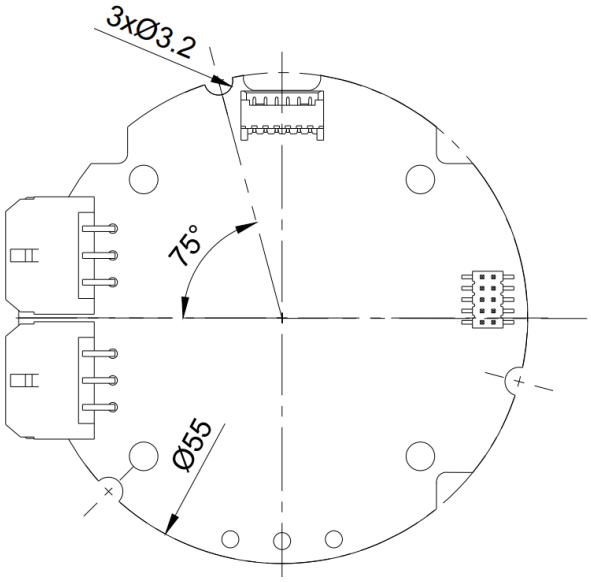
Fig. 1 Magnet - encoder placement cross section

The optimal height between the magnet and the encoder IC is 1mm. The magnet and the encoder must be on the same rotation axis.

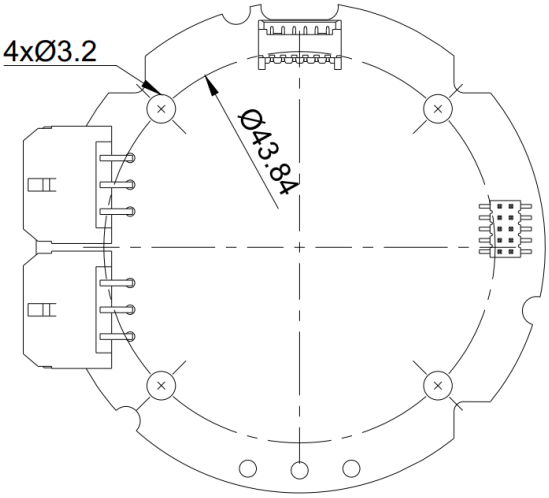
MD80 V3.0 is equipped with seven mounting holes. Please refer to the technical drawing below to find out the hole dimensions and their placement. The 3D model of the driver can be found in the downloads section.

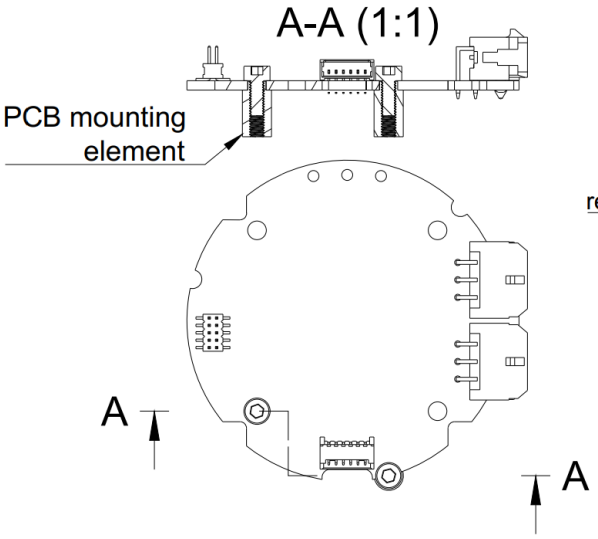
Skip to main content

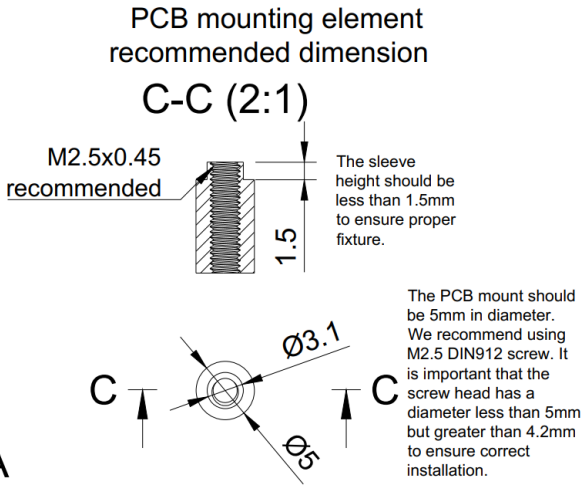
Three holes with a diameter of 3.2mm placed on a circle with a diameter of 55mm.



Four holes with a diameter of 3.2mm placed on a circle with a diameter of 43.84mm.



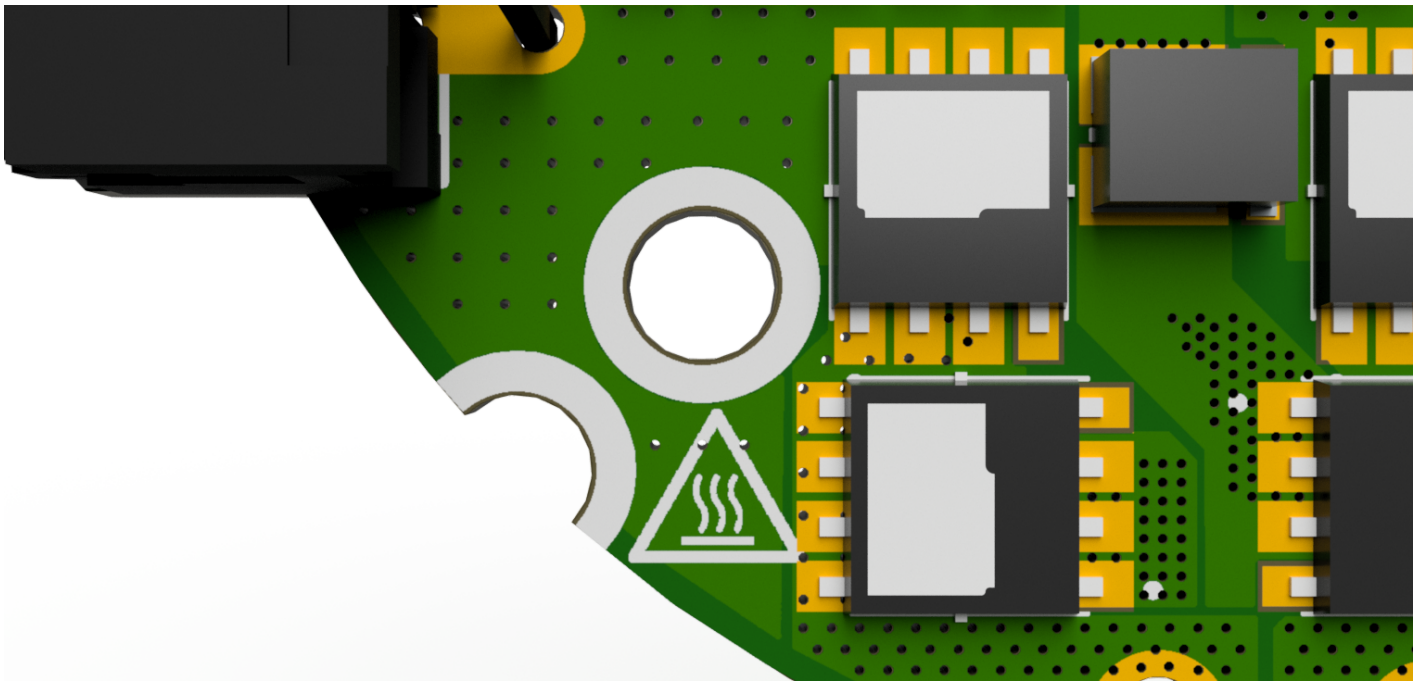




Warning

Always make sure the head of the screw is inside the white hole outline. Otherwise, it may cause permanent damage to the controller when a short circuit occurs between the head screw and any of the copper planes. Using M2.5 DIN912 stainless socket screws is recommended.

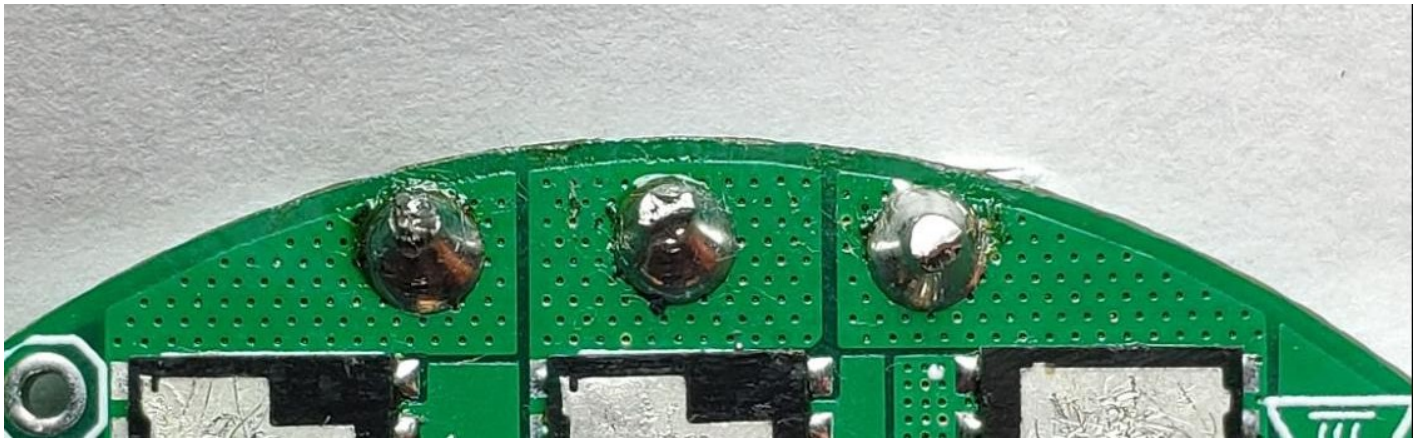
Skip to main content



- Solder the motor wires to the PCB making sure all the individual motor wires within a single phase are connected (in case the motor is wound with more than one wire on each phase). It is possible to solder the motor from the bottom, however, soldering the wires on the top is also acceptable. Make sure that the phase wires are connected only to their respective pads.

Warning

The order of the cables does not matter (does not change the rotation direction) as long as the order is not changed after the calibration. Each modification in wire order should be followed by a full motor calibration.



Skip to main content

Hint

Sometimes soldering may be difficult due to the high-temperature enamel on the copper wires. In that case, try to apply solder at high temperatures using flux until the solder sticks to all wires nicely.

- Connect the power supply to the controller through the CANdle device as specified in this section. When powered the controller should blink shortly with a green LED once a second. If the red LED is fully on there are some errors that should be cleared after the calibration.

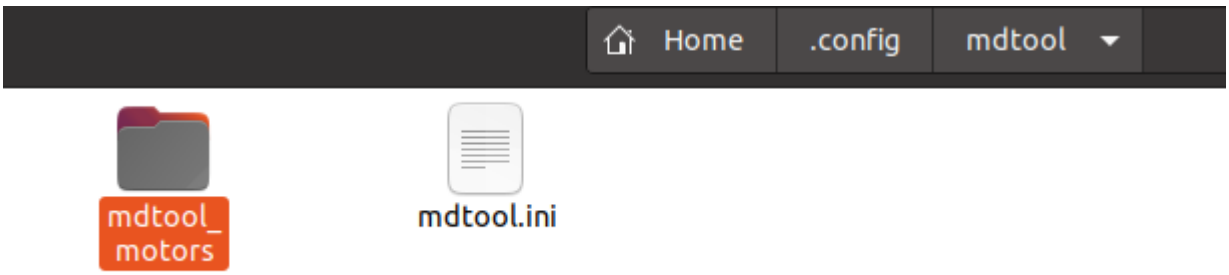
Warning

Always make sure that the polarity of the power supply is correct. MD controllers do not have reverse polarity protection so connecting the power supply in reverse polarity will cause permanent damage to the controller.

- Connect CANdle to the PC using a USB type-C cable.
- Ensure you've got the latest MDtool. For the MDtool installation guide refer to the MDtool section.
- Please upgrade the setup software if any of your devices (MD or CANdle device) is older than the one from "latest" row in the releases table.
- Once installed and run the MDtool will create an MDtool directory in **~/.config** directory.

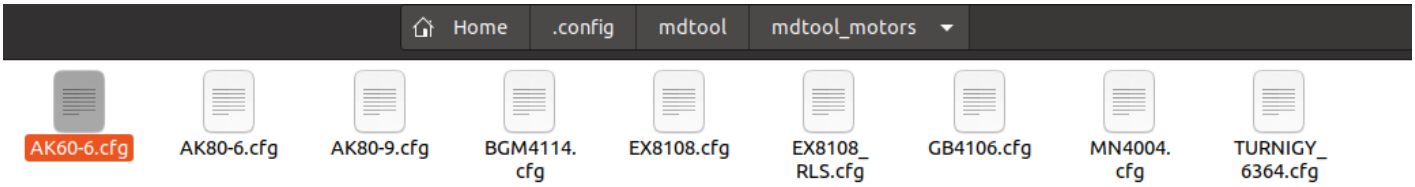
Hint

Press Ctrl+H to view the hidden folders (starting with a dot)



There, you will find a mdtool.ini file which contains factory settings and **should not** be modified, and a

Skip to main content



Feel free to add a new *.cfg file for your custom motors in there. Use the already existing files as a reference, especially the AK60-6.cfg which contains some additional comments.

- Check if the MD controller can be discovered properly using the `mdtool ping all` command

```
MAB@tutorial:~$ mdtool ping all
[MDTOOL] MDtool version: v1.2
[CANDLE] CANdle library version: v3.0
[CANDLE] Device firmware version: v2.0
[CANDLE] CANdle at /dev/ttyACM0, ID: 0xac12814dfe38f7d8 ready (USB)
[CANDLE] Pinging drives at 1M CAN speed...
Found drives.
1: ID = 100 (0x64)
[CANDLE] Pinging drives at 2M CAN speed...
No drives found.
[CANDLE] Pinging drives at 5M CAN speed...
No drives found.
[CANDLE] Pinging drives at 8M CAN speed...
No drives found.
```

- To setup the MD series controller simply call `mdtool setup motor <ID> <*.cfg>` where the ID is the ID that shows up after the `mdtool ping` command is called, and the *.cfg is one of the files existing in the mdtool_motors directory (press tab to list available config files). If anything fails during the process be sure to check your setup and try again.

```
MAB@tutorial:~/mdtool/md_tools$ mdtool setup motor 100 EX8108.cfg
[MDTOOL] MDtool version: v1.2.1
[CANDLE] CANdle library version: v3.1
[CANDLE] Device firmware version: v2.0
[CANDLE] CANdle at /dev/ttyACM0, ID: 0xac12800687432d34 ready (USB)
[CANDLE] Added MD80 with ID: 100  [OK]
[CANDLE] Saving in flash successful at ID: 100  [OK]
```

- Do not worry if at this point there are many errors after calling `mdtool setup info <ID>` command (like below). They should be cleared after a successful calibration.

Skip to main content

```
MAB@tutorial:~/mdtool/md_tools$ mdtool setup info 100
[MDTOOL] MDtool version: v1.2.1
[CANDLE] CANdle library version: v3.1
[CANDLE] Device firmware version: v2.0
[CANDLE] CANdle at /dev/ttyACM0, ID: 0xac12800687432d34 ready (USB)
[CANDLE] Added MD80 with ID: 100  [OK]
Drive 100:
- actuator name: EX8108
- CAN speed: 1 M
- CAN termination resistor: disabled
- gear ratio: 1.000
- firmware version: V2.1
- hardware version: HV13
- build date: 30.1.2023
- commit hash: f7d77669
- max current: 10.0 A
- bridge type: 1
- pole pairs: 21
- KV rating: 0 rpm/V
- motor shutdown temperature: 80 *C
- motor torque constant: 0.0910 Nm/A
- motor stiction: 0.000 Nm
- motor friction: 0.000 Nm
- d-axis resistance: 0.000 Ohm
- d-axis inductance: 0.000000 H
- torque bandwidth: 500 Hz
- CAN watchdog: 200 ms
- main encoder last check error stddev: 0.000000 rad
- main encoder last check min error 0.000000 rad
- main encoder last check max error: 0.000000 rad
- output encoder: no
- position: 0.19 rad
- velocity: -0.30 rad/s
- torque: 0.00 Nm
- MOSFET temperature: 0.00 *C
- motor temperature: 0.00 *C

***** ERRORS *****
- main encoder error:   0x4     (ERROR_EMPTY_LUT, )
- calibration error:    0x7     (ERROR_OFFSET_CAL, ERROR_RESISTANCE_IDENT, ERROR_INDUCTANCE_IDENT, )
- bridge error:         0x0     (ALL OK)
- hardware error:       0x0     (ALL OK)
- communication error:  0x0     (ALL OK)
```

- When succeeded, the motor is set up correctly and now's the time to calibrate it using `mdtool setup calibration <ID>`. Please follow the calibration guidelines for more information on the calibration process itself.
- After the calibration the motor should be ready to use - the best way to find out everything was completed without errors is to check the MD series motor controller info using the `mdtool setup info <ID>`. This command lists all the important parameters of the actuator. Errors are shown in red on the bottom if anything has failed during the process. If there are still errors after the calibration be sure to check out the error's description and try the recommended action to clear it status.

Correct after-calibration mdtool setup info command output may look like this:

```
MAB@tutorial:~/mdtool/md_tools/build/packages$ mdtool setup info 100
[MDTOOL] MDtool version: v1.2.1
[CANDLE] CANdle library version: v3.1
[CANDLE] Device firmware version: v2.0
[CANDLE] CANdle at /dev/ttyACM0, ID: 0xac12800687432d34 ready (USB)
[CANDLE] Added MD80 with ID: 100  [OK]
Drive 100:
- actuator name: EX8108
- CAN speed: 1 M
- CAN termination resistor: disabled
- gear ratio: 1.000
- firmware version: V2.1
- hardware version: HW20
- build date: 30.1.2023
- commit hash: 214328da
- max current: 10.0 A
- bridge type: 3
- pole pairs: 21
- KV rating: 0 rpm/V
- motor shutdown temperature: 80 *C
- motor torque constant: 0.0910 Nm/A
- motor stiction: 0.000 Nm
- motor friction: 0.000 Nm
- d-axis resistance: 0.207 Ohm
- d-axis inductance: 0.000035 H
- torque bandwidth: 500 Hz
- CAN watchdog: 200 ms
- main encoder last check error stddev: 0.000000 rad
- main encoder last check min error 0.000000 rad
- main encoder last check max error: 0.000000 rad
- output encoder: no
- position: -0.00 rad
- velocity: -0.37 rad/s
- torque: 0.00 Nm
- MOSFET temperature: 30.58 *C
- motor temperature: 0.00 *C

***** ERRORS *****
- main encoder error:   0x0     (ALL OK)
- calibration error:    0x0     (ALL OK)
- bridge error:         0x0     (ALL OK)
- hardware error:       0x0     (ALL OK)
- communication error:  0x0     (ALL OK)
```

Your actuator should be now ready to go! To make it move you can try the `mdtool test move <ID> <pos>` command, or try the CANdle lib C++ or Python examples.

Ecosystem overview

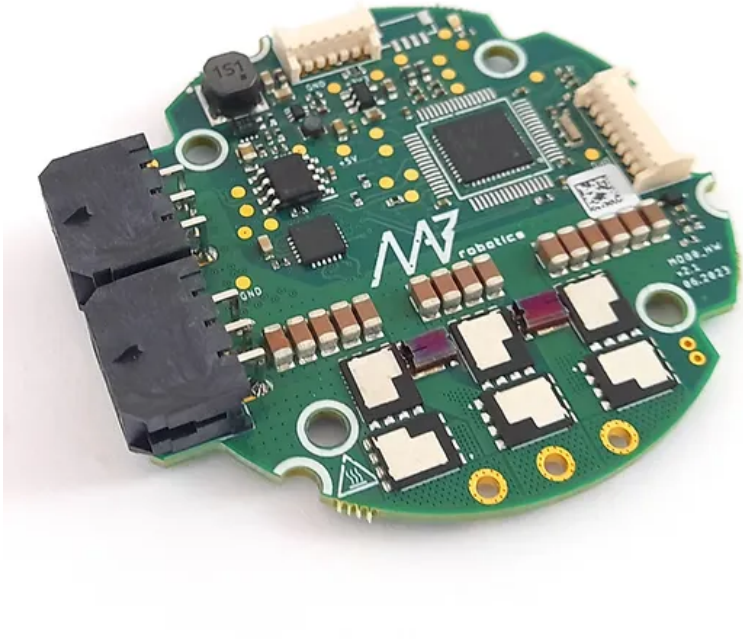
MD x CANdle is a system of brushless actuator controllers (MD series motor controller) and translator devices (CANdle) used for interfacing with them. MD series controller actuators can be used in

Skip to main content

advanced robotic projects like quadrupedal robots, robotic manipulators, exoskeletons, gimbals, and many more.

MD series motor controller

MD series motor controller is a highly integrated brushless motor controller. It can be interfaced with a great variety of motors to turn them into advanced servo actuators. MD series motor controller can work with both direct drive (no gearbox) and geared motors.



CANdle and CANdle HAT

CANdle (CAN + dongle) is a translator device between the host controller and the MD series motor controller drivers. It is possible to interface CANdle with a regular PC over USB bus or CANdle HAT with SBCs (such as Raspberry PI) over USB, SPI or UART bus. The latest CANdle and CANdle HAT devices are fitted with a switch for enabling and disabling the built-in termination resistor.

Note

Skip to main content

Safety information

Even though the MD series motor controller is a small brushless controller it can push a substantial amount of current through the motor windings. This means there are many hazards related to the high torques that the actuator is able to produce as well as elevated temperatures that may occur close to the MD series motor controller controller or the motor. Always make sure the actuator is mounted firmly and does not pose a threat to its surroundings. Make sure the power supply current limits are set to low values (~1A) as an additional safety measure when you are unsure about the tested behavior of the system.

Operating conditions (MD series motor controller and

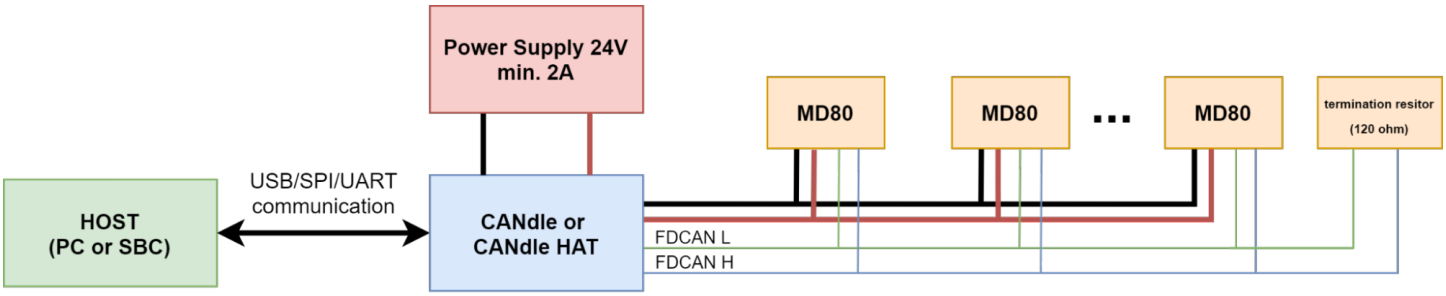
Skip to main content

CANdle)

Ambient Temperature (Operating)	0°C - 40°C
Ambient Temperature (non-operating)	0°C - 60°C
Maximum Humidity (Operating)	up to 95%, non-condensing at 40 ºC
Maximum Humidity (Non-Operating)	up to 95%, non-condensing at 60 ºC
Altitude (Operating)	-400 m to 2000 m

Hardware setup

CAN bus topology is a single-line network structure. A typical hardware connection/wiring scheme for CANdle x MD ecosystem is presented in the picture below:



Hint

In case you'd like to read more about the recommended lengths of the bus segments we suggest the elektormotus guide.

The CAN bus termination is a single Molex connector with an embedded 120Ohm resistor. Termination should be always be attached to the last controller in a string.

Fig. 2 CANdle MD-actuator string (USB bus)

Fig. 3 CANdle HAT MD-actuator string (SPI/UART bus using Raspberry Pi 4)

Skip to main content

Measurements

MD drivers are equipped with sensors that allow for measuring the motor position, velocity, and torque. Whether the motor has an integrated gearbox or not, **the position, velocity, and torque** are in the output shaft reference frame. This means that changing the position from 0.0 to 2π radians, will result in approximately one rotation of the motor for direct-drive (gearless) servos and approximately one rotation of the gearbox output shaft for geared motors.

Position

To measure the position of the rotor the MD driver uses an internal magnetic encoder. The resolution of the encoder is 14 bits (16384 counts per rotation). The drive aggregates all the measurements to provide **multi-rotation positional feedback**. The reference position (0.0 rad) is set by the user and stored in the non-volatile memory. Please see `mdtool config zero` command for more information on how to set the desired zero position.

> **Note**
>
> When using geared actuators with gear ratios above 1:1 it is not possible to determine the position after startup unambiguously, since the motor completes multiple rotations per single rotation of the output shaft. For example, for a 2:1 gearbox, there are two sections within a single output shaft rotation where the motor shaft is in the same position. Unless the motor is placed in the wrong “section” during startup the absolute encoder functionality will work. To deal with this issue please see the output encoder.

Velocity

The velocity is estimated by measuring position change in time, at a frequency of 40kHz. The measurements are then filtered using a low-pass filter with a cut-off frequency of 5 kHz since the position differentiation method introduces noise.

Skip to main content

Torque

Actuator torque is estimated by measuring motor phase currents. This method can be used on low-gear ratio actuators (preferably below 9:1), that are easily back-drivable, to get an estimate of the torque applied by the motor. In applications with higher gear ratios, the torque readout might be less accurate due to excessive friction in the gearbox.

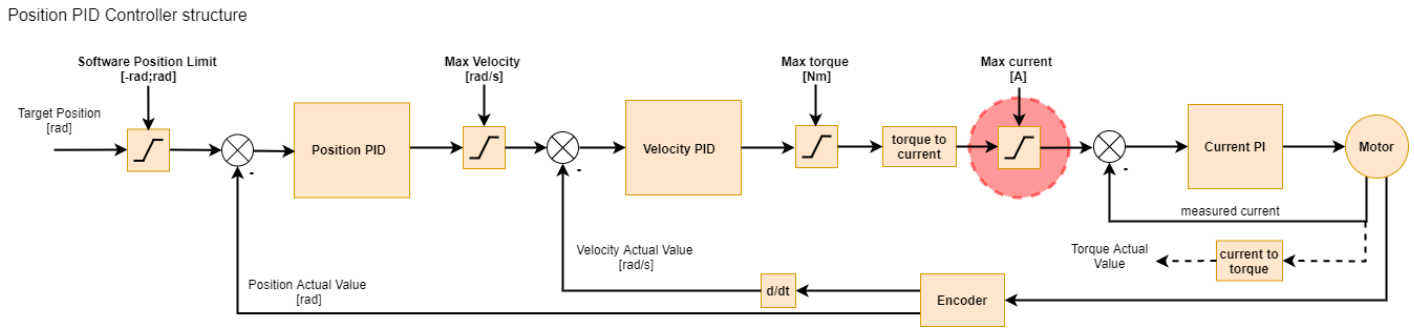
Safety limits

Safety limits are implemented to limit the actuator parameters, to protect the controller or motor from overheating, as well as the surrounding environment from too-powerful actuator movements. Limits apply to: position, velocity, torque, phase current, and temperature of the MOSFETs and the motor.

> **Warning**
>
> Setting the max current limit to above the maximum continuous current may damage the MDxx controller if the maximum torque is commanded for a prolonged period.

Current Limit

Let's start with the max current limit:



This setting limits the maximum current (and thus torque) the motor controller can output. It is the last user-configurable limit in the control scheme. The maximum current is set using the `mdtool config current` command, and by default, it is usually set to 10A. This setting can be saved in the non-

Skip to main content

volatile memory so that it is always loaded on the actuator power-up. To estimate the maximum current setting for a particular motor, you should use the following formula:

$$I[A] = \frac{\tau[Nm]\frac{1}{G_r}}{K_t[\frac{Nm}{A}]}$$

where

- $I[A]$ - calculated current in Amps
- τ - desired maximum torque
- G_r - gear ratio
- K_t - motor's torque constant

for example let's calculate the max current limit for AK80-9 motor, for a 2Nm max torque:

$\tau = 2Nm$

$G_r = 9:1 -> 9$

$K_t = 0.091Nm/A$

$I[A] = \frac{2[Nm]\cdot\frac{1}{9}}{0.091[\frac{Nm}{A}]}$

$I[A] \approx 2.44A$

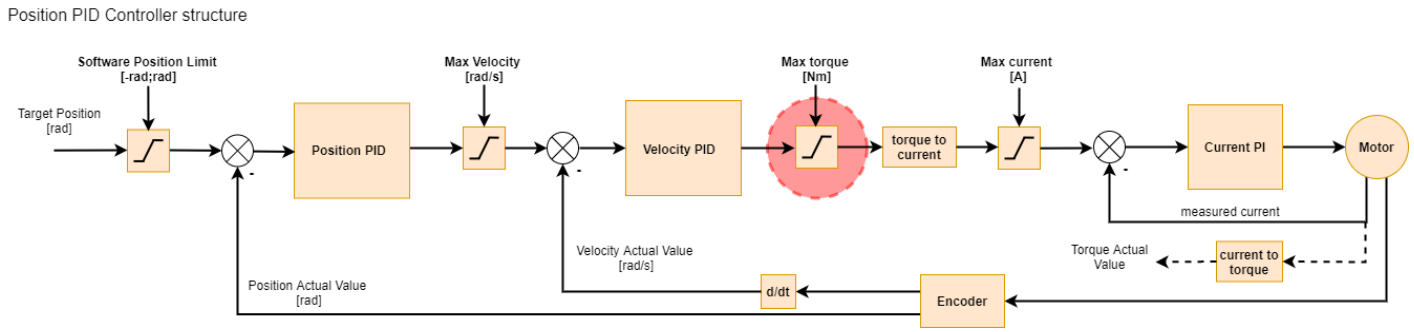
> **Note**
>
> Usually this limit should be set to the highest peak torque that is allowed in the system so that it doesn't limit the actuator performance.

Now, to save this value into the MDxx please refer to `mdtool config current` command. Don't forget to save it with the `mdtool config save` command.

Skip to main content

Torque Limit

The next limit is the max torque limit which can be set using the CANdle script. This limit applies to maximum torque and is expressed in Nm. It is respected in all motion modes. When target torque, set either by either of the controllers, exceeds the `max torque` param the target torque is limited and a motion warning is generated.

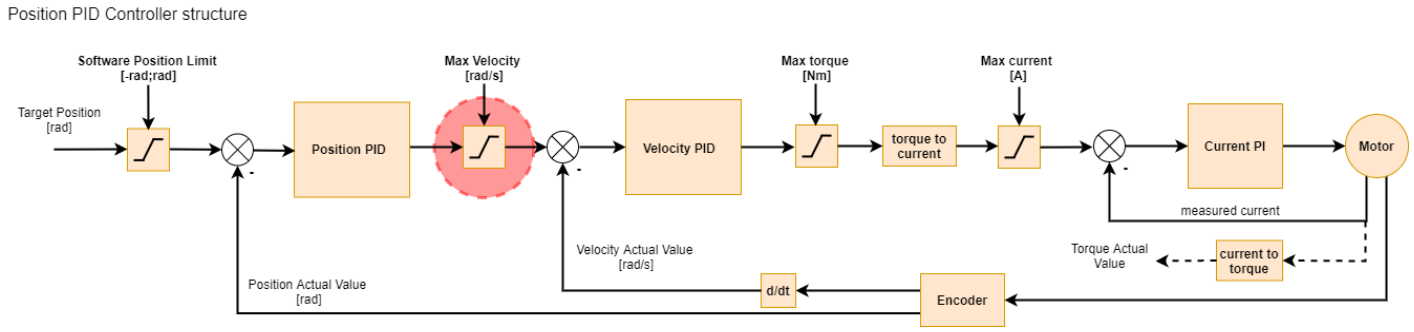


> **Note**
>
> if the torque bandwidth is set to a low value it is possible to read torque values that are above limits when external torque is applied (for example during impacts). This is only true in transition states - when the load is constant the limits will work as expected. This is because with low torque bandwidth the internal torque PI controllers may be too slow to compensate for rapidly changing torque setpoint when hitting the torque/current limit. If you care about accurate torque readout be sure to play with the torque bandwidth parameter and possibly increase it from the default level.

Velocity Limit

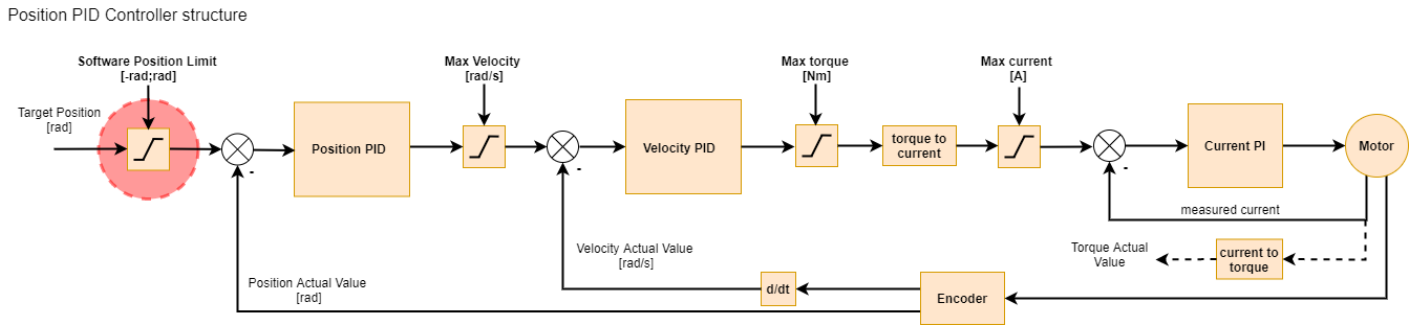
Velocity limit is respected only in Velocity PID / Profile Velocity and Impedance / Position PID / Profile Position modes. When target velocity, set either by the user or the Position PID, exceeds the `max velocity` param the target is limited, and a motion warning is generated.

Skip to main content



Position Limit

Position limit is respected only in Impedance / Position PID / Profile Position modes. When target position, set by the user, exceeds the `<position limit min : position limit max>` param range the target is limited to that range, and a motion warning is generated. Attempt to start the motor outside the range will generate a motion error.



FDCAN Watchdog

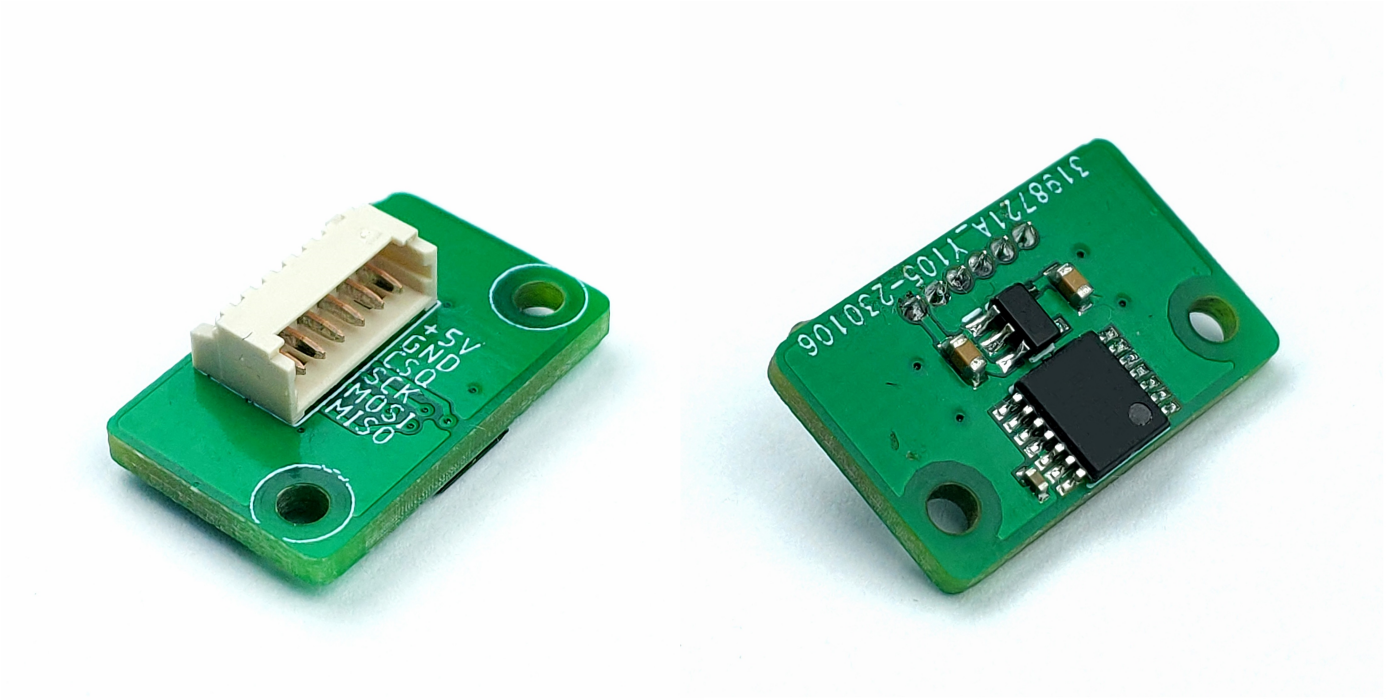
MD drivers feature an FDCAN Watchdog Timer. This timer will shut down the drive stage when no FDCAN frame has been received in a specified time. This is to protect the drive and its surroundings in an event of loss of communications, for example by physical damage to the wiring. By default, the watchdog is set to 250ms. This time can be set to any value in the range of 1 to 2000ms using `mdtool config can` command. When the watchdog is set to 0, it will disable the timer, however, this can lead to dangerous situations, and it is not a recommended way of operating MDxx.

> **Warning**
>
> We do not recommend disabling the CAN watchdog timer.

Skip to main content

Output Encoder

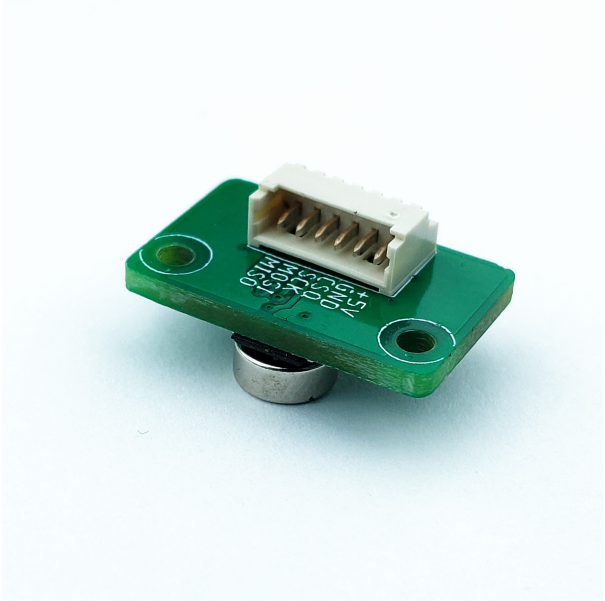
Output encoder is a position sensor that can be attached to the output shaft of the actuator. It is usually useful for geared motors where the output shaft position after startup cannot be determined unambiguously using the MD's onboard encoder due to the gearbox. By using an output encoder one can make sure that the output shaft position is always known at startup.



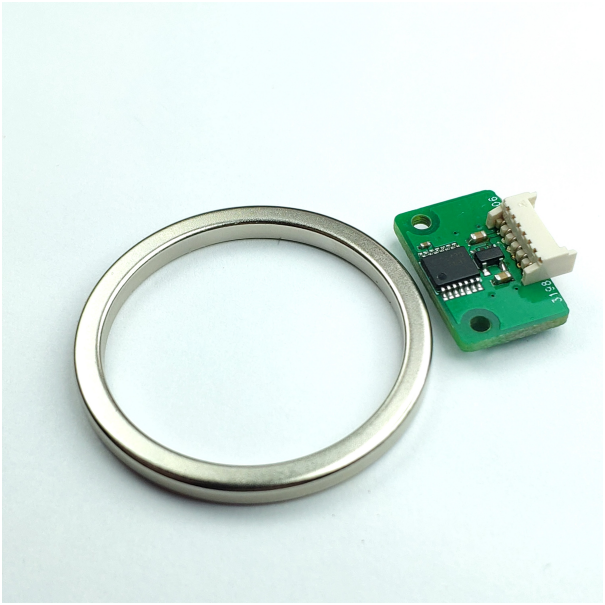
Currently we support one encoder type with two placement configurations:

- AS5047 placed axially on the output shaft with a regular diametrically magnetized magnet

Skip to main content



- AS5047 placed non-axially together with a diametrically magnetized ring magnet



Configuration

The output encoder configuration is performed in the motor config files and saved to the MD series motor controller using the `mdtool setup motor` command. There are only two parameters used in output encoder setup:

```
[output encoder]
```

Skip to main content

```
output encoder mode = <encoder mode>
```

where:

<encoder type>	Description
AS5047_CENTER	for axially placed AS5047 encoder
AS5047_OFFAXIS	for non-axially placed AS5047
MB053SFA17BENT00	Renishaw RS422 17-bit RLS encoder
CM_OFFAXIS	CubeMars motors offaxis encoder

<encoder mode>	Description
STARTUP	initial position from **output encoder**, report **main encoder** values, motion based on **main encoder**
MOTION	initial position from **output encoder**, report **output encoder** values, motion based on **output encoder**
REPORT	initial position from **main encoder**, report **output encoder** values, motion based on **main encoder**, calibration of the **output encoder** is impossible
MAIN	**output encoder** is used as the **main encoder**. All **output encoder** measurements are mapped as **main encoder** values.
CALIBRATED_REPORT	initial position from **main encoder**, report **output encoder** values, motion based on **main encoder**, calibration of the **output encoder** is possible

Skip to main content

Warning

The non-axial configuration outputs a nonlinear position values. This means it requires a full calibration (your setup should be able to rotate by at least one full rotation), and in case of the report mode it will output nonlinear position and velocity readings that will have to be compensated in the host's software

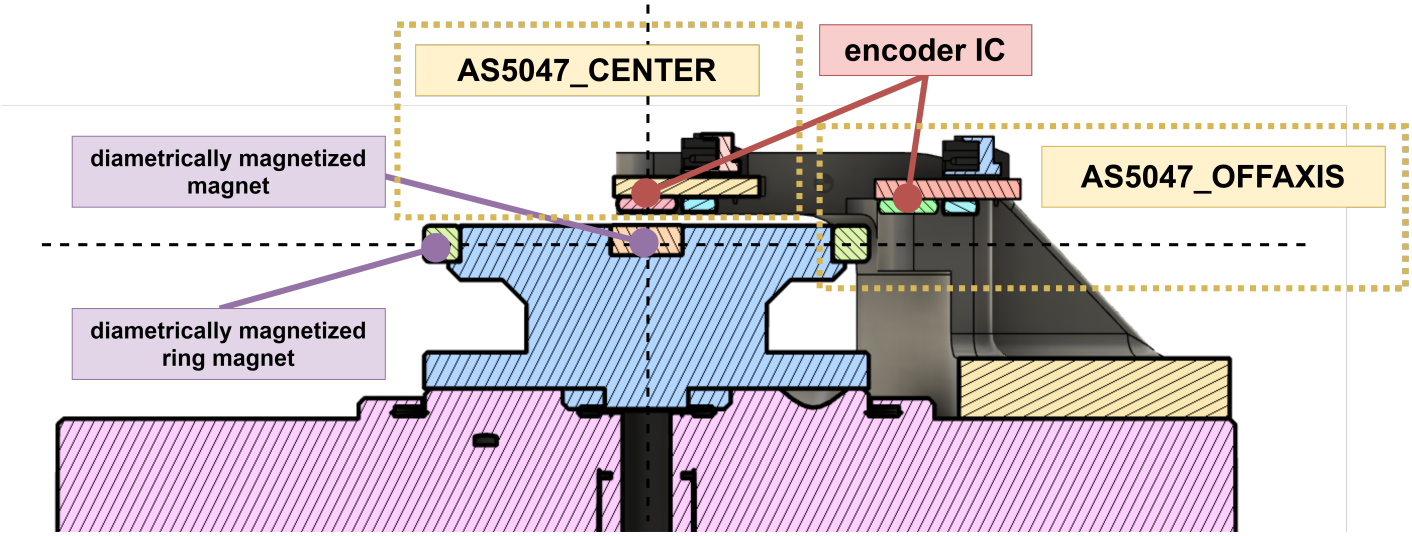
Not all modes are recommended for every encoder. The non-axially placed AS5047 encoder is inherently more noisy and less accurate and thus we recommend using it only in STARTUP mode. Please refer to the table below:

<encoder type>	Valid modes	Description
AS5047_CENTER	STARTUP / MOTION / REPORT / MAIN / CALIBRATED_REPORT	-
AS5047_OFFAXIS	STARTUP	This configuration is much more noisy than the axial placement
MB053SFA17BENT00	STARTUP / MOTION / REPORT / MAIN / CALIBRATED_REPORT	-
CM_OFFAXIS	STARTUP	Only offaxis configuration is supported

Steps to add an external encoder to the driver setup:

- make sure the encoder sensor is placed correctly:

Skip to main content



- - in case of axially placed sensors make sure they are placed in center at correct height above the magnet (1 mm is usually optimal)
  - in case of non-axial configuration make sure the magnet is close to the ring magnet (<0.5mm) and the sensor IC is at least 2mm above or below the ring magnet horizontal plane.
- Connect the MDxx with the encoder using a picoblade series cable assembly and connect power to the MDxx.
- Modify the motor config file according to your setup and save it to the MDxx using mdtool

```
MAB@tutorial:~/mdtool/md_tools/build/packages$ mdtool setup motor 100 AK80-9_aux_center_start.cfg
[MDTOOL] MDtool version: v1.2.1
[CANDLE] CANdle library version: v3.1
[CANDLE] Device firmware version: v2.0
[CANDLE] CANdle at /dev/ttyACM0, ID: 0xac12800687432d34 ready (USB)
[CANDLE] Added MD80 with ID: 100  [OK]
[CANDLE] Saving in flash successful at ID: 100  [OK]
```

You can confirm the setup using the `mdtool setup info` command to make sure all parameters are correct:

Skip to main content

```
MAB@tutorial:~/mdtool/md_tools/build/packages$ mdtool setup info 100
[MDTOOL] MDtool version: v1.2.1
[CANDLE] CANdle library version: v3.1
[CANDLE] Device firmware version: v2.0
[CANDLE] CANdle at /dev/ttyACM0, ID: 0xac12800687432d34 ready (USB)
[CANDLE] Added MD80 with ID: 100  [OK]
Drive 100:
- actuator name: AK80_9_AUX_CENT_STA
- CAN speed: 1 M
- CAN termination resistor: disabled
- gear ratio: 0.111
- firmware version: V2.1
- hardware version: HW20
- build date: 30.1.2023
- commit hash: 214328da
- max current: 20.0 A
- bridge type: 3
- pole pairs: 21
- KV rating: 0 rpm/V
- motor shutdown temperature: 80 *C
- motor torque constant: 0.0910 Nm/A
- motor stiction: 0.000 Nm
- motor friction: 0.000 Nm
- d-axis resistance: 0.000 Ohm
- d-axis inductance: 0.000000 H
- torque bandwidth: 500 Hz
- CAN watchdog: 200 ms
- main encoder last check error stddev: 0.000000 rad
- main encoder last check min error 0.000000 rad
- main encoder last check max error: 0.000000 rad
- output encoder: AS5047_CENTER
- output encoder mode: STARTUP
- output encoder position: -2.326282 rad
- output encoder velocity: -0.028824 rad/s
- output encoder last check error stddev: 0.000000 rad
- output encoder last check min error 0.000000 rad
- output encoder last check max error: 0.000000 rad
- position: -2.33 rad
- velocity: -0.02 rad/s
- torque: 0.04 Nm
- MOSFET temperature: 30.50 *C
- motor temperature: 0.00 *C

***** ERRORS *****
- main encoder error:   0x4     (ERROR_EMPTY_LUT, )
- output encoder error: 0x4     (ERROR_EMPTY_LUT, )
- calibration error:    0x7     (ERROR_OFFSET_CAL, ERROR_RESISTANCE_IDENT, ERROR_INDUCTANCE_IDENT, )
- bridge error:         0x0     (ALL OK)
- hardware error:       0x0     (ALL OK)
- communication error:  0x0     (ALL OK)
```

Note

At this point some errors will be present as the setup is not yet calibrated.

- Calibrate the MDxx using `mdtool setup calibration` command
- Calibrate the output encoder using `mdtool setup calibration_out` command
- Test the encoders using `mdtool test encoder` command.

Skip to main content

- Use the `mdtool setup info all` command to make sure there are no errors and the test results (min, max and stddev errors) are within your expectations.
- The external encoder is ready to use! For more information on external encoder parameters please see the output encoder calibration section.

Calibration

Calibration should be performed when the MDxx controller is first mounted to the motor or when anything changes in the motor-controller assembly. It has three main stages during which specific parameters of the setup are measured and saved.

> **Note**
>
> The calibration has to be performed on a motor that is free to rotate with no load attached to its output shaft. If the calibration fails, you will see errors when executing the `mdtool setup info` command. If the failure is critical the MDxx will remain disabled until the next calibration attempt.

Pole pairs detection

In the first stage the motor will execute one full rotor revolution in order to check if the pole pair count is correctly configured. If the detected number of pole pairs is not consistent with the number that was typed in the *.cfg file during motor setup the calibration will fail and an error ERROR_POLE_PAIR_DET will be shown until the next calibration attempt. If you are unsure about the number of pole pairs (you can check it by counting magnets and dividing it by 2) just place zero in the *.cfg file. Then the pole pairs will be automatically detected.

Encoder eccentricity

Encoder eccentricity is the second measurement that takes place. During this part, the motor performs a slow single rotation in both directions to assess the amount of error due to non-axial encoder placement and external magnetic disturbances.

Skip to main content

Motor resistance and inductance

Motor parameters - resistance and inductance are measured in order to calculate the correct current PI controller gains to achieve a certain torque bandwidth (please see the section below). The parameters are measured in the DQ reference frame meaning the resultant resistance and inductance values have to be transformed from either line-to-line quantities or phase quantities.

Torque bandwidth

Even though the torque command on MD controllers seems to be applied instantaneously, in reality, it's not the case. As in every system, there's a response to the command which depends on the system itself and the controller gains. A parameter called bandwidth was introduced to describe how fast the output of a system reacts to the changing input. Calibrating the motor for a certain torque bandwidth requires measuring motor parameters. This happens in the last calibration stage and manifests as an audible sound (beep). The torque bandwidth default setting is set using the motor config file. It can be set to anywhere from 50 Hz to 2.5 kHz, however it is important to note that higher torque bandwidth causes a higher audible noise level. Please see the `mdtool setup calibration` command for more details on calibrating the actuators. When the system that you're designing is a highly dynamic one, you want the torque bandwidth to be higher than the default setting of 50 Hz. Start by calibrating the drives for 1 kHz torque bandwidth, and if you see this is still not enough you can increase it further.

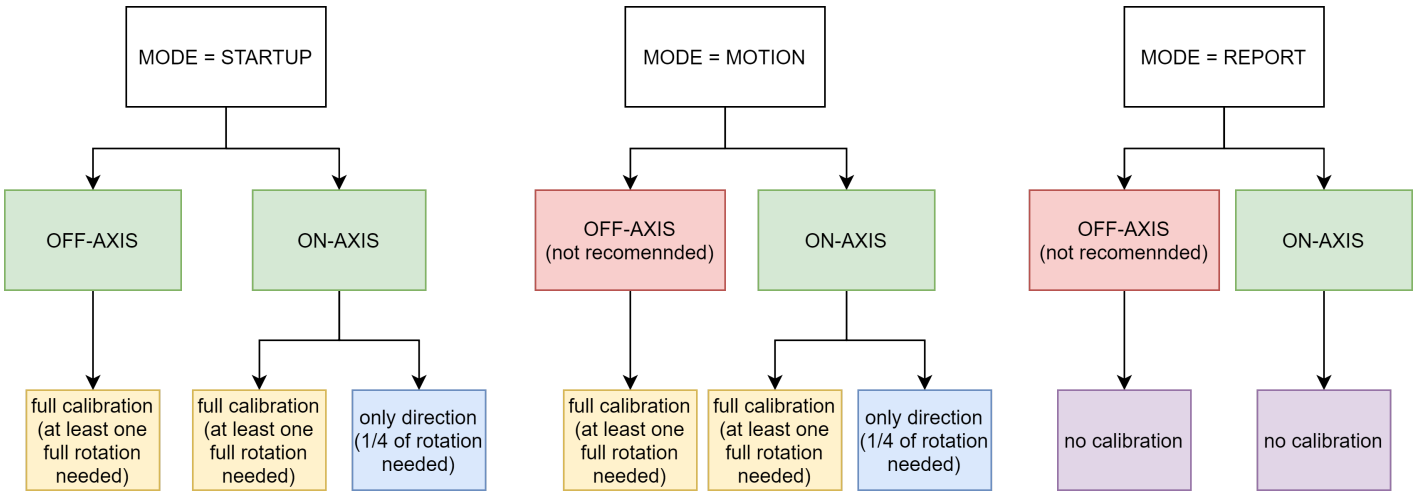
Output Encoder Calibration

The output encoder calibration routine is used to recognize the correct direction of rotation, and record the correction lookup-table to account for non-axial placement of the encoder in respect to the magnet.

> **Warning**
>
> The full calibration routine rotates the actuators output shaft by more than one single rotation in the FULL calibration mode. Please make sure the shaft is free to rotate during the test.

Skip to main content

In case your setup is not able to complete a full rotation due to mechanical constraints you can set the `output encoder calibration mode` to `DIRONLY` in your *.cfg file in the [output encoder] section. This way the calibration will end on the first stage - checking the correct direction of rotation, so only 1/4 of a full rotation is needed. Please note that this is not possible using off-axis encoder - it requires a full calibration routine.



To run the routine, use the **`mdtool setup calibration_out`** command. After completing the routine the MDxx will reboot and after that it is recommended to run the mdtool setup info command in order to make sure the setup reports no errors:

Skip to main content

```
MAB@tutorial:~/mdtool/md_tools/build/packages$ mdtool setup info 100
[MDTOOL] MDtool version: v1.2.1
[CANDLE] CANdle library version: v3.1
[CANDLE] Device firmware version: v2.0
[CANDLE] CANdle at /dev/ttyACM0, ID: 0xac12800687432d34 ready (USB)
[CANDLE] Added MD80 with ID: 100  [OK]
Drive 100:
- actuator name: AK80_9_AUX_OFF_START
- CAN speed: 1 M
- CAN termination resistor: disabled
- gear ratio: 0.111
- firmware version: V2.1
- hardware version: HW20
- build date: 30.1.2023
- commit hash: 214328da
- max current: 20.0 A
- bridge type: 3
- pole pairs: 21
- KV rating: 0 rpm/V
- motor shutdown temperature: 80 *C
- motor torque constant: 0.0910 Nm/A
- motor stiction: 0.000 Nm
- motor friction: 0.000 Nm
- d-axis resistance: 0.187 Ohm
- d-axis inductance: 0.000032 H
- torque bandwidth: 500 Hz
- CAN watchdog: 200 ms
- main encoder last check error stddev: 0.000000 rad
- main encoder last check min error 0.000000 rad
- main encoder last check max error: 0.000000 rad
- output encoder: AS5047_OFFAXIS
- output encoder mode: STARTUP
- output encoder position: -0.758170 rad
- output encoder velocity: 0.016989 rad/s
- output encoder last check error stddev: 0.000000 rad
- output encoder last check min error 0.000000 rad
- output encoder last check max error: 0.000000 rad
- position: -0.76 rad
- velocity: 0.04 rad/s
- torque: 0.00 Nm
- MOSFET temperature: 35.52 *C
- motor temperature: 0.00 *C

***** ERRORS *****
- main encoder error:   0x0     (ALL OK)
- output encoder error: 0x0     (ALL OK)
- calibration error:    0x0     (ALL OK)
- bridge error:         0x0     (ALL OK)
- hardware error:       0x0     (ALL OK)
- communication error:  0x0     (ALL OK)
```

The output encoder parameters are rather straightforward, except the “output encoder last check” errors. These values are filled during the output encoder check routine, which can be run using `mdtool test encoder output`. These params represent the output encoder errors (max, min and

Skip to main content

are large inaccuracies during the calibration, or the output encoder moves in your setup, you can always check how accurate it is by running the check_aux routine.

Example errors for AS5047_CENTER:

```
- output encoder: AMS5047_CENTER
- output encoder mode: MOTION
- output encoder position: 0.000767 rad
- output encoder velocity: 0.052428 rad/s
- output encoder last check error stddev: 0.001191 rad
- output encoder last check min error -0.007006 rad
- output encoder last check max error: 0.002098 rad
```

Example errors for AS5047_OFFAXIS:

```
- output encoder: AMS5047_OFFAXIS
- output encoder mode: STARTUP
- output encoder position: -0.000383 rad
- output encoder velocity: -0.026279 rad/s
- output encoder last check error stddev: 0.015753 rad
- output encoder last check min error -0.092981 rad
- output encoder last check max error: 0.037362 rad
```

As can be seen, the non-axial encoder features larger errors, and thus can be utilized only for initial position determination rather than output shaft control. In case the errors get too large they will turn yellow after running `mdtool test encoder output` command indicating there might be a problem with your setup:

```
- output encoder: AMS5047_OFFAXIS
- output encoder mode: STARTUP
- output encoder position: 0.003068 rad
- output encoder velocity: -0.114227 rad/s
- output encoder last check error stddev: 0.202380 rad
- output encoder last check min error -0.287952 rad
- output encoder last check max error: 1.023281 rad
```

Note

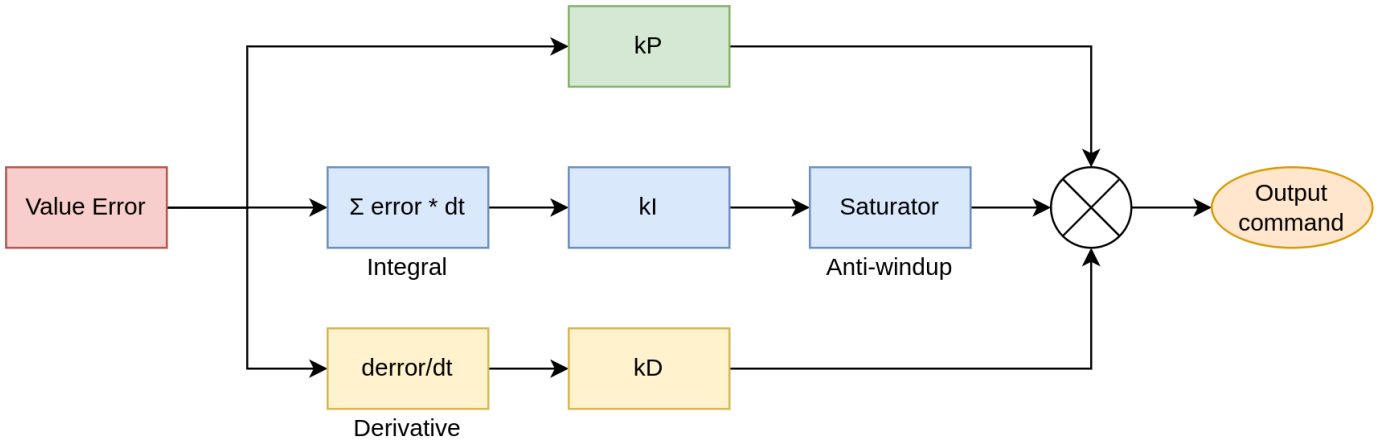
These errors will limit the maximum gear ratio that can be used to unambiguously determine the startup output shaft position. Be sure to keep them as low as possible in your setup.

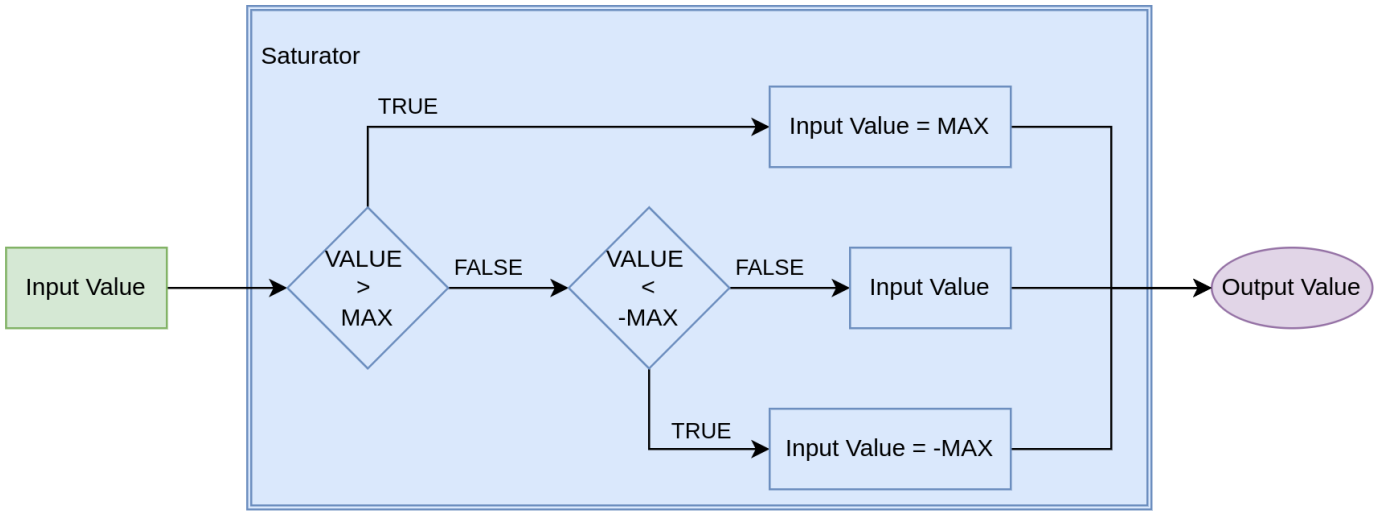
Skip to main content

Motion modes

> **Hint**
>
> TL;DR: MD x CANdle - motion modes

To control the motor shaft with the user's command MDxx is equipped with multiple control loops. All controllers are based on a regular PID controller design with an anti-windup block. The saturator (anti-windup) is an additional module that acts as a limiter to the 'I' part of the controller, as in many systems, the error integration may grow to very large numbers, completely overwhelming 'P' and 'D' parts of the controller.





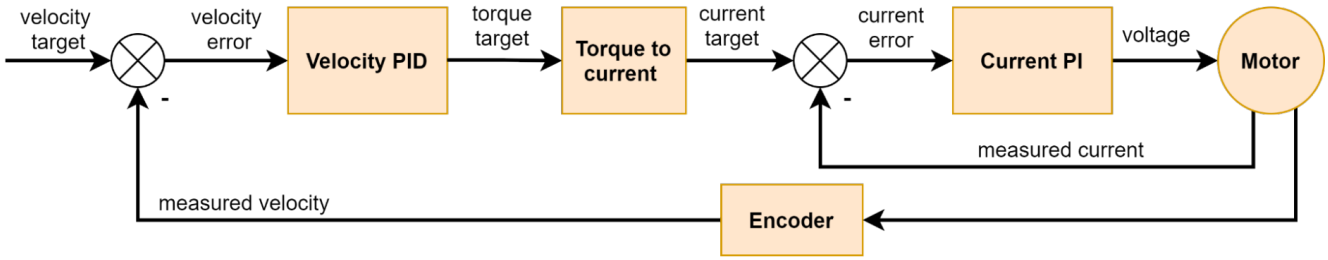
Skip to main content

Velocity PID

Velocity PID controller calculates velocity error based on target velocity (set by user) and estimated velocity read from the encoder. Its output is a torque command for the internal current/torque controller. The parameters of the controller are:

- Velocity Target (in [rad/s])
- kP (proportional gain)
- kI (integral gain)
- kD (derivative gain)
- I windup (maximal output of an integral part in[Nm])

Velocity PID Controller structure



Position PID

Position PID mode is the most common controller mode used in industrial servo applications. In MDxx, it is implemented as a cascaded PID controller. This means that the controller is working in two stages, firstly the position error is calculated, and it is then passed to the Position PID, which outputs the target velocity. This value is then passed as an input to the Velocity PID controller, which outputs commanded torque. This mode uses both Position PID and Velocity PID and thus needs the following parameters:

For Position PID:

- Position Target (in [rad])
- kP (proportional gain)
- kI (integral gain)

Skip to main content

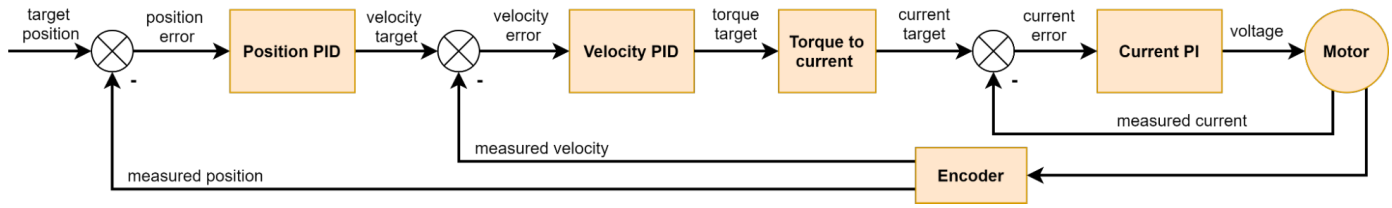
- kD (derivative gain)
- I windup (maximal output of an integral part in [rad/s])

For Velocity PID:

- Velocity Target (in [rad/s])
- kP (proportional gain)
- kI (integral gain)
- kD (derivative gain)
- I windup (maximal output of an integral part in[Nm])

To properly tune the controller, it is recommended to first tune the velocity controller (in velocity PID mode), and then the position PID. The controller can be described with a diagram:

Position PID Controller structure



Impedance PD

Impedance Control mode is a popular choice for mobile or legged robots, as well as for any compliant mechanism. The main idea behind it is to mimic the behavior of a torsional spring with variable stiffness and damping. The parameters of the controller are:

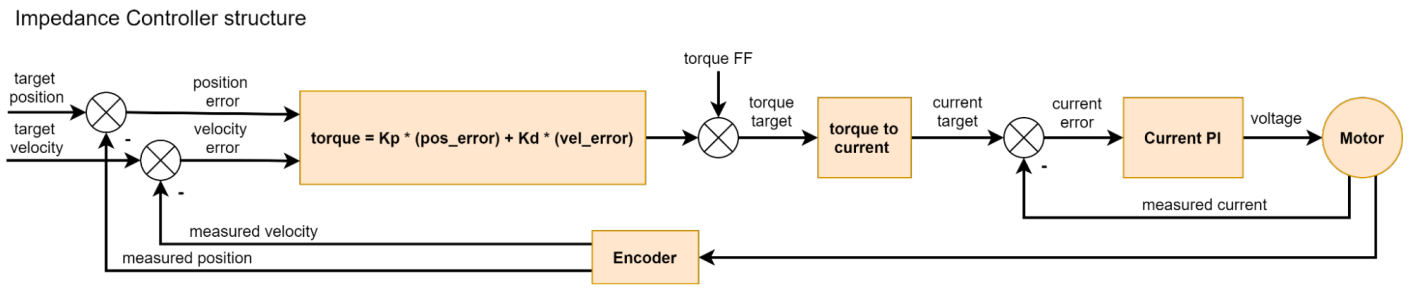
- Position Target
- Velocity Target
- kP (position gain)
- kD (velocity gain)
- Torque Feed Forward (Torque FF)

The torque output is proportional to the position error and velocity error and additionally supplemented with a torque command from the user. Here are some of the most common applications for this control mode:

Skip to main content

- **Spring-damper mechanism** - when velocity target is set to 0, impedance controllers kP gain acts as the virtual spring stiffness and kD as its damping coefficient. Example use case: a variable suspension for a wheeled robot, where suspension stiffness can be regulated by kP, damping by kD, and height (clearance) by changing the target position;
- **High-frequency torque controller**, where its targets and gains can act as stabilizing agents to the torque command. Example use case: In legged robots, force control can be achieved by advanced control algorithms, which usually operate at rates below 100 Hz. It is usually enough to stabilize the robot but too slow to avoid vibrations. Knowing desired robot's joint positions, velocities, and torques, drives can be set to produce the proper torque and hold the position/velocity with small gains. This would compensate for any high-frequency oscillations (vibrations) that may occur, as the impedance controller works at 40kHz (much faster than <100 Hz).
- **Raw torque controller** - when kP and kD are set to zero, the torque_ff command is equal to the output controller torque.
- **Idle** - when kP and kD are set to zero, and the torque_ff command is equal to zero, the motor shaft will be free to rotate. When the drive is disabled it connects all the windings together for safety. This mode can be useful for enabling free rotation of the shaft, but the rotational energy shoudl not be too high as the voltages induced in the motor windings coudl break the driver.

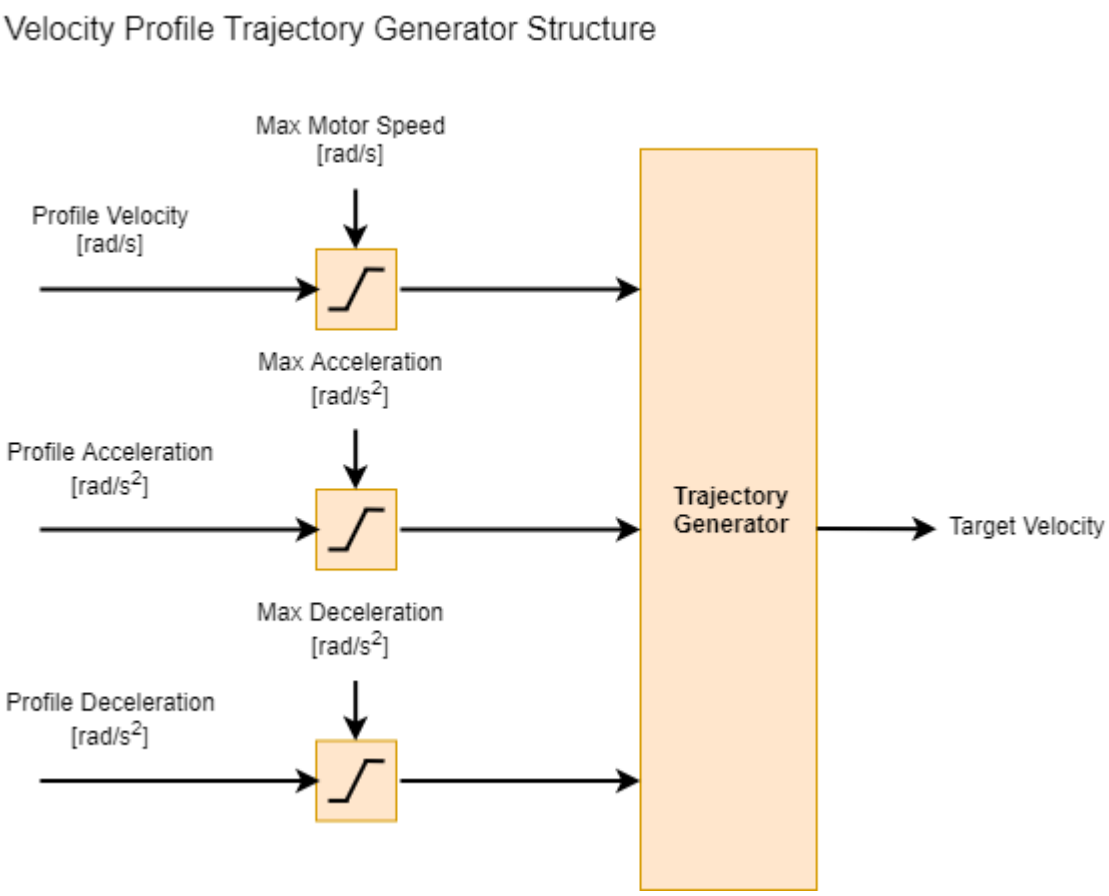
The impedance controller is relatively simple and works according to the schematic below:



Velocity Profile

The velocity profile motion mode uses trapezoidal velocity profiles to achieve smooth velocity changes with predefined acceleration and deceleration. The trajectory generator module interpolates between two velocity setpoints to achieve constant acceleration.

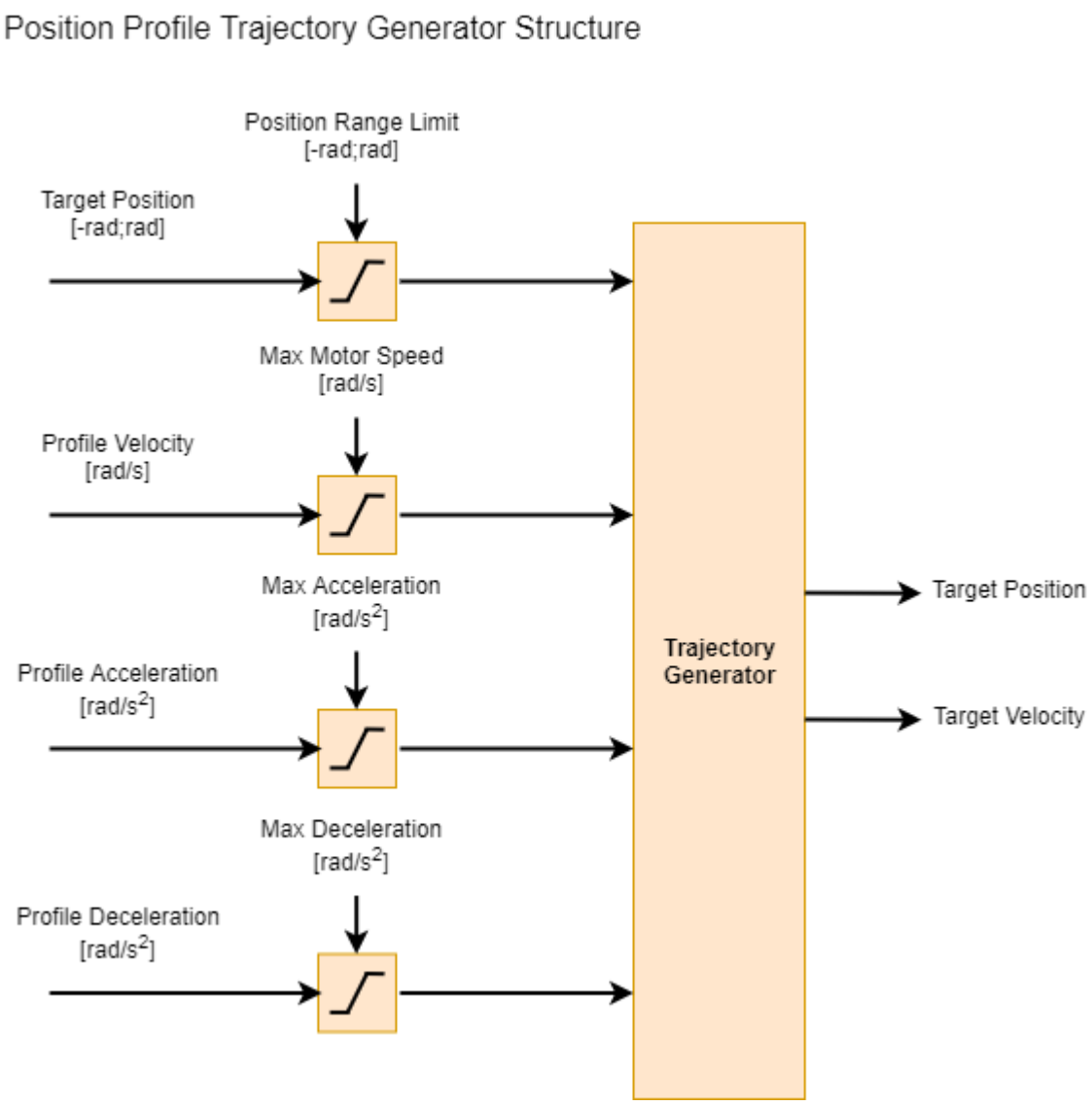
Skip to main content



Position Profile

The position profile motion mode uses trapezoidal velocity profiles to achieve smooth position changes with predefined acceleration, deceleration, and velocity. The trajectory generator module interpolates between two position setpoints to achieve constant acceleration, and linearly changing velocity.

Skip to main content



Controller implementation

Controller implementation can be useful for simulating the actuators in virtual environments. Please find the PID and impedance C/C++ language implementations below:

```
float mab_controller_performPid(PID_controller *c, float act_value)
{
        c->error_last = c->error;
        c->error = c->target - act_value;
        c->integrator += c->error*c->dt;
        c->integrator = mab_commons_range(c->integrator, -c->integrator_windup, c->i
        c->de = (c->error - c->error_last)/c->dt;
        c->output = mab_commons_range( (c->kp * c->error + c->ki * c->integrator + c
        return c->output;
```

```
float mab_controller_performImpedanceController(Impedance_controller *c, float posit
{
        c->output = c->setTorque + c->kp * (c->positionTarget - position) + c->kd *
        c->output = mab_commons_range(c->output, -c->outputMax, c->outputMax);
        return c->output;
}
```

Motion controller tuning

> **Hint**
>
> The best way to get started with tuning is to copy the default gains and tweak them. You can treat this section as our recommendation for tuning the controllers, but online articles can be useful as well

The first step to correctly set up the gains is to start with our default gains. There are three sets of default gains that are set on each motor power up and thus they allow for restoring the actuator to a default state in case some gains were set incorrectly by the user. These gains are also a great starting point for user modifications when the actuator has to be used in a specific application requiring high positioning accuracy or very dynamic movements.

> **Note**
>
> Default gains are set to work with CANdle examples. This way they can be assumed to be universal but it does not have to always be the case

> **Hint**
>
> When something does go wrong during the tuning process just power-cycle the actuator - the default gains will be restored.

> **Warning**
>
> Always keep your safety limits low when experimenting with gains. Gains not suitable for your system may cause oscillations and unstable operation of the MD-based actuators

Velocity PID / Velocity Profile

1. Start by slowly increasing kp gain of the controller keeping ki kd and iWindup set to zero
2. increase set iWindup to 1.0 and try increasing ki to see if it helps to reach the setpoint velocity
3. Generally try to avoid using kd

Position PID / Position Profile

1. Tune the Velocity PID first - make sure in Velocity PID mode the actuator is able to follow the setpoints
2. Start by slowly increasing kp gain of the position controller keeping ki kd and iWindup set to zero
3. Increase set iWindup to 1.0 and try increasing ki to see if it helps to reach the setpoint velocity
4. Generally try to avoid using kd

Impedance PD

1. Increase kp to the point you’re satisfied with the stiffnes of the output shaft (the spring coefficient)
2. Increase kd to the point youre satified with the damping of the output shaft (the damping coeficient)
3. Avoid setting kd too high - it may cause severe vibrations.

Current PI

Current/torque PI is the lowest-level controller. Its gains are not directly user-configurable, however, they can be modified using the bandwidth parameter. Please see the calibration section for more insight on the topic.

Config

MDxx’s config allows for configuring the controller for a specific motor and the application it is used in. This section will cover the parameters that are used in config files.

[motor] section

- `name` - actuator name. Max 20 characters
- `pole pairs` - the number of rotor magnets divided by 2. If you are unsure type zero here - during calibration it will be autodetected. Later on it is advised to retype it after the calibration to the config file. It can be accessed using mdtool setup info or by register access.
- `KV` - declated KV of the motor - its used when torque constant is set to zero.
- `torque constant` - motor torque constant in Nm/A
- `gear ratio` - gear ratio -> example 6:1 reductor is 0.166666 whereas 1:2 multiplicator is 2
- `max current` - maximum allowable phase (not power supply) current
- `torque bandwidth` - torque bandwidth setting
- `shutdown temp` - temperature threshold in [*C] *of the motor that will cause a overtemperature stop. Note: this safety limit works only with a motor thermistor connected. If motor temp is 0*C when mdtool setup info is called, the thremistor is not populated or is not working.

[limits] section

Global limits used to issue warnings or errors.

- `max torque` - maximum allowed torque in Nm
- `max velocity` - maximum allowed velocity in rad/s
- `max position` - maximum position limit in rad
- `min position` - minimum position limit in rad
- `max acceleration` - max acceleration in rad/s^2
- `max deceleration` - max deceleration in rad/s^2

[profile] section

These settings are respected in POSITION PROFILE and VELOCITY PROFILE modes.

- `acceleration` - profile acceleration in rad/s^2

Skip to main content

- `velocity` - profile velocity in rad/s

[output encoder] section

For more information please refer to output encoder section.

- `output encoder` - valid types: AS5047_CENTER, AS5047_OFFAXIS, MB053SFA17BENT00, CM_OFFAXIS
- `output encoder mode` - valid modes: STARTUP, MOTION, REPORT, CALIBRATED_REPORT

[position PID] section

Position PID default gains (used at every startup, then can be modified using the C++/Python script, or register access)

- `kp` - proportional gain
- `ki` - integral gain
- `kd` - derivative gain
- `windup` - integral limit

[velocity PID] section

Velocity PID default gains (used at every startup, then can be modified using the C++/Python script, or register access)

- `kp` - proportional gain
- `ki` - integral gain
- `kd` - derivative gain
- `windup` - integral limit

[impedance PD] section

Impedance PID default gains (used at every startup, then can be modified using the C++/Python

- `kp` - proportional gain
- `kd` - derivative gain

Status

When an abnormal situation takes place the controller sets an error bit indicating a particular error or warning. The table below lists all available error and warning codes and their descriptions. The easiest way to check all statuses is to use mdtool. Another way could be to use the CANdle lib register access and read the statuses, or decode the general “Quick Status” using the CANdle lib getQuickStatus() function.

Errors and warnings can be cleared by register access, or using `mdtool clear` command. Please note that all warnings and only non-critical errors can be cleared.

Quick Status

Quick status provides a general info about errors in each category of statuses. No warnings are indicated here. Last bit indicatest whether the current target (position or velocity) has been reached.

Error bit	Error description
0	Main encoder error
1	Output encoder error
2	Calibration encoder error
3	MOSFET bridge error
4	Hardware errors
5	Communication errors
6	Motion errors
8-14	RESERVED
15	Target position (in Position PID / Profile position mode) or velocity (in Velocity PID / Velocity profile mode) reached within position or velocity window

Skip to main content

Main / Output Encoder Errors

Error name	Error bit	Error description	Action to clear it
ERROR_COMMUNICATION	0	MDxx could not communicate with the encoder	Check connections
ERROR_WRONG_DIRECTION	1	Indicates the calibrated output encoder direction is different from the main encoder direction	Recalibrate
ERROR_EMPTY_LUT	2	Indicates the encoder eccentricity table is empty	Recalibrate
ERROR_FAULTY_LUT	3	Indicates the encoder eccentricity table is faulty (contains too large corrections)	Check the setup and recalibrate
ERROR_CALIBRATION_FAILED	4	Calibration failed due to wrong motor <> encoder setup	Check setup, recalibrate in case of problems contact MABRobotics
ERROR_POSITION_INVALID	5	Position reading is invalid	Check endcoder physical setup, in case of problems contact MABRobotics
ERROR_INIT	6	Encoder initialization failed	Check endcoder setup and connection, in case of problems contact MABRobotics
WARNING_LOW_ACCURACY	30	Encoder position readout accuracy may be lower than specified	Check endcoder physical setup and reboot the MDxx

Calibration Errors

Error name	Error bit	Error description	Action to clear it
ERROR_OFFSET_CAL	0	Problem with the offset determination during calibration	Try recalibrating
ERROR_RESISTANCE_IDENT	1	Problem with resistance identification	Try recalibrating or running the `mdtool config bandwidth` command
ERROR_INDUCTANCE_IDENT	2	Problem with inductance identification	Try recalibrating or running the `mdtool config bandwidth` command
ERROR_POLE_PAIR_CAL	3	Problem with pole pair detection routine	Try recalibrating
ERROR_SETUP	4	Problem with motor config file download, or the setup parameters themselves	Check the config file again and try to upload one more time

Skip to main content

Bridge errors

Error name	Error bit	Error description	Action to clear it
ERROR_BRIDGE_COM	0	Communication problem with the bridge	Contact MABRobotics
ERROR_BRIDGE_OC	1	The bridge detected overcurrent	Lower the current limit, clear the error or restart the drives

Skip to main content

ERROR_BRIDGE_GENERAL_FAULT	2	Usually indicates a hardware issue	Contact MABRobotics

Hardware errors

Error name	Error bit	Error description	Action to clear it
ERROR_OVER_CURRENT	0	Overcurrent detected	Lower the current limit, clear the error or restart the drive
ERROR_OVER_VOLTAGE	1	Overvoltage detected	Lower the system voltage, avoid rapid braking in the system, use a modern PSU, or a LiPo battery
ERROR_UNDER_VOLTAGE	2	Undervoltage detected	Ensure your power supply has enough current capability for your system
ERROR_MOTOR_TEMP	3	Motor temperature exceeded the limit set in the config file	Wait for the motor to cool down
ERROR_MOSFET_TEMP	4	MDxx power side exceeded the limit of 100*C	wait for the MDxx to cool down
ERROR_ADC_CURRENT_OFFSETS	5	Error during adc current offsets calibration	Usually indicates a hardware error - contact MABRobotics

Comunication errors

Error name	Error bit	Error description	Action to clear it
WARNING_CAN_WD	30	Indicates the communication with the host was ended by the watchdog	make sure candle.end() is called in your script, clear using mdtool

Motion status

Error name	Error bit	Error description	Action to clear it
ERROR_POSITION_OUTSIDE_LIMITS	0	Current shaft position is outside the limits from the config file	Re-home the actuator, set a temporary zero to move it back into the limits, or increase the limit range, clear using mdtool
ERROR_VELOCITY_OUTSIDE_LIMITS	1	Velocity exceeded the max velocty param	Ensure the velocity limit is set to a proper value, clear using mdtool
WARNING_ACCELERATION_CLIPPED	24	Acceleration command was clipped to max acceleration at least once	Check acceleration limits, clear using mdtool
WARNING_TORQUE_CLIPPED	25	Torque command was clipped to max torque at least once	Check torque limits, clear using mdtool
WARNING_VELOCITY_CLIPPED	26	Velocity command was clipped to max velocity at least once	Check velocity limits, clear using mdtool
WARNING_POSITON_CLIPPED	27	Position command was clipped to either max or min position at least	Check position limits, clear using mdtool

Skip to main content

The following table shows when warnings and errors are issued based on the mode the controller is currently in:

	setpoints				measured values	
	torque clipped	acceleration clipped	velocity clipped	position clipped	velocity outside limits	position outside limits
IDLE	OFF	OFF	OFF	OFF	OFF	OFF
SERVICE	OFF	OFF	OFF	OFF	OFF	OFF
POSITION_PID	WARNING	OFF	WARNING	WARNING	ERROR	ERROR
POSITION_PROFILE	WARNING	WARNING	WARNING	WARNING	ERROR	ERROR
VELOCITY_PID	WARNING	OFF	WARNING	OFF	ERROR	ERROR
VELOCITY_PROFILE	WARNING	WARNING	WARNING	OFF	ERROR	ERROR
RAW_TORQUE	WARNING	OFF	OFF	OFF	ERROR	ERROR
IMPEDANCE	WARNING	OFF	WARNING	WARNING	ERROR	ERROR

Utilities

GPIO

All of the MD controllers have two multi-purpose GPIO pins. Currently they have two functionalities:

- Auto Brake - in this mode MDxx will automatically engage MAB Robotics's provided braking systems via GPIO A pin, please contact us for more information in that regard
- GPIO input - in this mode MDxx will output GPIO pin states to state register (*userGpioState 0x161*)

Important

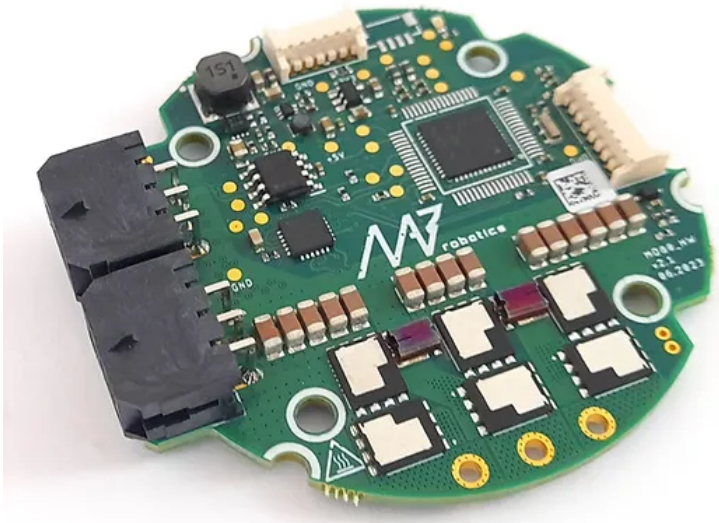
The GPIO are connected directly to board's MCU. **DO NOT** drive this pins higher than +5V or lower than 0V as this **WILL** damage the board.

Also the GPIO pins are floating so if you want to use those pins as an external function switches, do so with proper pull-up/down resistor or push-pull circuitry.

Skip to main content

Controllers

MD80



Skip to main content

MD20

MD80

General parameters

MD80 is a brushless motor controller. It can work with a variety of motors and reducers that can be precisely matched to the users' specifications. All MD80 variants are using an advanced motor control algorithm (FOC), a high-resolution encoder, a high-speed FDCAN communication bus, and a common communication interface. The controller foratures position PID, velocity PID, impedance

Skip to main content

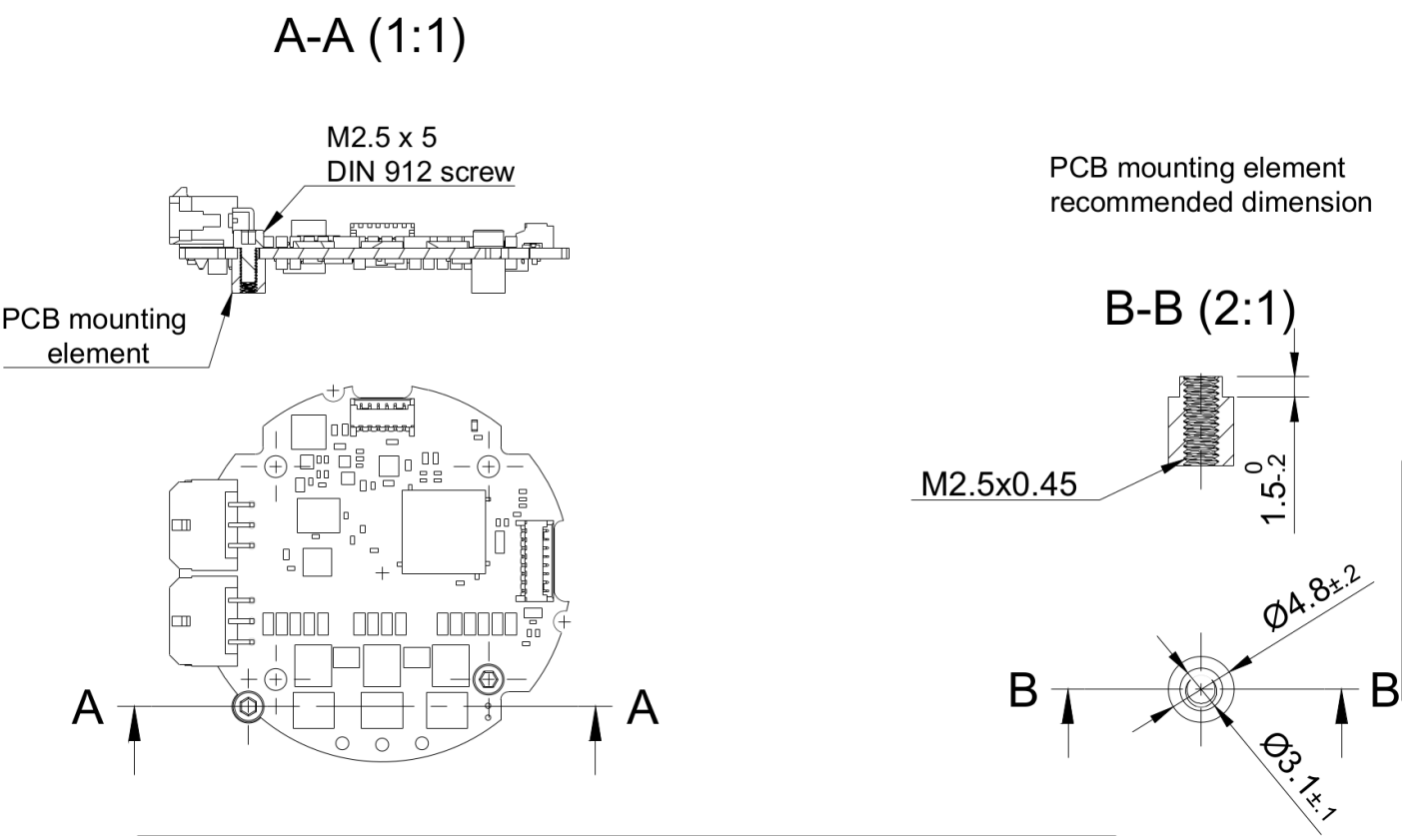
profile position and profile velocity operation modes. MD80 can be easily daisy-chained, for easy connection of many drives in a single control network.

> **Hint**
>
> You can easily check your MD80 version using the `mdtool setup info` command.

General parameters table for MD80 V2.0 and V2.1:

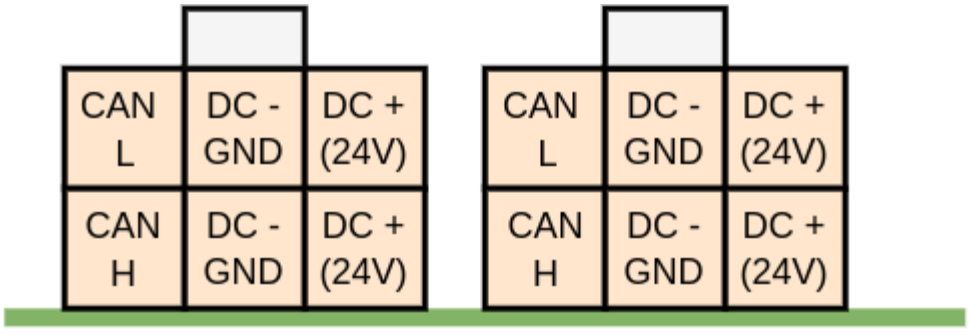
Parameter	Value
Input Voltage	**10 - 48 VDC**
Nominal Input Voltage	24 VDC
Max Input Current	10 A (RMS)
Max Continuous Phase Current	20 A
Max Peak Phase Current (t = 2 s)	**80 A**
Built-in software-controlled termination resistor	**optional**
FDCAN Baudrate (adjustable)	1/2/5/8 Mbps
Position PID Controller Execution Frequency	1 kHz
Velocity PID Controller Execution Frequency	5 kHz
Impedance Controller Execution Frequency	40 kHz
Torque Control Execution Frequency	40 kHz
Torque Bandwidth (adjustable)	50 Hz - 2.5 kHz
External encoder connector (SPI, RS422)	**yes**
External 5V power supply max current	**150 mA**

Skip to main content



Connectors pinout

The connectors used in the system on the CAN FD side are MOLEX Micro-Fit series 3.0. Both connectors are connected in parallel for easy daisy-chaining. The connector pinout is presented below:



Skip to main content

The colors of the corresponding wires in the Molex socket, as supplied by MAB (looking from the side of the wires):

Yellow	White	Pink
Grey	Brown	Green

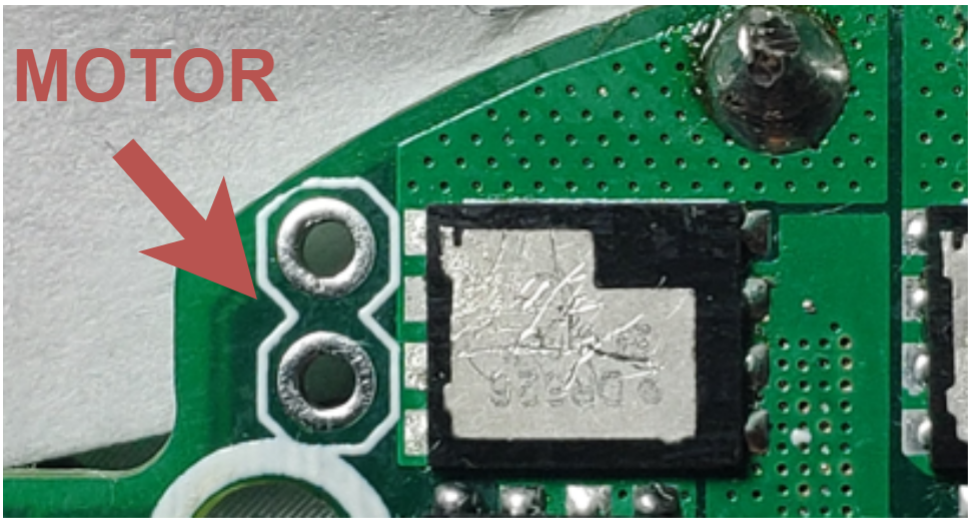
Warning

Always make sure CAN bus lines are not shorted to the positive power rail. Applying supply voltage to these pins will cause permanent damage to the controller!

All MD80 versions have the capability to measure the MOSFET and motor temperature. This is to ensure the safe operation of the driver and motor. The motor shutdown temperature is configurable up to 140°C max with a hysteresis of 20°C. The driver shutdown temperature is fixed at 100°C with a

Skip to main content

In the case of the MD80 HW V2.0 the MOSFET thermistor is built-in directly under the power stage and only the motor thermistor connector is available:



The connectors in the case of the HW1.1 and HV1.3

Note

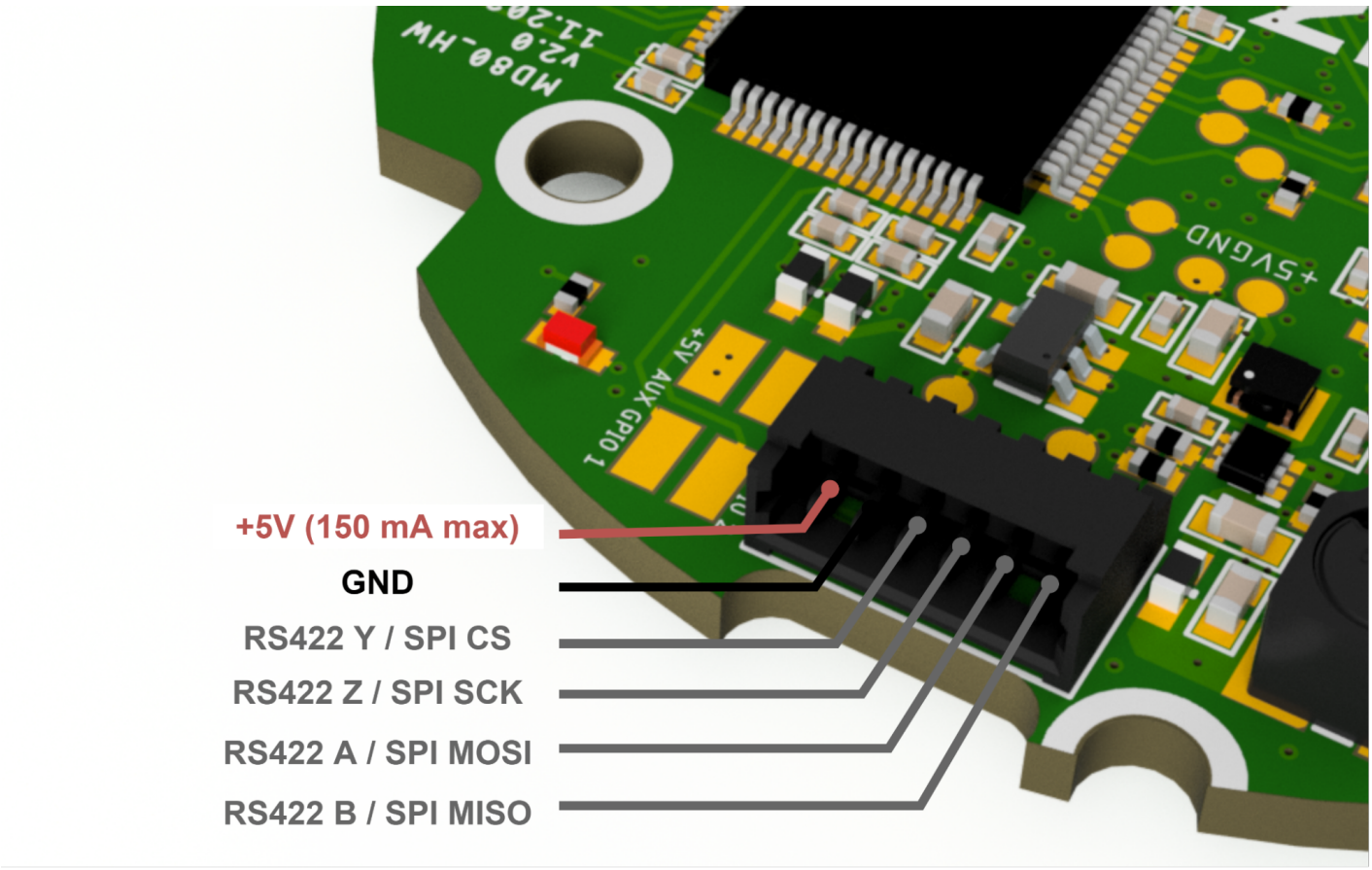
For the current version of the board we recommend using NTCMHP10K thermistors. Using other thermistors may result in imprecise temperature readout.

Note

On older (<2.2) designs NTCMHP100K is recommended.

Since version HW V2.0 the PCB is equipped with an auxiliary connector (picoblade series connector 53048-0650 compatible with 797580006 and 510210600) for communication with output encoders. The connector pinout is available below:

Skip to main content



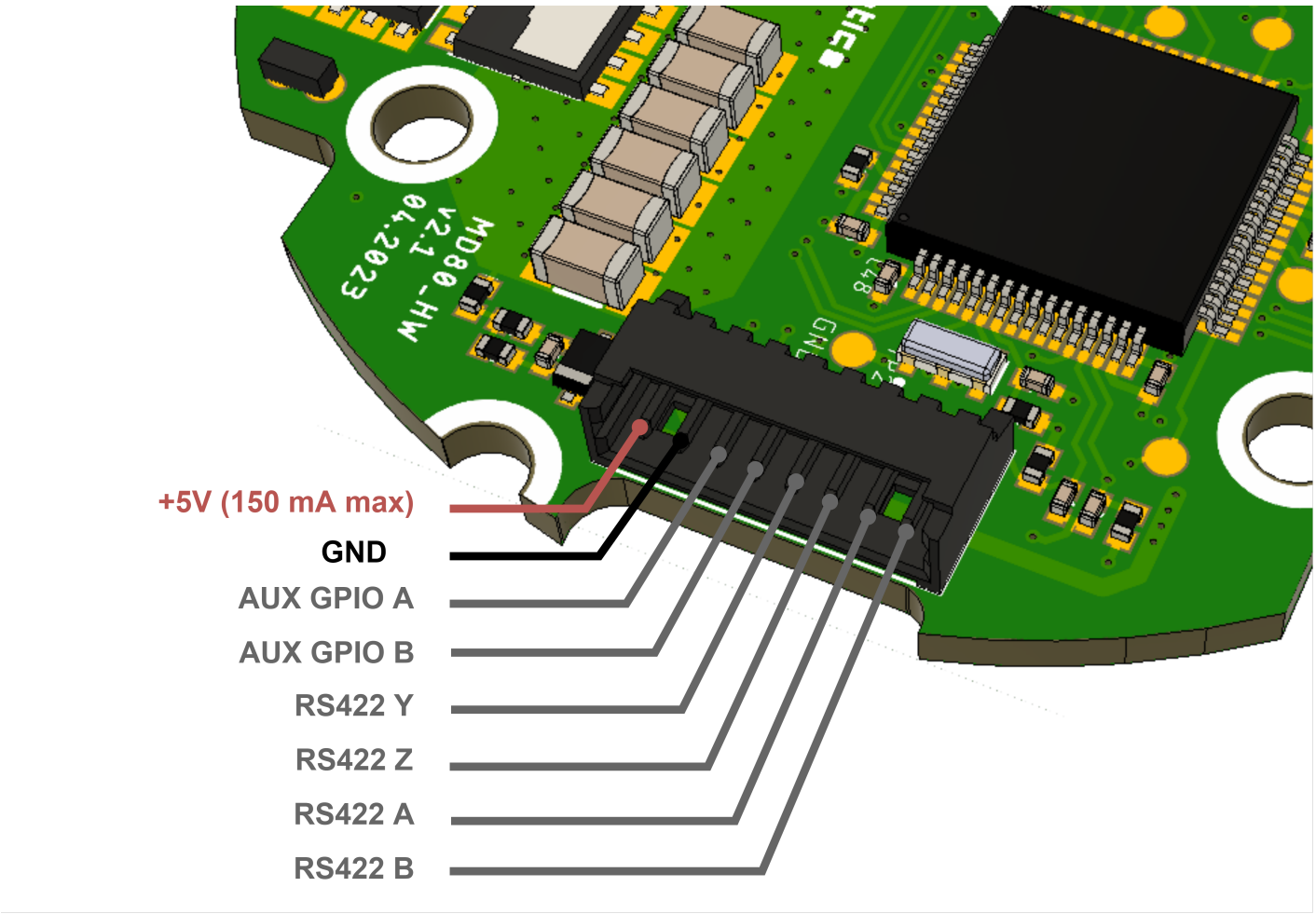
External connector pin functions (RS422 / SPI) are selectable using resistors on the bottom PCB side.

> **Warning**
>
> The auxiliary connector pins are 3.3V tolerant. Applying 5V to these pins will cause permanent damage to the controller!

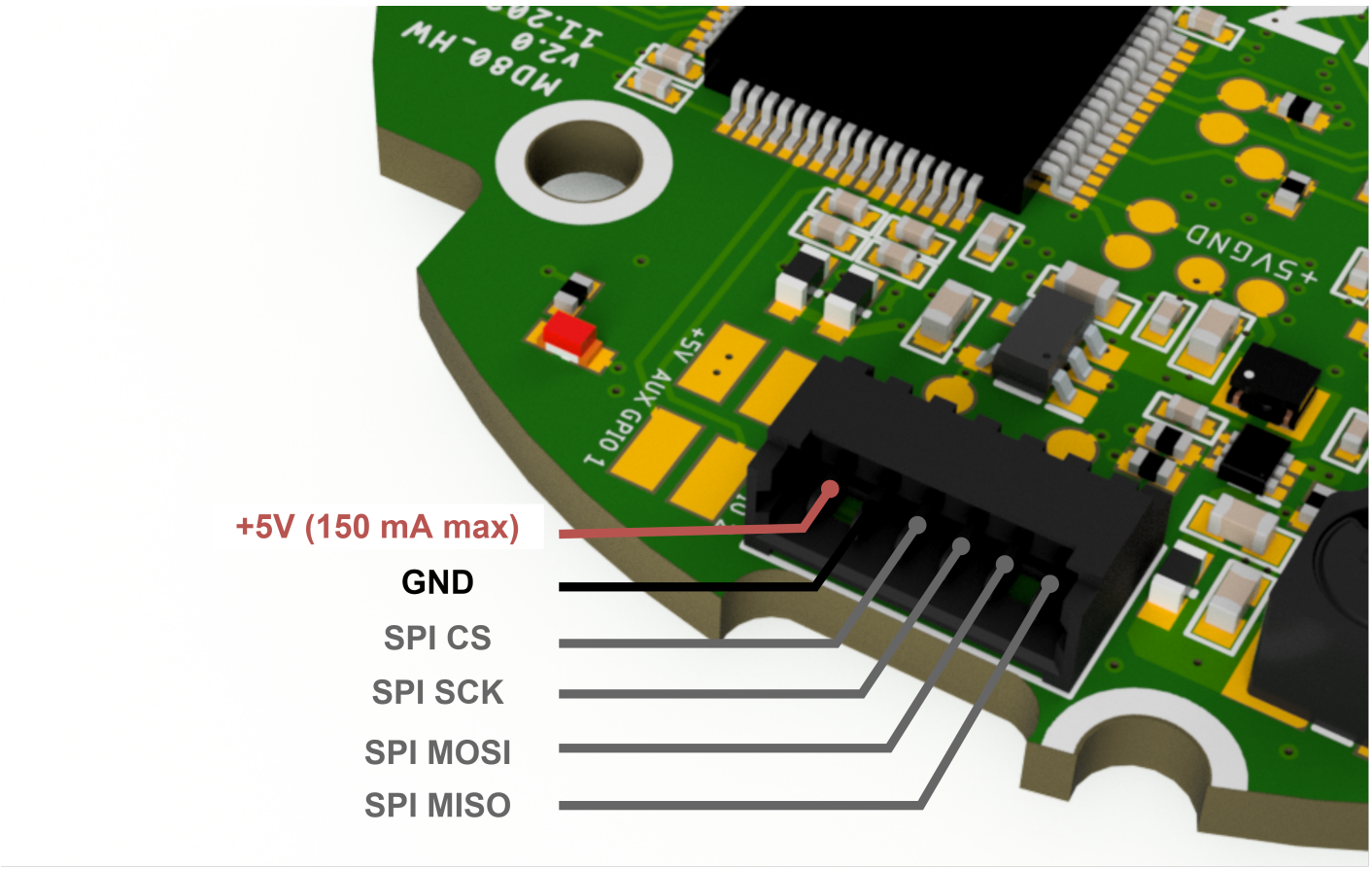
The HW2.1 features a separate RS422 connector. Thus the AUX1 connector features only the SPI singals, while AUX2 is dedicated for the RS422 communication bus.

AUX GPIO A pin can be used as an automatic brake control output signal. This signal when amplified, can be used to control MAB SLIM electromagnetic brakes. Enabling of this mode is done via `userGpioConfiguration` register. When enabled driver will automatically disengage the brake when enabled.

Skip to main content



Skip to main content



We are able to integrate custom functions such as GPIOs for external sensors and indicators. For more information please contact us: contact@mabrobotics.pl

MD20

The MD20 is a compact brushless motor controller, ideal for small, precise gimbal motors. It offers multiple control modes, including position PID, velocity PID, impedance, raw torque, profile position, and profile velocity. The MD20 also supports easy daisy-chaining, allowing seamless integration of multiple controllers in a single control network.

Skip to main content

General parameters

Parameter	Value
Nominal Input Voltage Range	24 - 42 VDC
Maximum Input Voltage Range	10 - 48 VDC
Max Input Current	3.1 A
Max Continuous Phase Current w/o cooling	4.5 A
Max Peak Phase Current (t = 2 s)	20 A
Built-in software-controlled termination resistor	optional
FDCAN Baudrate (adjustable)	1/2/5/8 Mbps
Position PID Controller Execution Frequency	1 kHz
Velocity PID Controller Execution Frequency	5 kHz
Impedance Controller Execution Frequency	40 kHz
Torque Control Execution Frequency	40 kHz
Torque Bandwidth (adjustable)	50 Hz - 2.5 kHz
External encoder connector (SPI, RS422)	yes
External 5V power supply max current	150 mA

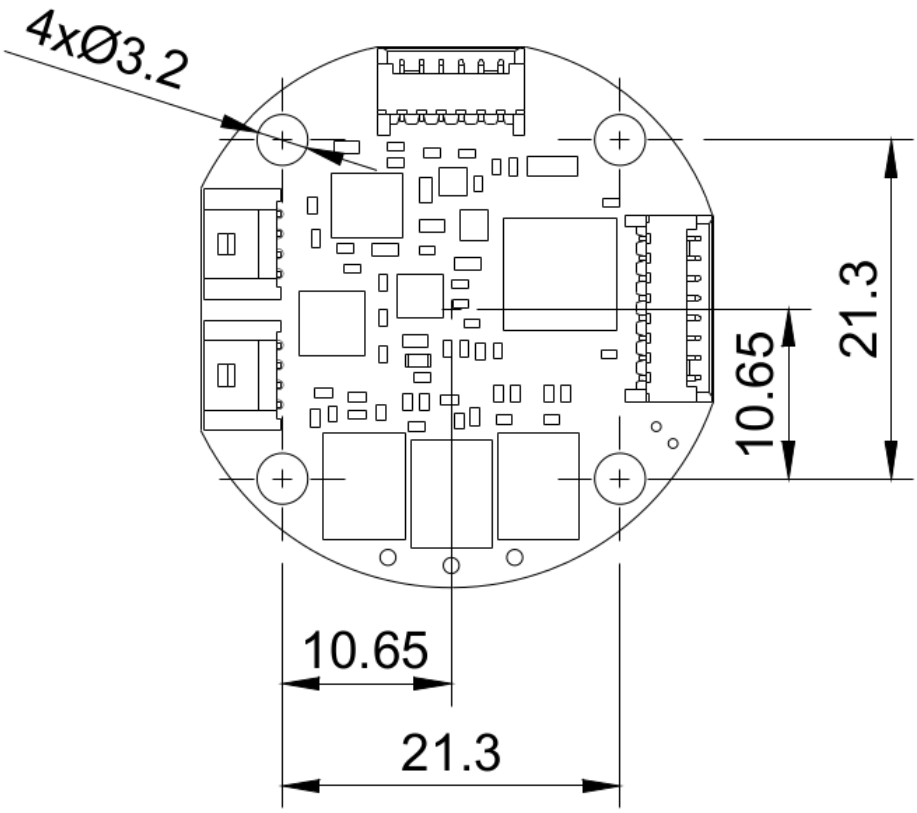
Skip to main content

Important

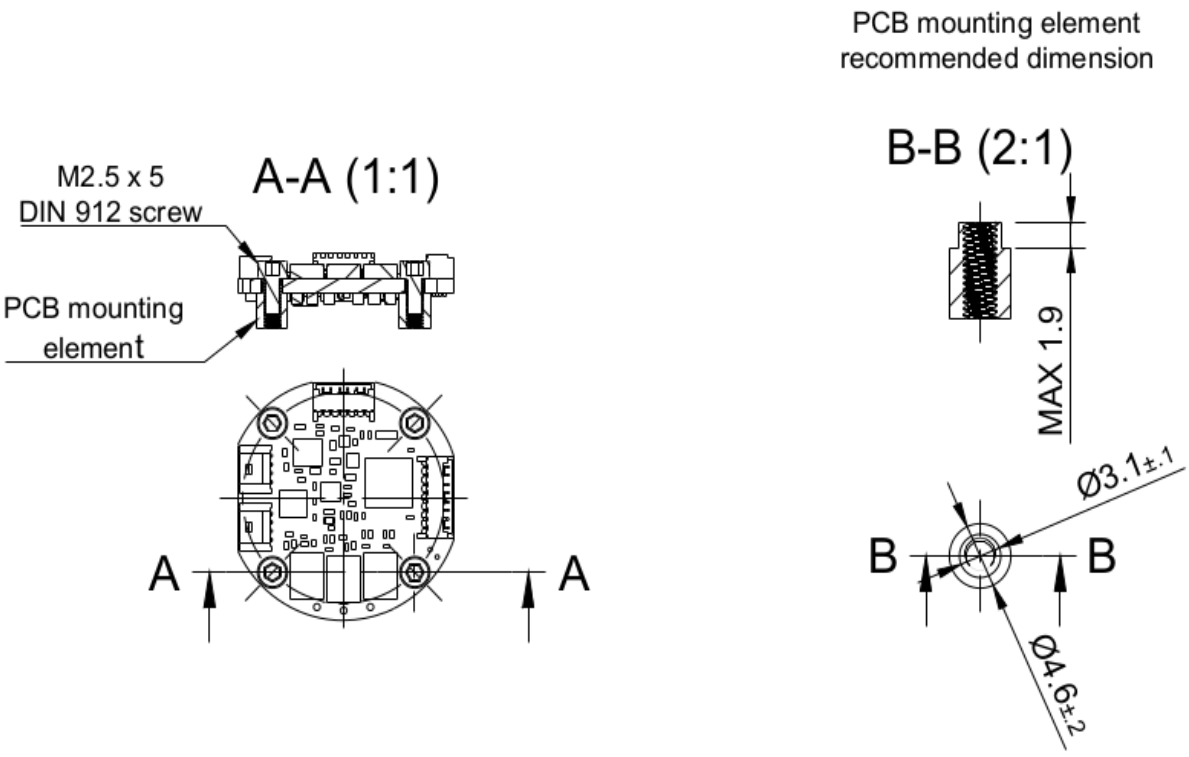
The MD20 is equipped with a custom shunt resistor suited for more precise current sensing. As a result all the configuration files (.cfg) dedicated for MD20 **MUST** contain `[hardware]` section with `shunt resistance` parameter equal to 0.004, ie.:

```
[hardware]
shunt resistance = 0.004
```

Mechanical data



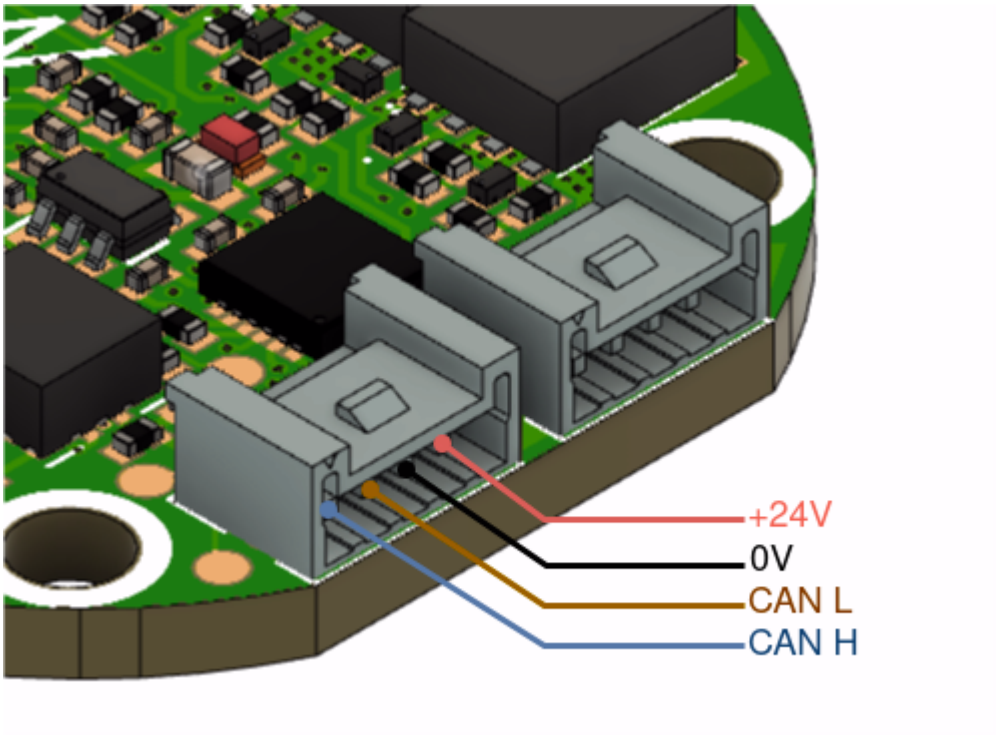
Skip to main content



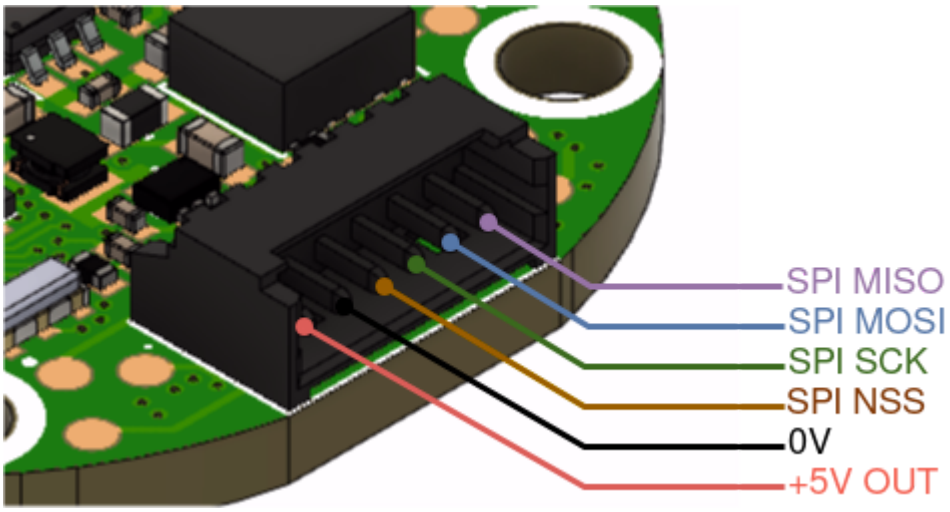
Connectors pinout

For power supply and CAN bus connection two *2200980471 Molex Micro-lock* connectors are available. They act as an internal bridge for easy chaining of the drives.

Skip to main content

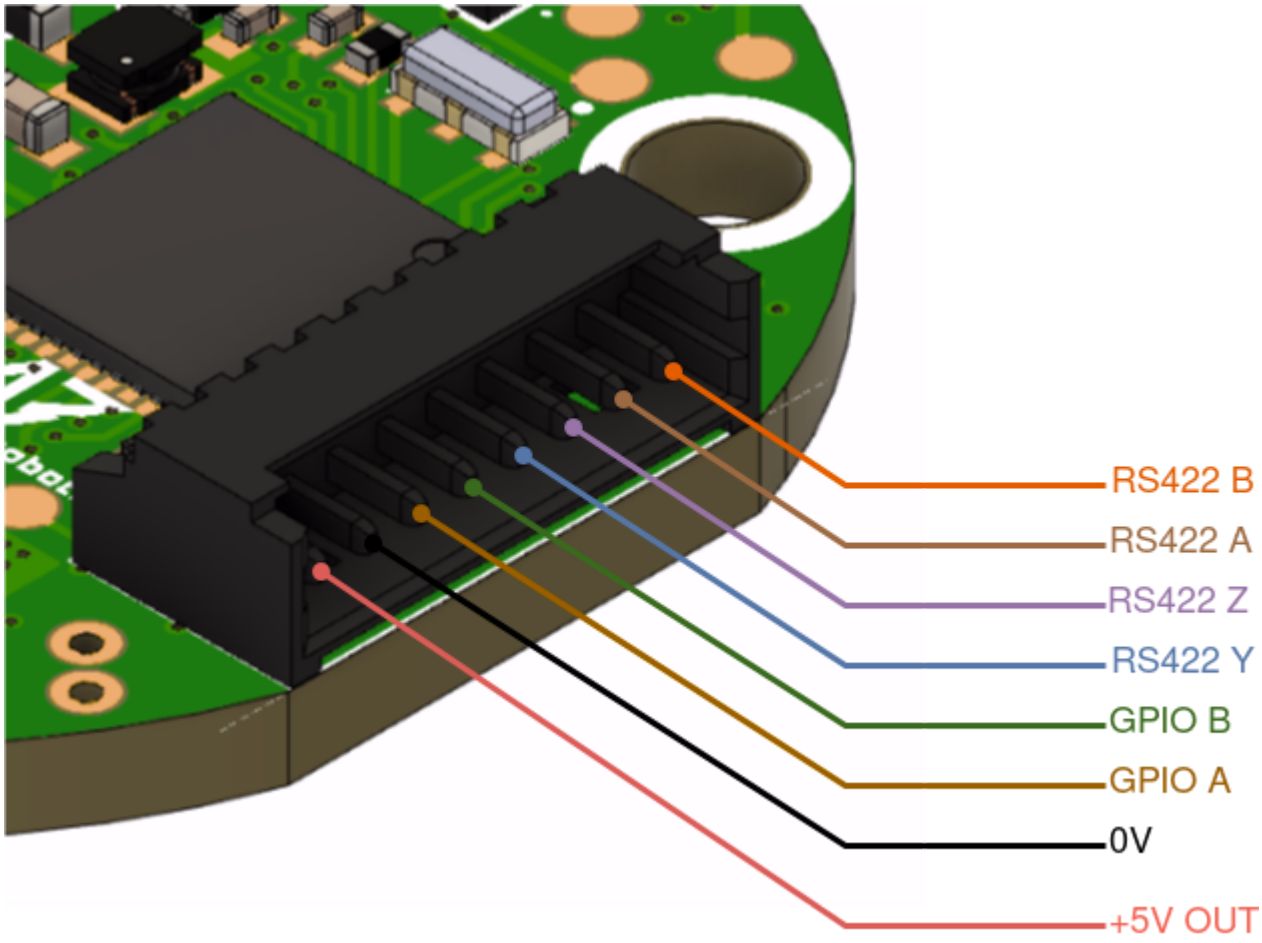


For external encoder connection via the SPI *530480650 Molex PicoBlade* connectors is provided.

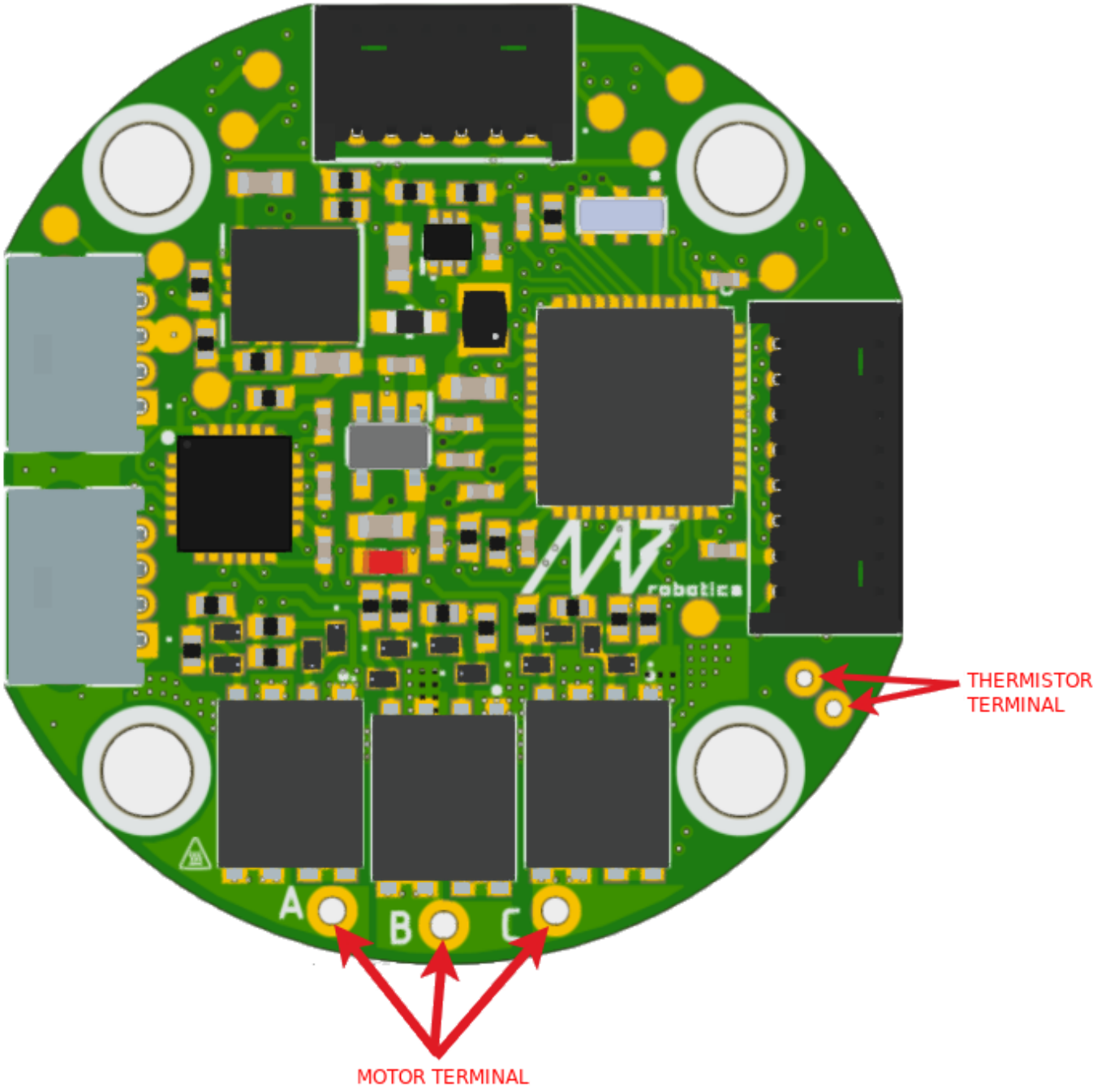


For external encoder connection via the RS422 and the GPIO pins utilization *530480810 Molex PicoBlade* connectors could be used.

Skip to main content



Skip to main content



Note

For the current version of the board we recommend using NTCMHP10K thermistors. Using other thermistors may result in imprecise temperature readout.

Skip to main content

Thermal specification

Thermal performance and efficiency

MD20 is a tiny but high-power, highly integrated motor controller. It creates challenges related to heat management and thermal performance. Taking this in mind, the design of the PCB is 6 layers to achieve excellent performance even without any cooling. The used components include small and efficient power converters and MOSFETs with best-in-class FOM(figure of merit).

The motor controller is only a part of the actuator, so we have done tests in the worst scenarios in a real setup, to provide valuable and relevant information about performance in the assembled actuator, instead of the idealized or simulated one.

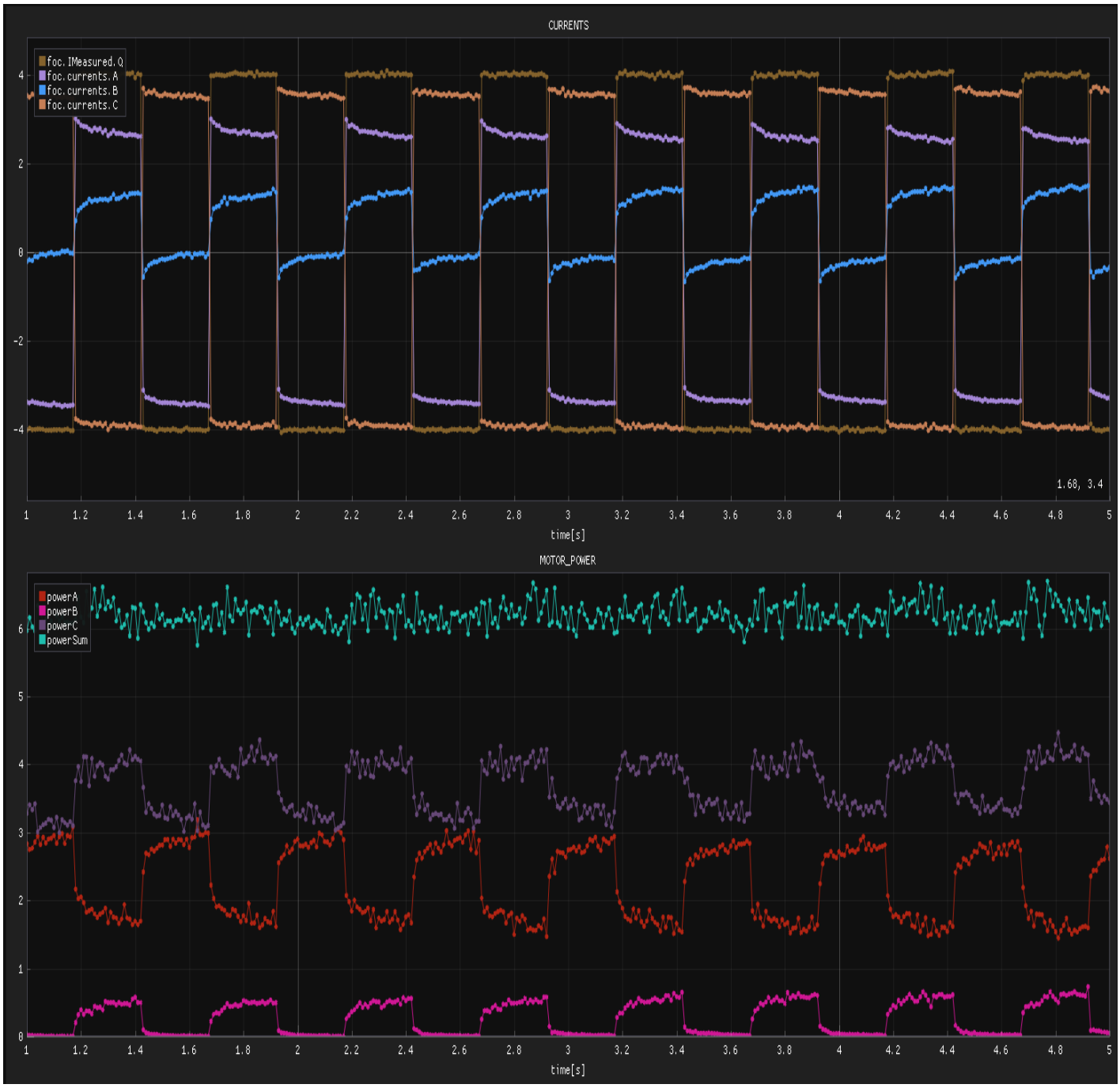
Power losses and system efficiency

To understand the charts below, we need to distinguish between mechanical power efficiency and electrical power conversion efficiency.

Mechanical power is the torque exerted by the motor ($\boldsymbol{\tau}$) multiplied by the angular velocity ($\boldsymbol{\omega}$) of the shaft. If the shaft is locked in place, the mechanical power will be zero, and thus, the mechanical power efficiency will also be zero in that scenario.

Electrical power conversion can be explained as the ratio between the power delivered to the electric motor and the total electric power consumed by the system. In this case, even though the motor’s mechanical power is zero, the efficiency of the driver can still be greater than zero due to the conversion of power to heat within the motor.

The scenario where mechanical power is zero is the worst-case scenario for electrical power conversion efficiency. In this case, all of the energy gets converted into heat.



As can be seen in the figure, the measured IQ is equal to the Ic current because the motor shaft was blocked.

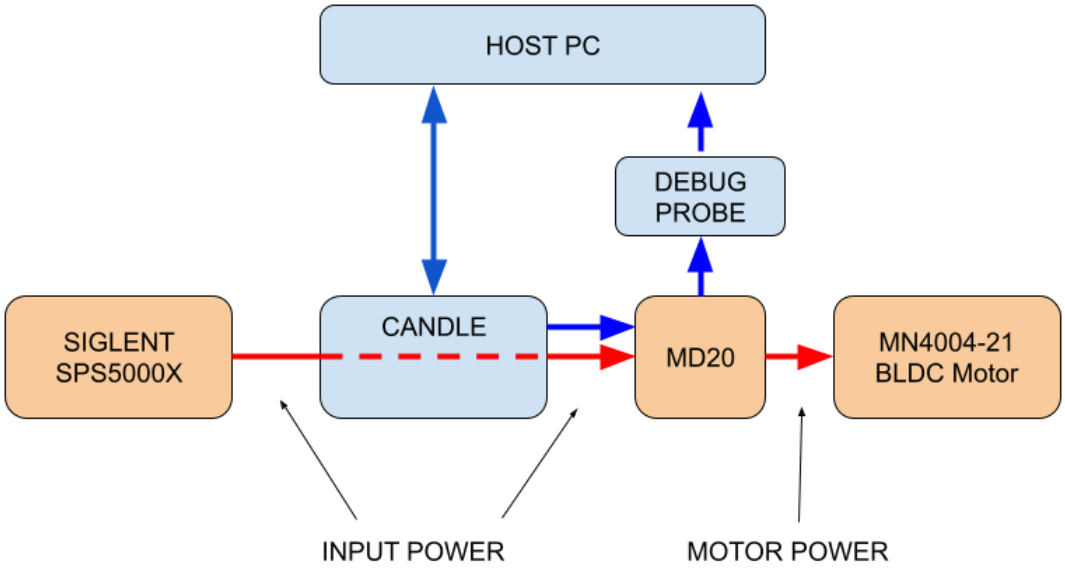
Ia, Ib, and Ic are motor phase currents, and Iq is the virtual quantity, which can be understood as the current in an old-fashioned brushed motor. Two quantities are important to analyze the inverter efficiency and losses: input power to the motor controller and the energy transferred from the inverter to the motor windings. Those quantities are the base that tells almost everything about the influence of the motor controller on the actuator system performance. There is only one exception connected to

Skip to main content

can increase the torque generated from the motor phase current by reducing phase currents THD, but also increase switching losses. Detailed analysis of these effects is out of the scope of the documentation, as it is motor-dependent on the system level. The gearbox/torque transfer efficiency as well.

Returning to the inverter, various losses are mainly related to transistor conduction and switching. Keeping as the rule to avoid theory and mathematical models, for the MD20 motor controller, which has a switching frequency of 40 kHz, at 24VDC supply total losses were measured the way described below.

The motor driver was supplied by the SIGLENT SPS5000X Digital power supply. Measurements of input Current, Voltage, and Power consumed by the whole setup were taken from this device. The power of the motor itself was calculated using discrete data of currents and voltages in each phase measured by the MD20 in each FOC cycle to ensure sampling at the same frequency as the control loop.



at various currents are collected in the table below:

Skip to main content

Input current[A]	Input power[W]	Respective peak phase current [A]	Output power[W]	Losses[W]	Efficiency[%]
0,05	1,25	1	0,63	0,62	50,40%
0,10	2,34	2	1,92	0,42	82,05%
0,17	4,12	3	3,79	0,33	91,99%
0,29	7,06	4	6,58	0,48	93,20%
0,47	11,35	5	10,6	0,75	93,39%
0,70	16,87	6	15,6	1,27	92,47%
1,05	25,17	7	23,2	1,97	92,17%
1,46	35,1	8	32	3,1	91,17%

During the analysis the idle power consumption of the board came out to be about 0,5W.

Because the input data for the table was taken when all parameters became stable (due to the temperature interia) we were not able to measure phase currents over 8A because the motor

was overheated.

Input power was measured by the digital power source Siglent SPS5000X. The output power was computed based on the phase voltages and currents measured with the oscilloscope. Losses were computed as the difference between input and output power.

The table above shows the losses. The tests described below present how the MD20 handles it.

Thermal tests

To cover all potential cases, three cooling scenarios have been performed:

- No additional cooling
- Passive cooling
- Active air cooling

In each test, the **MD20** motor driver was coupled with the MN4004-21 BLDC Motor.

MN4004-21 Motor parameters:

Test Item	KV400	Weight (Incl. Cable)	53g
Motor Dimensions	Φ44.35×19mm	Internal Resistance	359mΩ
Lead	50mm	Configuration	18N24P
Stator Diameter	40mm	Rated Voltage(Lipo)	4-6S
Idle Current(22V)	0.2A	Peak Current(180s)	12A
Max. Power(180s)	300W		

To maximize the convection effect and thus simulate the “worst case” of the setup position, the motor was mounted horizontally, under the MD20 so the heat dissipated by the motor additionally heats the driver PCB. We did our best to provide usable and relevant data about the performance.

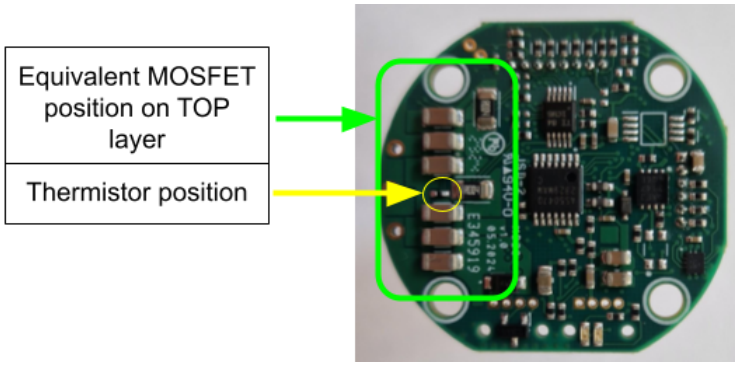
To ensure constant DC current in the motor phases, its shaft was blocked and the control loop was tweaked so that the current was equally distributed to each phase of the motor.

In each scenario, the tests were repeated for various cases of phase currents. from 0,5A to 5A with the step of 0,5A, and from 5A to the maximum current (that does not overheat the driver) with the step of 1A. For each current, the setup was tested until the temperature reached a stable value.

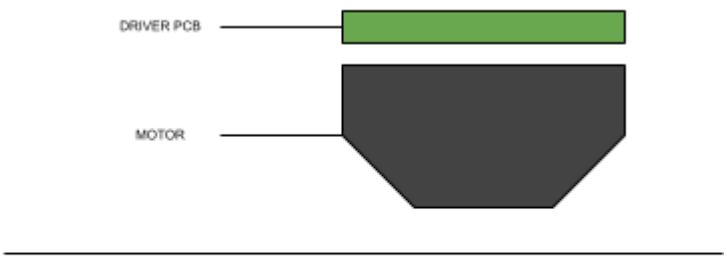
All tests were performed in constant environment conditions, in still air at 25 C ambient temperature.

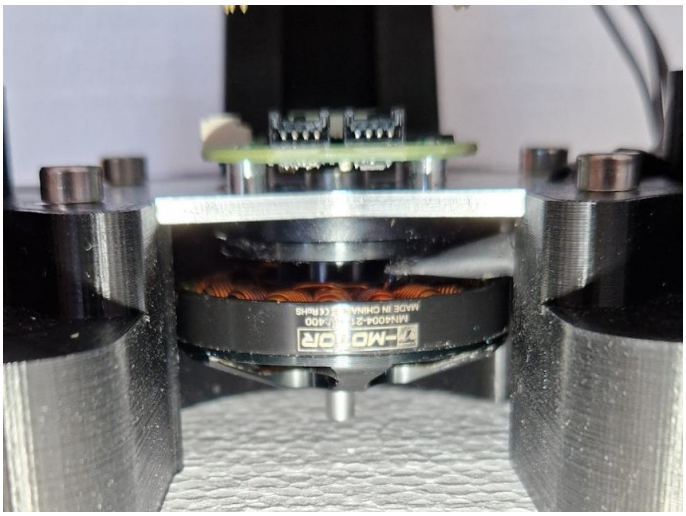
Before each particular test, the device was powered on and preheated itself to the idle temperature of about 39°C.

MD20 Temperature was measured with the usage of an onboard thermistor placed on the bottom layer of the PCB, directly below the MOSFETs.



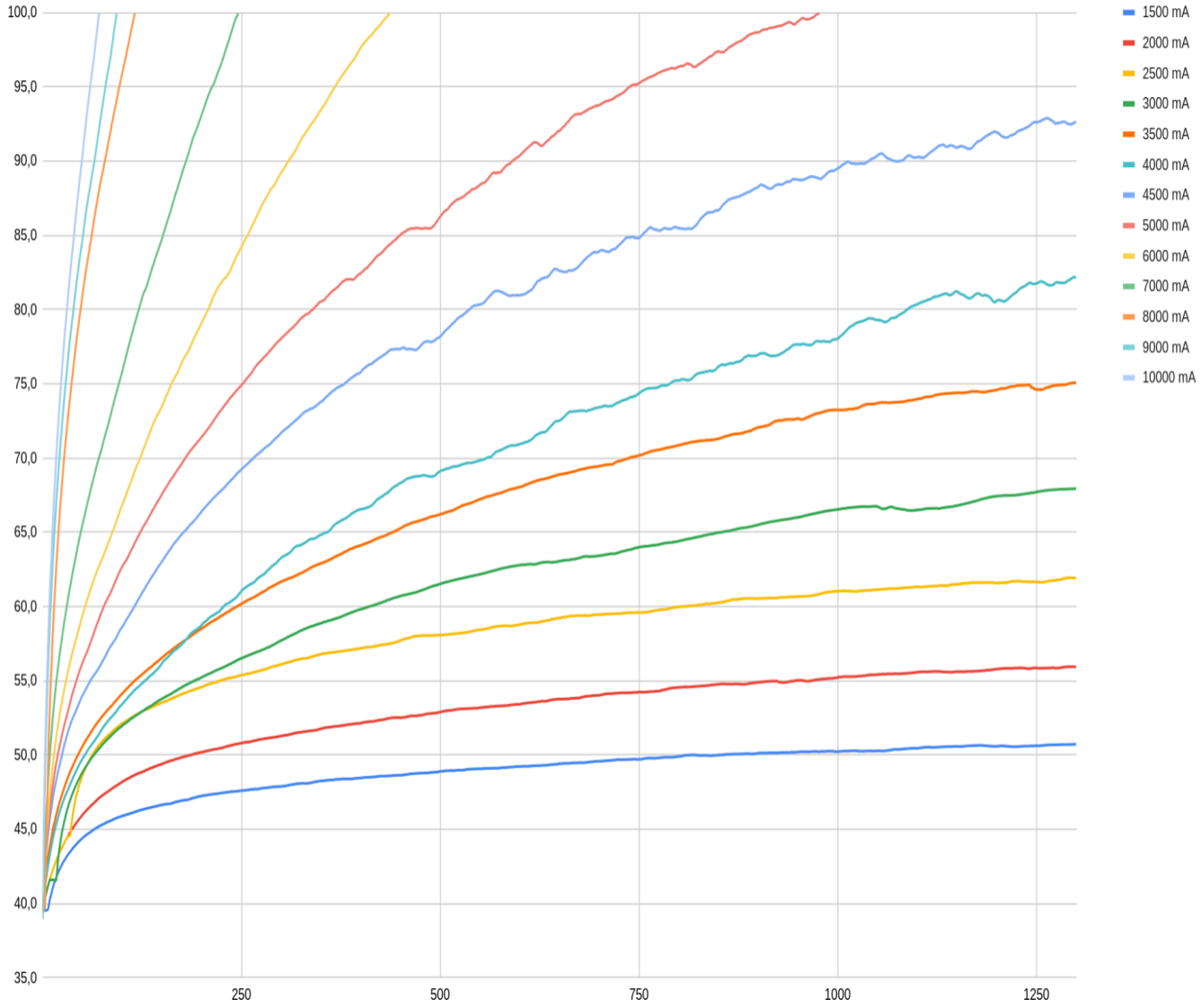
Skip to main content





Temperature vs Time characteristic:

Skip to main content

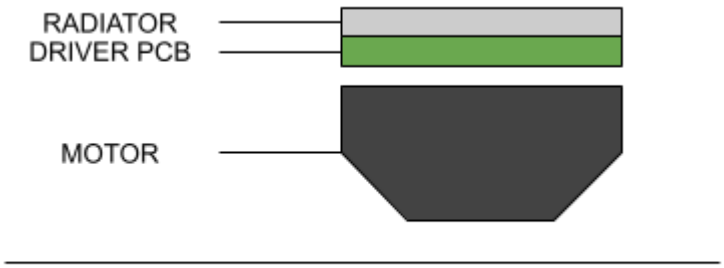


Summary table:

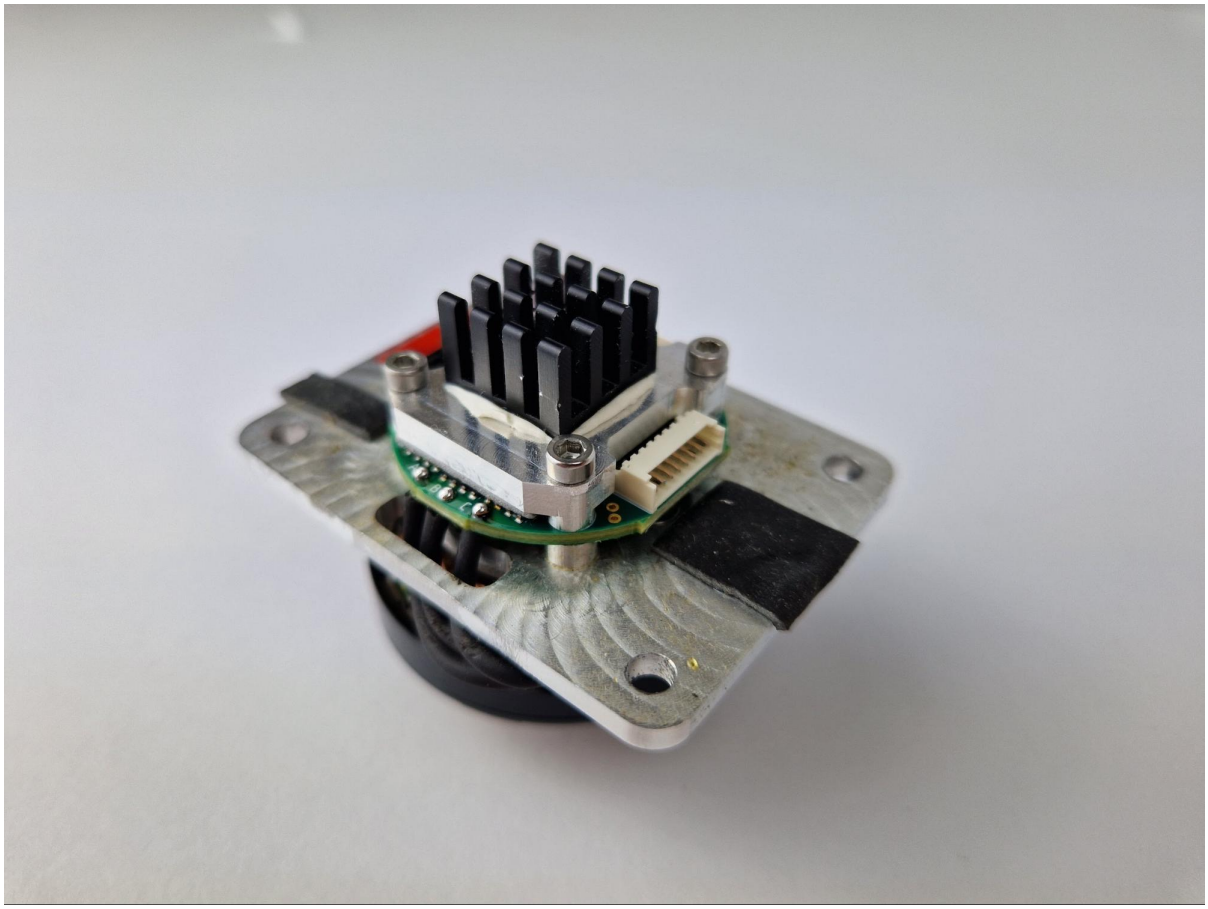
Skip to main content

No cooling thermal test summary		
Maximum continuous current	**4,5A**	
Current	**Stable temperature**	**Time to reach**
1A	46°C	~ 1250 s
1,5A	51°C	
2A	56°C	
2,5A	62°C	
3A	68°C	
3,5A	78°C	
4A	83°C	
4,5A	92°C	
5A	100°C	~ 980 s
6A	100°C	~ 440 s
7A	100°C	~ 250 s
8A	100°C	~ 120 s
9A	100°C	~ 95 s
10A	100°C	~ 70 s

Test scenario II - passive cooling.

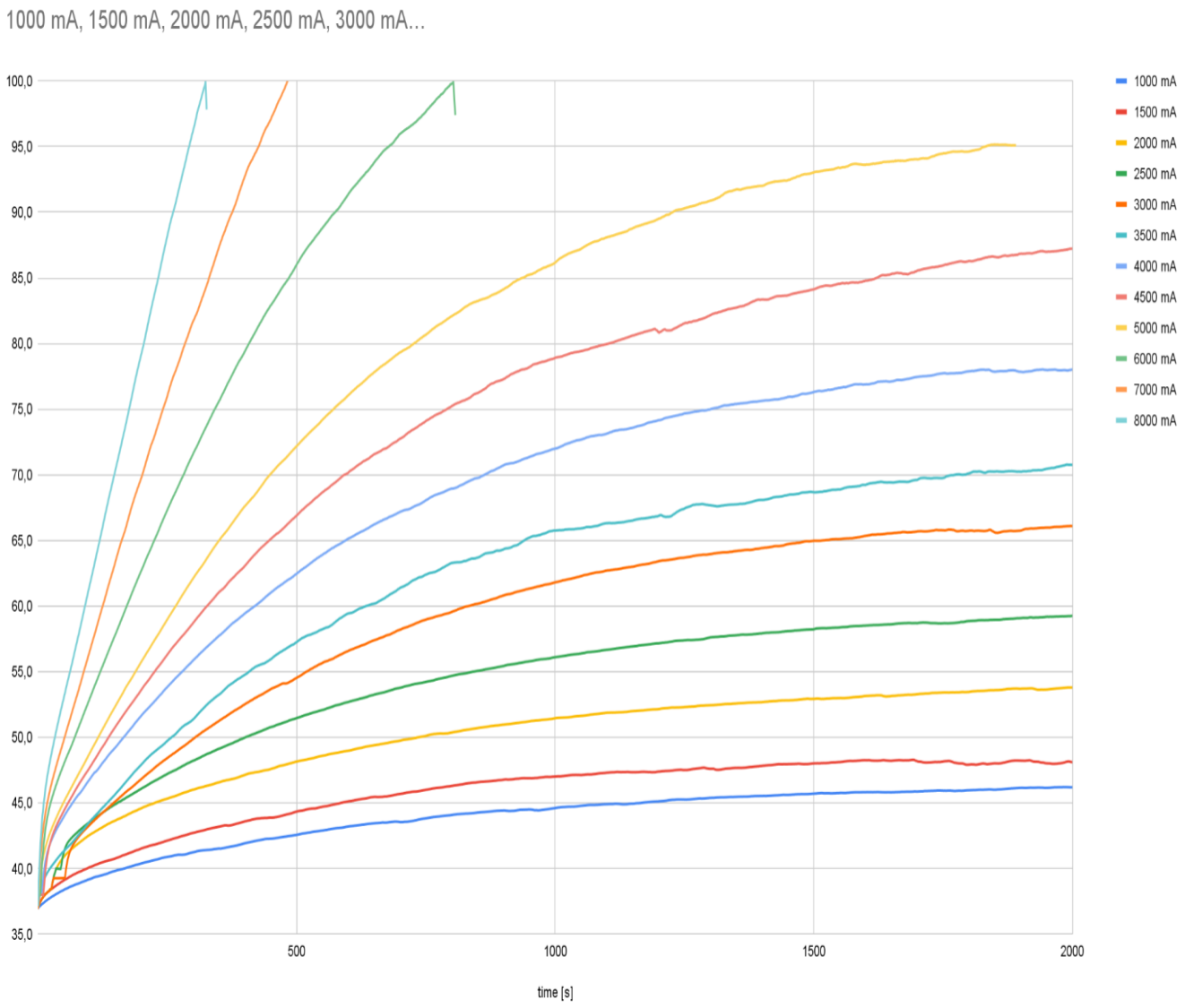


Skip to main content



Temperature vs Time characteristic:

Skip to main content



Summary table:

Skip to main content

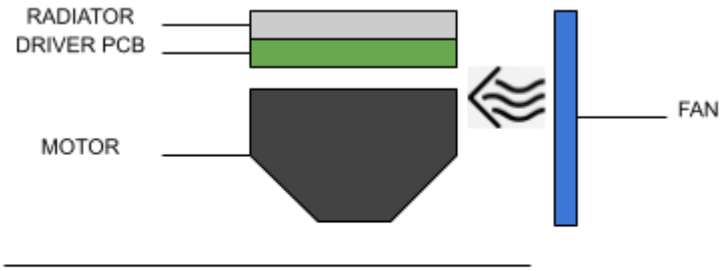
Passive cooling thermal test summary		
Maximum continuous current	**5A**	
Current	**Stable temperature**	**Time to reach**
1 A	46,5°C	~ 2100 s
1,5A	48,2°C	~ 2100 s
2A	53.8°C	~ 2000 s
2,5A	59,5°C	~ 2000 s
3A	66,3 °C	~ 2000 s
3,5A	72,1°C	~ 2800 s
4A	79,7°C	~ 2600 s
4,5A	88,9°C	~ 2600 s
5A	95,1°C	~ 1850 s
6A	100°C	~ 800 s
7A	100°C	~ 480 s
8A	100°C	~ 325 s

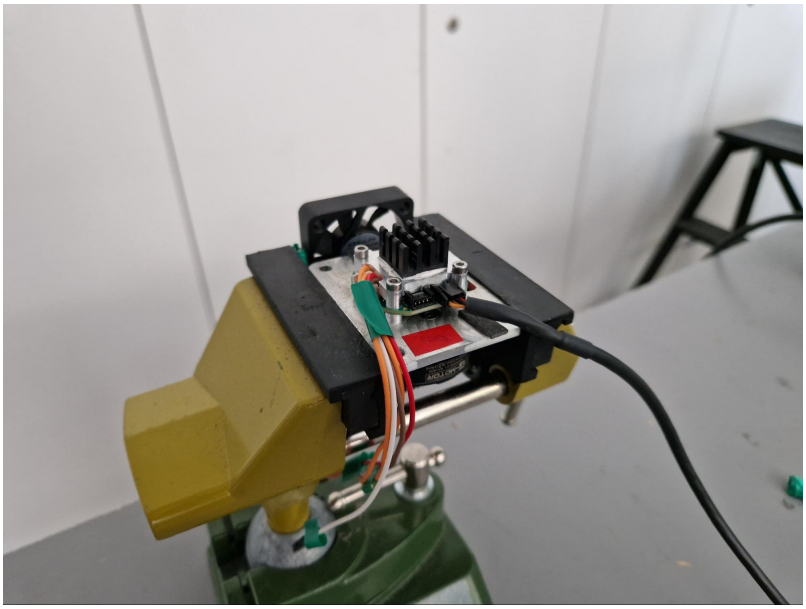
Note

During the tests in this scenario, the motor windings were getting warmer much faster than the MD20 PCB so the tests were stopped after 8A to avoid damaging the motor. The motor started to produce smoke before the motor controller reached 100 degrees C, so the test was finished on the 8A.

Test scenario III - active cooling

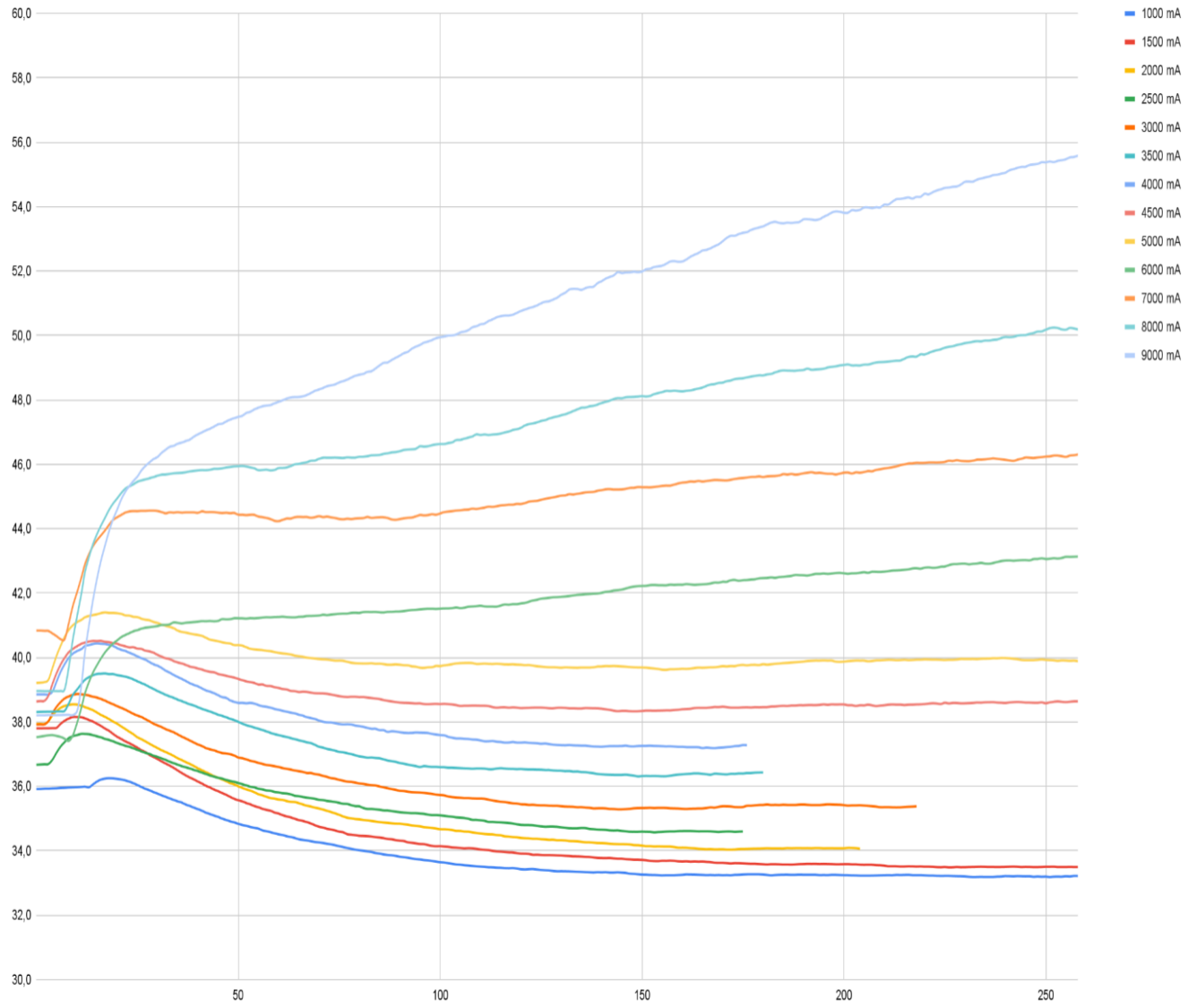
Skip to main content



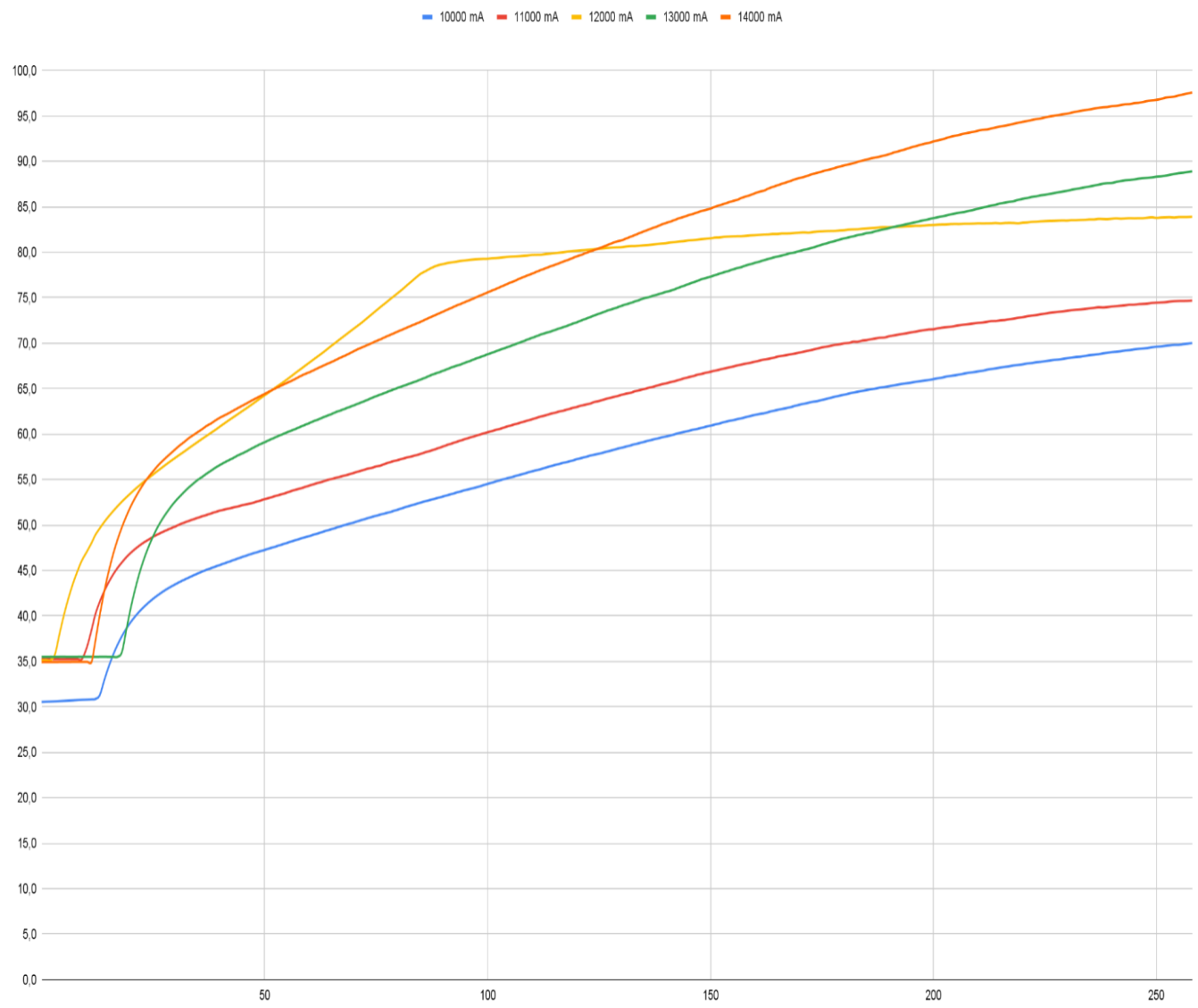


Temperature vs Time characteristic (note that this characteristic was splitted into two current ranges):

Skip to main content



Skip to main content



Summary table:

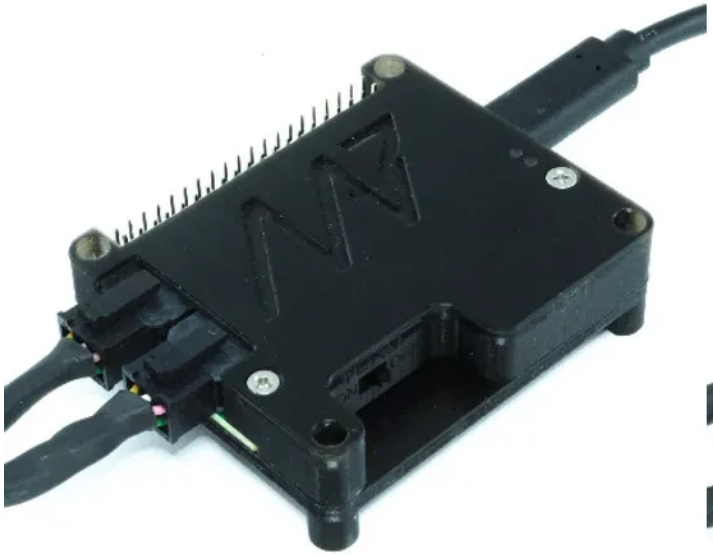
Skip to main content

Active cooling thermal test summary		
Maximum continuous current	**13A**	
Current	**Stable temperature**	**Time to reach**
1 A	33,2°C	~ 170 s
1,5 A	33,6°C	~ 170 s
2 A	34°C	~ 180 s
2,5 A	34,6°C	~ 180 s
3 A	35,4 °C	~ 180 s
3,5 A	36,4 °C	~ 180 s
4 A	37,3 °C	~ 180 s
4,5 A	38,5 °C	~ 200 s
5 A	40°C	~ 230 s
6 A	43,5°C	~ 400 s
7 A	47,5°C	~ 490 s
8 A	51,5°C	~ 580 s
9 A	58,5°C	~ 630 s
10 A	73°C	~ 430 s
11 A	78°C	~ 420 s
12 A	84,9°C	~ 540 s
13 A	94,9°C	~ 540 s
14 A	100°C	~ 280 s

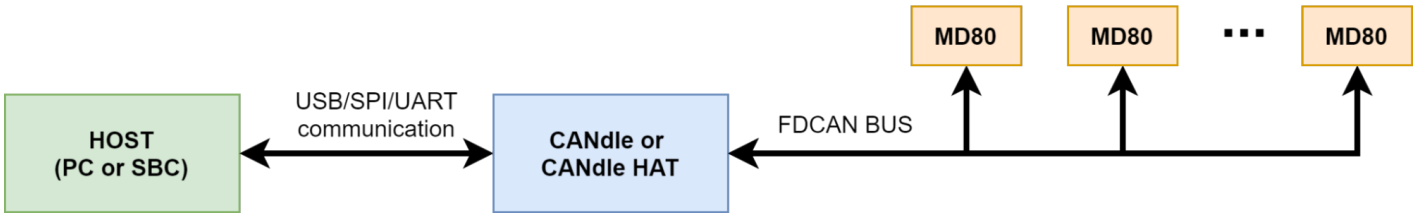
Skip to main content

CANdle and CANdle HAT

CANdle is a converter used to communicate between MD controllers and the host device. Currently, there are two CANdle versions - CANdle and CANdle HAT.



The first one is a simple version that uses only the USB bus to communicate with the host, whereas the latter can communicate using USB, SPI, and UART bus, and is easy to integrate with SBCs such as Raspberry PI. The communication with MD controllers is performed using FDCAN bus.



To achieve the fastest communication speeds you should aim for the USB bus. For more details on the latency topic please check out the latency section.

Skip to main content

Important

Currently CANdle supports only Linux operating systems.

Principle of operation

CANdle can work in two different modes: CONFIG and UPDATE. When in CONFIG mode, it works as a traditional translator device between two selected buses - USB/SPI/UART and FDCAN. This mode is used to set up the drives and prepare them for a low-latency operation in the UPDATE mode. When the configuration is done the user calls `candle.begin()` which starts a low-latency continuous connection with the MD controllers. In the UPDATE mode, you are not allowed to call the config functions. To make them easier to recognize, each config function starts with a config keyword. The user exits the UPDATE mode using `candle.end()` method.

When in Update mode the communication speed is dictated by the number of drives attached to the bus. Please see the latency section for maximum communication speeds.

Generally, a program using CANdle should follow the workflow below:

Creating a Candle object creates a class that will hold all the data and provides an API for the user. During the creation of the class, the software will attach itself to the ttyACMx port used by the CANdle, or SPI/UART bus if CANdle HAT is considered. It will also perform a reset operation on the device and set up basic parameters. This ensures that the device is in the known state at the start of the program. When an object is created, the CANdle is in the CONFIG state.

Now the configuration of the drives can be done. As a rule of thumb, all class methods starting with the word ‘config’ can be used here. They do not require adding MDxx to the update list, just require an ID of the drive to talk to. This is a good place to set current limits, change FDCAN parameters, or save data to flash.

Skip to main content

> **Note**
>
> This is also a good place to call `Candle::ping()`, this will trigger the CANdle device to send an FDCAN frame to all valid FDCAN IDs. The method will return a vector of all IDs that have responded. This can be used to check if all the drives have power and if all communication is set up correctly.

The next step is adding MDs to the update list. To do so, use Candle::addMd80() method, with an FDCAN ID (drive ID) as an argument. This will trigger CANdle to quickly check if the drive is available on the bus at the ID, and if it is, the CANdle device will add the drive to its internal list and send an acknowledgment to the CANdle lib. If the drive is successfully added the addMd80() method will add this particular MDxx to its internal vector for future use and return true.

When all drives have been added, the drives should be ready to move. This can be done with methods starting with the “control(…)” keyword. Firstly the control mode should be set, then the zero position set (if desired), and finally the drives can be enabled by using `Candle::controlMd80Enable()` method.

> **Note**
>
> Sending an ENABLE frame will start the CAN Watchdog Timer. If no commands follow, the drive will shut itself down.

When all drives are enabled, `Candle::begin()` can be called. This will set the CANdle (both device and library) to UPDATE state. The device will immediately start sending command frames to the MDs. From now on the library will no longer accept config* methods. Right now it is up to the user to decide what to do. After the first 10 milliseconds, the whole MDxx vector will be updated with the most recent data from MDs and the control code can be executed to start moving the drives. Individual drives can be accessed via Candle::md80s vector. The vector holds instances of ‘Md80’ class, with methods giving access to every MD series motor controller control mode. Latest data from md80’s responses can be accessed with `Md80::getPosition()`, `Md80::getVelocity()`, `Md80::getTorque()`, `Md80::getErrorVector()`.

Skip to main content

> **Note**
>
> As the communication is done in the background, it is up to the user to take care of the software timing here. If you for example set a position command, but don't put any delay after it, the program will get to an end, disabling the communication and the servo drives, without you seeing any movement!

When the control code finishes, the `Candle::end()` method should be called. This will ensure a 'clean exit' meaning a properly closed communication on both USB and FDCAN side. `Candle::begin()` can be called later to resume communication if needed.

USB bus

The USB bus is the most common one, used in both CANdle and CANdle HAT. This is one of the slowest communication bus when it comes to performance, due to the non-realtime nature of the host, however, it's the easiest one to set up and test. Since the USB communication interface is not well-suited for real-time applications due to random host delays the update rates are dependent on your master controller device.

> **Hint**
>
> We highly recommend using the USB bus set up for the first run.

SPI bus

The SPI bus is only available on CANdle HAT devices. It's the fastest possible bus that can be used to communicate with the MD series motor controller controllers using CANdle HAT. Together with the RT-PATCH'ed kernel of the system, you will get the best performance.

> **Hint**
>
> CANdle HAT in SPI mode works with all FDCAN speeds, however, we advise setting it to 8M for the best performance.

Skip to main content

> **Note**
>
> Since it needs some additional configuration on Single Board Computers such as Raspberry PI, we recommend starting playing with it after getting accustomed to the ecosystem using the USB bus.

UART

The UART bus is only available on CANdle HAT devices. Its speed on Raspberry PI microcomputers with CANdle HAT is comparable to that of USB, so it should be only used as an emergency bus when the SPI and USB ports are not available.

> **Note**
>
> CANdle HAT in UART mode works with all FDCAN speeds, however, we advise setting it to 8M for the best performance.

Using CANdle and CANdle HAT

PC (USB bus)

The library does not require any additional software to be functional, It can work as-is. However, to make full use of it we recommend using setserial package (for increasing maximal access frequency to the serial port used for communication with CANdle). To install it please call:

```
sudo apt install setserial
```

To enable access to CANdle from userspace, the user should be added to dialout group by calling:

```
sudo usermod -a -G dialout <user>   # where <user> is current username
```

If this is not possible, devices access level can be granted by:

Skip to main content

```
sudo chmod 777 /dev/ttyACMx   # where x is CANdle port number, usually 0
```

If this is also not possible, programs that use CANdle (including examples), can be launched with sudo.

SBC (USB/SPI/UART)

Running CANdle or CANdle HAT using a USB bus on SBC is identical to running it on a Linux PC (section above). However, when using SPI or UART a few other requirements have to be met. We will guide you through the setup process on Raspberry PI 4.

> **Note**
>
> When using SBCs other than Raspberry the process may vary and should be performed according to the board manual or with the help of the manufacturer.

SPI

To enable the SPI bus you should call:

```
sudo nano /boot/config.txt
```

uncomment the following line, save the file

```
dtparam=spi=on
```

and reboot:

```
sudo reboot
```

to make sure SPI is enabled call:

```
ls /dev | grep spi
```

Skip to main content

you should see an output similar to this:

```
spidev0.0
spidev0.1
```

UART

To enable the UART bus you should call:

```
sudo nano /boot/config.txt
```

and add the following lines on the end of the file

```
enable_uart=1
dtoverlay=disable-bt
```

after that open the cmdline.txt

```
sudo nano /boot/cmdline.txt
```

and remove the part:

```
console=serial0,115200
```

and reboot:

```
sudo reboot
```

Latency

The latency was measured using the `mdtool test latency 8M` command. Since the CAN frames are synchronized with master device frames the update rate of the master is the same as MD. The

setup was tested on a PC using only USB bus (PC Ideapad Gaming 3 AMD Ryzen 7 4800H) and Raspberry PI 3b+ with RT PATCH (4.19.71-rt24-v7+) on USB, SPI, and UART bus.

USB CANdle/CANdle HAT

VM PC Ideapad Gaming 3 AMD Ryzen 7 4800H			
mdtool test latency 8M		mean [Hz]	stdev [Hz]
priority high	12 MD80s	245,1	4,25
	6 MD80s	424,1	5,32
	3 MD80s	643,65	34,67
	1 MD80	1064,55	93,02

RPI 3b+ RT PATCH (4,19,71-rt24-v7+)			
mdtool test latency 8M		mean [Hz]	stdev [Hz]
priority high	12 MD80s	135,6	1,26
	6 MD80s	230,9	2,23
	3 MD80s	362,6	0,52
	1 MD80	726,7	0,95

SPI CANdle HAT

RPI 3b+ RT PATCH (4,19,71-rt24-v7+)			
mdtool test latency 8M		mean [Hz]	stdev [Hz]
priority high	12 MD80s	346,3	0,48
	6 MD80s	651,3	0,48
	3 MD80s	1162,3	2,06
	1 MD80	2847,6	26,04

UART CANdle HAT

RPI 3b+ RT PATCH (4,19,71-rt24-v7+)			
mdtool test latency 8M		mean [Hz]	stdev [Hz]
priority high	12 MD80s	76,5	0,51
	6 MD80s	164,2	0,42
	3 MD80s	302,7	0,48
	1 MD80	696,5	0,71

High task priority was achieved using the following snippet in the mdtool test latency function:

```
struct sched_param sp;
memset(&sp, 0, sizeof(sp));
sp.sched_priority = 99;
sched_setscheduler(0, SCHED_FIFO, &sp);
```

To be able to change task priority be sure to call the test with `sudo`.

To change a running task priority use the snippet below. It can be useful when your program cannot be run directly with sudo - for example, when dealing with ROS nodes.

```
CONTROL_PID=$(sudo pidof -s <NAME_OF_YOUR_EXECUTABLE>)
CONTROL_PRIORITY=99
sudo chrt -f -p ${CONTROL_PRIORITY} ${CONTROL_PID}
```

During testing on Raspberry PI SBCs we have found out that isolating a CPU core (isolcpus) specifically for the CANdle process did not result in a performance increase - rather made it less performant.

Skip to main content

> **Note**
>
> When dealing with the MD x CANdle ecosystem for the first time we advise using the USB bus that is available on both CANdle and CANdle HAT devices.

Software Pack

The MD x CANdle software pack consists of a few modules. All of them are based on the main CANdle C++ library which takes care of the low-level communication and provides API for high-level software.

MDtool

MDtool is a console application for configuring and performing basic diagnostics on MD drives via CANdle. It is designed as a complementary tool for APIs, reducing the overhead when setting up the drives for the first time or reconfiguring them. It uses the CANdle C++ library on its backend.

Installation

The easiest way to install the MDtool is to select the appropriate *.deb package from the MDtool GitHub repo releases page (compliant with your system's architecture). To install after the download simply call:

```
sudo apt install ./mdtool_xx-x_xx.deb
```

After the install please make sure the current user is added to the dialout group using the command:

```
sudo adduser <current user> dialout
```

if it wasn't, please reboot the PC

```
sudo reboot
```

Skip to main content

It is also recommended to install the setserial package which allows for higher communication speeds:

```
sudo apt install setserial
```

Be sure to call

```
mdtool bus <SPI/UART/USB>
```

to configure MDtool for the desired communication bus before first use, if you're using CANdle HAT and SPI or UART bus.

In case the CANdle device still doesn't work, make sure a /dev/ttyACM0 device is listed when you call:

```
ls /dev/ttyACM*
```

> **Note**
>
> For the command prompt to work after the installation you have to restart the terminal window

Commands

`mdtool bus <bus> <device>`

MDtool is able to work with CANdle and CANdle HAT. This is why before the first use it has to be configured for a particular communication bus. Use the bus command to set it to USB, SPI, or UART, based on which device you own. The default bus setting is USB. You don't have to repeat this setting unless you want to change the current communication bus. The device parameter is optional and can be used in case of the UART and SPI bus, if the default device (/dev/spidev0.0 in case of SPI or /dev/ttyAMA0 in case of UART) is not suitable for your application.

`mdtool ping <baud>`

MDtool is able to discover the drives that are connected to the CAN bus. You can ping the drives at a specific speed (1M/2M/5M/8M) or just use the ‘all’ keyword for pinging all speeds in one go.

> **Note**
>
> CANdle does not support working with drives configured with different CAN speeds on the same CAN bus – please make sure when “mdtool ping all” command is executed, all discovered drives lie in a single speed category.

`mdtool config zero <ID>`

This command sets the current motor position to zero - from the moment this command is called all encoder measurements are referenced from the current position.

> **Note**
>
> This setting has to be saved to be preserved after power down! Please see the config save command.

`mdtool config can <current ID> <new ID> <baud> <watchdog period [ms]> <termination>`

This command is used to change MD parameters such as CAN ID, CAN speed, and CAN watchdog.

- CAN IDs should be in range <10:2000>
- Baud should be one of the available speeds (1M/2M/5M/8M)
- Watchdog period should be in range <1:2000> ms, 0 disables the watchdog. For more information on CAN watchdog please refer to section FDCAN Watchdog Timer.
- Termination should be either 1 to turn the termination on or 0 to turn the termination off.

> **Warning**
>
> Software-controlled termination is available since version HW V2.0. It is an optional setting -

Note

This setting has to be saved to be preserved after power down! Please see the mdtool config save command.

`mdtool config current <ID> <current>`

This command is used to set the maximum phase current that is allowed to flow through the motor when high torques are commanded. By default, the maximum current is set to a rather low value that will not lead to motor or driver burnout. However, this also limits the motor's maximum torque capabilities. Using the config current command one can increase the maximum current. For the absolute maximum please refer to maximum ratings section.

Warning

The warranty does not include burnout actions due to too high current settings. For max continuous driver current please refer to the general parameters and safety limits sections.

Note

This setting has to be saved to be preserved after power down! Please see the config save command.

`mdtool config bandwidth <ID> <torque bandwidth in Hz>`

This command can be used to change the torque bandwidth without recalibrating the actuator. For more information on the torque bandwidth please see the section about calibration.

Note

This setting has to be saved to be preserved after power down! Please see the config save command.

Skip to main content

For the config commands to take action after the next power cycle a save command has to be executed. This command saves the current drive's settings to the non-volatile FLASH memory.

`mdtool setup calibration <ID>`

This command runs the basic calibration routine. During calibration, the drive has to be able to rotate freely and the power supply should be able to deliver at least 1A of current. For more detail on the calibration process please refer to the calibration section.

`mdtool setup calibration_out <ID>`

This command runs the output encoder calibration routine. During output encoder calibration, the drive has to be able to rotate for at least two full rotations of the output shaft and the power supply should be able to deliver at least 1A of current. For more detail on the calibration process please refer to the output encoder calibration section.

`mdtool setup motor <ID> <*.cfg>`

This command is used to write a new motor config. For the config file argument one of the *.cfg files from ~/.config/mtool/mdtool_motors/ directory should be passed. Check out the description below for more information on the respective config fields:

```
[motor]
# motor name [20 characters max]
name = AK60 6
# pole pairs - the number of rotor magnets/2 if you are unsure just type zero - during calibration it will be autodetected
pole pairs = 0
# declated KV of the motor - its used when torque constant is set to zero
KV = 80
# motor torque constant Nm/A
torque constant = 0.119
# gear ratio -> example 6:1 reductor is 0.166666 whereas 1:2 multiplicator is 2
gear ratio = 0.166666
# maximum allowable phase current (can be also changed using "mdtool config current" command)
max current = 10
# torque bandwidth in Hz (please refer to the manual for more info)
torque bandwidth = 500
# motor shutdown temperature
shutdown temp = 80

[output encoder]
# output encoder type - valid types: AS5047 CENTER, AS5047 OFFAXIS
output encoder = 0
# output encoder mode - valid modes: STARTUP, MOTION, REPORT
output encoder mode = 0

[position PID]
# position PID default gains (used at every startup, then can be modified using the C++/Python script)
kp = 20.0
ki = 0.5
kd = 0.0f
windup = 15.0
max out = 30.0

[velocity PID]
# velocity PID default gains (used at every startup, then can be modified using the C++/Python script)
kp = 0.05
ki = 0.5
kd = 0.0
windup = 1.5
max out = 10.0

[impedance PD]
# impedance PID default gains (used at every startup, then can be modified using the C++/Python script)
kp = 3.5
kd = 0.25
max out = 10.0
```

`mdtool setup info <ID> [all]`

This command is used to read the motor internal parameters. Use an optional ‘all’ keyword at the end of the command to read full parameter set. An example command output might look like this:

```
MAB@tutorial:~/mdtool/md_tools/build/packages$ mdtool setup info 100
[MDTOOL] MDtool version: v1.2.1
[CANDLE] CANdle library version: v3.1
[CANDLE] Device firmware version: v2.0
[CANDLE] CANdle at /dev/ttyACM0, ID: 0xac12800687432d34 ready (USB)
[CANDLE] Added MD80 with ID: 100  [OK]
Drive 100:
- actuator name: AK80_9_AUX_OFF_START
- CAN speed: 1 M
- CAN termination resistor: disabled
- gear ratio: 0.111
- firmware version: V2.1
- hardware version: HW20
- build date: 30.1.2023
- commit hash: 214328da
- max current: 20.0 A
- bridge type: 3
- pole pairs: 21
- KV rating: 0 rpm/V
- motor shutdown temperature: 80 *C
- motor torque constant: 0.0910 Nm/A
- motor stiction: 0.000 Nm
- motor friction: 0.000 Nm
- d-axis resistance: 0.187 Ohm
- d-axis inductance: 0.000032 H
- torque bandwidth: 500 Hz
- CAN watchdog: 200 ms
- main encoder last check error stddev: 0.000000 rad
- main encoder last check min error 0.000000 rad
- main encoder last check max error: 0.000000 rad
- output encoder: AS5047_OFFAXIS
- output encoder mode: STARTUP
- output encoder position: -0.758170 rad
- output encoder velocity: 0.016989 rad/s
- output encoder last check error stddev: 0.000000 rad
- output encoder last check min error 0.000000 rad
- output encoder last check max error: 0.000000 rad
- position: -0.76 rad
- velocity: 0.04 rad/s
- torque: 0.00 Nm
- MOSFET temperature: 35.52 *C
- motor temperature: 0.00 *C

***** ERRORS *****
- main encoder error:   0x0     (ALL OK)
- output encoder error: 0x0     (ALL OK)
- calibration error:    0x0     (ALL OK)
- bridge error:         0x0     (ALL OK)
- hardware error:       0x0     (ALL OK)
- communication error:  0x0     (ALL OK)
```

Reading the errors is the easiest way of debugging possible problems with the drive. For errors description please visit the status section.

Skip to main content

This command is mostly used to find an MD drive on a long CAN bus using its ID – the command makes the drive flash its onboard LEDs for easy identification.

`mdtool test move <ID> <position>`

This command is used to test the actuator movement in impedance mode. It helps to assess if the calibration was successful and if there are no issues visible to the naked eye. The position argument is always the amount of position for the motor to be moved from the current position.

`mdtool test move absolute <ID> <target position> <velocity> <acceleration> <deceleration>`

This command is used to test the actuator movement in profile position mode. The motor is going to use `acceleration` parameter to speed up to `velocity` parameter and then `deceleration` parameter to slow down and reach the `target position`. The move is absolute to the currently set zero position. By providing only the `ID` and `target_velocity` the MDxx will use the default acceleration, deceleration and velocity parameters.

`mdtool test latency <baudrate>`

This command is used to test the PC<>CANdle communication speed which greatly affects the PC<>MD communication speed. The higher the measured frequency the better.

`mdtool test encoder <type> <ID>`

This command is used to check how accurate a praticular encoder was calibrated. The ‘type’ argument can be either ‘main’ for onboard encoder, or ‘output’ for output encoder. This command runs a routine that makes one full rotation of the shaft (either motor or output shaft, depending on the chosen encoder type) and after completing fills up the max, min and standard deviation errors that can be accessed using the `mdtool setup info all` command.

Skip to main content

> **Warning**
>
> Main encoder errors can be larger for non-sinusoidal motors (BLDC motors) because of their back-emf waveform shape. If you care about very precise positioning we advise using PMSM motors (sinusoidal).

`mdtool encoder <ID>`

This command is useful when one wants to measure the position of the actuator in the current setup (without writing a custom script). After the command is executed the screen shows the current position of the actuator's shaft and it does so until you press Ctrl + C.

`mdtool clear error <ID>`

This command is used to clear non-critical MDxx errors.

`mdtool clear warning <ID>`

This command is used to clear all MDxx warnings.

`mdtool register read <ID> <regID>`

This command is used to read a register from MDxx controller.

`mdtool register write <ID> <regID> <value>`

This command is used to write a register with `value` param.

`mdtool reset <ID>`

This command is used to reset an MDxx controller.

CANdle C++ library

CANdle C++ library is the base module of software that all other modules are based on. It takes care of low-level communication between the host and the MD controllers. Using the CANdle C++ library directly is the best option to reach the full performance of the drives when it comes to communication frequency between the host and MD controllers.

Quick start

The quick startup guide includes cloning the repo, building and running the examples. First, you should clone the candle repo from the MAB Robotics GitHub page to your local machine. Then, make sure you're in the main directory candle/ and run the following commands:

```
mkdir build
cd build
cmake ..
make
```

starting from the top one these commands: create a build directory, go into the build directory, generate makefiles using CMake and compile the source code using make. After executing these commands you should be able to see the compiled examples in the candle/build/ directory. To run one of them use the following command:

```
./exampleX
```

where X is the number of the example.

Building as a static lib

Candle C++ library can be built as a static or shared object library. In the quick startup guide, we used the default settings, thus the library was compiled to a shared object file. In case you'd like to build it for a static lib you should pass additional arguments to the cmake .. command:

After executing this command you should be able to see the following CMake output:

```
MAB@tutorial:~/MAB/candle/build$ cmake .. -DCANDLE_BUILD_STATIC=TRUE
-- The C compiler identification is GNU 9.4.0
-- The CXX compiler identification is GNU 9.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
[CANDLE] library type: STATIC
--
CANDLE_BUILD_SHARED: FALSE
CANDLE_BUILD_STATIC: TRUE
CANDLE_BUILD_PYTHON: FALSE
-- Configuring done
-- Generating done
-- Build files have been written to: /home/klonyyy/MAB/candle/build
```

In case you'd like to go back to shared lib just call:

```
cmake .. -DCANDLE_BUILD_STATIC=FALSE
```

or delete the build directory contents and call cmake .. again (the default library type is shared). This is what the cmake output looks like when reconfiguring for shared lib:

```
MAB@tutorial:~/MAB/candle/build$ cmake .. -DCANDLE_BUILD_STATIC=FALSE
[CANDLE] library type: SHARED
--
CANDLE_BUILD_SHARED: TRUE
CANDLE_BUILD_STATIC: FALSE
CANDLE_BUILD_PYTHON: FALSE
-- Configuring done
-- Generating done
-- Build files have been written to: /home/klonyyy/MAB/candle/build
```

CANdle Python library

CANdle Python library is a translated version of the C++ library using pybind11. The package can be

Skip to main content

```
python3 -m pip install pyCandleMAB
```

It can be used to quickly start playing with the actuators, without the need to build the C++ software pack. Example usage of Python examples is shown in the getting started guide. To achieve the best performance in low latency systems we advise using the C++ libraries directly.

> **Note**
>
> We distribute the binaries as well as sources - in case your platform is not recognized with the available binaries pip will try to build and install the library from the source.

CANdle ROS/ROS2 nodes

> **Hint**
>
> TL;DR: MD x CANdle - ROS/ROS2 startup guide

While C++ API is the most flexible way of interfacing with CANdle/MDxx, ROS/ROS2 APIs are also available. These have been designed as standalone C++ nodes that use the CANdle library on the backend. The nodes are designed to be used with already configured drives, thus functions such as setting FDCAN parameters are unavailable via ROS/2 API. We recommend configuring all drives first using MDtool or C++/Python API.

Nodes use ROS/2 services to perform initialization and enable/disable the drives. The initialization services available are:

```
/add_md80s
/zero_md80s
/set_mode_md80s
```

There are also two additional services for enabling/disabling the drives:

```
/enable_md80s
/disable_md80s
```

Skip to main content

Once the drives are enabled via `enable_md80s` service, the nodes will ignore all calls to services other than `disable_md80s`. When enabled, communication switches from service-based to topic-based. The nodes will publish to the topic:

```
/md80/joint_states
```

And will subscribe to topics:

```
/md80/motion_command
/md80/impedance_command
/md80/velocity_pid_command
/md80/position_pid_command
```

Quick startup guide - ROS

Let's run a simple example of the candle ROS node. In order to run the node, clone it into your local ROS workspace in the src folder. After that, build it with 'catkin' and run using the 'rosrun' command. Be sure to source your workspace with source devel/setup.sh prior to running the package, and in each new terminal window you're going to send commands related to the node.

First, start the roscore with the roscore command. Then run the node with arguments that fit your MD x CANdle setup. The general syntax is:

```
rosrun candle_ros candle_ros_node <BUS> <FDCAN baud>
```

for more information on how to run the node you can call:

```
rosrun candle_ros candle_ros_node --help
```

Example output from the terminal after launching the node:

```
MAB@tutorial:~/ROS/ros1_workspace$ rosrun candle_ros candle_ros_node USB 8M
[CANDLE] CANdle library version: v3.0
[CANDLE] Device firmware version: v1.4
[CANDLE] CANdle at /dev/ttyACM0, ID: 0xac12814dfd055747 ready (USB)
[CANDLE] Found CANdle with ID: 12399114896061847367
```

Skip to main content

In this example, we will be working with USB bus and 8M FDCAN baudrate.

Adding drives

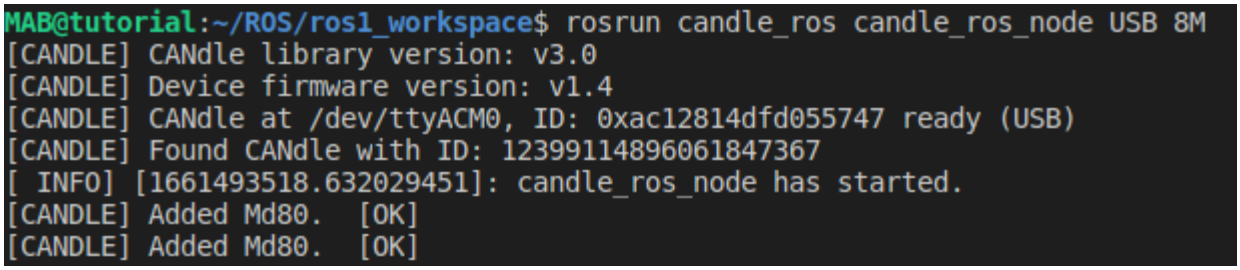
Firstly, the node should be informed which drives should be present on the FDCAN bus. This can be done via `/add_md80s service`. For example:

```
rosservice call /add_md80s "drive_ids: [200, 800]"
```

Should produce the following output:

```
drives_success: [True, True]
total_number_of_drives: 2
```

informing, that both drives (ids: 200 and 800), have been successfully contacted, and were added to the node's drives list. You can also look for status messages in the terminal window where the node was started:



According to the status messages we have added two MD series actuators.

Set mode

Next the desired control mode should be selected. This is accomplished with `/set_mode_md80s` service. For example:

```
rosservice call /set_mode_md80s "{drive_ids: [200, 800], mode:["IMPEDANCE", "IMPEDAN
```

Should produce:

Skip to main content

```
drives_success: [True, True]
```

Informing that for both drives mode has been set correctly.

Set Zero

Often when starting, setting a current position to zero is desired. This can be accomplished with a call to `/zero_md80s` service.

```
rosservice call /zero_md80s "{drive_ids:[200, 800]}"
```

Enabling/Disabling drives

Using services `/enable_md80s` and `/disable_md80s` the drives and the node publishers and subscribers can be enabled/disabled.

> **Note**
>
> After calling `/enable_md80s` service, no calls to services other than `/disable_md80s` should be done.

After enabling, the node will publish current joint states to `/joint_states` at a frequency dependent on a currently chosen communication bus and speed mode. Joint names are generated based on drive ID, for example, a drive with id 546 will be called `Joint 546`.

The node will also listen for the messages on topics for controlling the drives. All of the above topics are listened to all the time, but currently applied settings are dependent on the MD controller mode set before enabling.

```
rosservice call /enable_md80s "{drive_ids:[200, 800]}"
```

```
rosservice call /disable_md80s "{drive_ids:[200, 800]}"
```

Skip to main content

Controlling drives

Controlling the drives is done via the four topics listed above. For commands to be viable, all fields of each message must be filled properly. For example, to set up custom gains for IMPEDANCE mode use:

```
rostopic pub /md80/impedance_command candle_ros/ImpedanceCommand "{drive_ids:[200, 8
```

Example set up of custom gains for POSITION PID mode:

```
rostopic pub /md80/position_command candle_ros/PositionPidCommand "{drive_ids: [200,
```

Example set up of custom gains for VELOCITY PID mode:

```
rostopic pub /md80/velocity_command candle_ros/VelocityPidCommand "{drive_ids: [200,
```

Setting desired position, velocity, and torque is done via /md80/motion_command topic. Note that for it to take effect, all fields in the message should be correctly filled. For example, to move the drives in impedance mode, it is possible to use the following command

```
rostopic pub /md80/motion_command candle_ros/MotionCommand "{drive_ids:[81,97], targ
```

Quick start - ROS2

Let's run a simple example of the candle ROS2 node. In order to run the node, clone it into your local ROS2 workspace. After that, build it with `colcon` and run using the `ros2 run` command. Be sure to source your workspace with `source install/setup.bash` prior to running the package, and in each new terminal window you're going to send commands related to the node.

First, let's run the node with arguments that fit your MD x CANdle setup. The general syntax is:

```
ros2 run candle_ros2 candle_ros2_node <BUS> <EDCAN_baud>
```

for more information on how to run the node you can call

```
ros2 run candle_ros2 candle_ros2_node --help.
```

Example output from the terminal after launching the node:

```
MAB@tutorial:~/ROS/ros2_workspace$ ros2 run candle_ros2 candle_ros2_node USB 8M
[CANDLE] CANdle library version: v3.0
[CANDLE] Device firmware version: v1.4
[CANDLE] CANdle at /dev/ttyACM0, ID: 0xac12814dfd055747 ready (USB)
[CANDLE] Found CANdle with ID: 12399114896061847367
[INFO] [1661498441.729464170] [candle_ros2_node]: candle_ros2_node v1.1 has started.
```

In this example, we will be working with a USB bus and 8M FDCAN baudrate.

Adding drives Firstly, the node should be informed which drives should be present on the FDCAN bus. This can be done via `/candle_ros2_node/add_md80s` service. Note: Do not forget to source your ros2 workspace in new terminal window For example

```
ros2 service call /candle_ros2_node/add_md80s candle_ros2/srv/AddMd80s "{drive_ids:
```

Should produce the following output: response:

```
candle_ros2.srv.AddMd80s_Response(drives_success=[True, True], total_number_of_drive
```

informing, that both drives (ids: 200 and 800), have been successfully contacted, and were added to the node's drives list. You can also look for status messages in the terminal window where the node was started:

```
MAB@tutorial:~/ROS/ros2_workspace$ ros2 run candle_ros2 candle_ros2_node USB 8M
[CANDLE] CANdle library version: v3.0
[CANDLE] Device firmware version: v1.4
[CANDLE] CANdle at /dev/ttyACM0, ID: 0xac12814dfd055747 ready (USB)
[CANDLE] Found CANdle with ID: 12399114896061847367
[INFO] [1661498441.729464170] [candle_ros2_node]: candle_ros2_node v1.1 has started.
S[CANDLE] Added Md80.  [OK]
[CANDLE] Added Md80.  [OK]
```

According to the status messages we have added two MD actuators.

Skip to main content

Set mode

Next the desired control mode should be selected. This is accomplished with /candle_ros2_node/set_mode_md80s service.

For example:

```
ros2 service call /candle_ros2_node/set_mode_md80s candle_ros2/srv/SetModeMd80s "{dr
```

Should produce:

```
response:
candle_ros2.srv.SetModeMd80s_Response(drives_success=[True, True])
```

Informing that for both drives mode has been set correctly.

Set Zero

Often when starting, setting a current position to zero is desired. This can be accomplished with a call to `/candle_ros2_node/zero_md80s` service.

```
ros2 service call /candle_ros2_node/zero_md80s candle_ros2/srv/GenericMd80Msg "{driv
```

Enabling/Disabling drives

Using services `/candle_ros2_node/enable_md80s` and `/candle_ros2_node/disable_md80s` the drives and the node publishers and subscribers can be enabled/disabled.

Note

After calling `/candle_ros2_node/enable_md80s` service, no calls to services other than `/candle_ros2_node/disable_md80s` should be done.

Skip to main content

After enabling, the node will publish current joint states to `/joint_states` at a frequency dependent on a currently chosen communication bus and speed mode. Joint names are generated based on drivie ID, for example, drive with id 546 will be called `Joint 546`.

The node will also listen for the messages on topics for controlling the drives. All of the above topics are listened to all the time, but currently applied settings are dependent on the MD controller mode set before enabling.

```
ros2 service call /candle_ros2_node/enable_md80s candle_ros2/srv/GenericMd80Msg "{dr
```

```
ros2 service call /candle_ros2_node/disable_md80s candle_ros2/srv/GenericMd80Msg "{d
```

Controlling drives

Controlling the drives is done via the four topics listed above. For commands to be viable, all field of each message must be filled properly. For example, to set up custom gains for IMPEDANCE mode use:

```
ros2 topic pub /md80/impedance_command candle_ros2/msg/ImpedanceCommand "{drive_ids:
```

Example set up of custom gains for POSITION PID mode:

```
ros2 topic pub /md80/position_pid_command candle_ros2/msg/PositionPidCommand "{drive
```

Example set up of custom gains for VELOCITY PID mode:

```
ros2 topic pub /md80/velocity_pid_command candle_ros2/msg/VelocityPidCommand "{drive
```

Setting desired position, velocity, and torque is done via `/md80/motion_command` topic. Note that for it to take effect, all fields in the message should be correctly filled. For example, to move the drives in impedance mode, it is possible to use the following command

```
ros2 topic pub /md80/motion_command candle_ros2/MotionCommand "{drive_ids: [200, 800
```

MD FDCAN communication

The easiest way to communicate with MD controllers is to use a CANdle device connected to a PC. Even though we are aware some customers want to integrate the MD controllers in their product with minimal setup to reduce the costs and the system's complexity. This manual will guide you through the process of communicating with MD actuators from your custom FDCAN-capable master controller.

Hardware requirements

The main requirement for the host system is to be equipped with an FDCAN peripheral (either a built-in one or an external one) and an FDCAN transceiver capable of speeds up to 8Mbps. Lower maximum speed transceivers can be used as well, however for the cost of limited update rates. Depending on your custom setup you should be able to integrate a 120 ohm terminating resistor on both ends of your CAN bus.

> **Note**
>
> MD controllers from version 2.0 can be upgraded to software controlled termination. Please contact us for more information

Communication Structure

The communication stack is based on a register access using two frames - register read and register write. The list of available registers can be found at the end of this chapter. All fields are little-endian - least significant byte first, and all float fields are 4 bytes long encoded in IEEE-754 standard.

Default response

The default response is sent by the drive in case a register write operation was successful.

Skip to main content

	BYTE 0	BYTE 1-2	BYTE 3	BYTE 4-7	BYTE 8-11	BYTE 12-15	BYTE 16-19	BYTE 20-23
NAME	FRAME ID	QUICK STATUS	MOTOR TEMPERATURE	MAIN ENCODER POSITION	MAIN ENCODER VELOCITY	MOTOR TORQUE	OUTPUT ENCODER POSITION	OUTPUT ENCODER VELOCITY
TYPE	uint8_t	uint16_t	uint8_t [*C]	float [rad]	float [rad/s]	float [Nm]	float [rad]	float [rad/s]
VALUE	0x0A	0x0000 - 0xFFFF	0 - 255	-	-	-	-	-

In case the operation initiated by a frame was unsuccessful the MDxx will not respond.

Write register frame

Write register frame is used to modify values of the user-modifiable registers. Only registers with write access can be modified.

FRAME NAME	DRIVE ID	LENGTH	BYTE 0 [ID]	BYTE 1	BYTE 2-3	BYTE 4-X	BYTE X+1-X+2	BYTE X+4-X+Y
WRITE_REGISTER	10-999	X (64 max)	0x40	0x00	reg ID	value	reg ID	value

Params:

- regID (uint16_t) - first register ID (please see the end of this section)
- value (uint8_t/uint16_t/uint32_t/float/char[]) - first register value to be written
- regID (uint16_t) - second register ID (please see the end of this section)
- value (uint8_t/uint16_t/uint32_t/float/char[]) - second register value to be written
- … (up to 64 bytes total)

When all registers write operations succeed the drive will respond with default response.

Read Register Frame

Read register command is used to retrieve certain register values. The actuator will respond with a frame consisting of the addresses and values of the registers issued in the master request. The master request should have the following form:

FRAME NAME	DRIVE ID	LENGTH	BYTE 0 [ID]	BYTE 1	BYTE 2-3	BYTE 4-X	BYTE X+1-X+2	BYTE X+4-X+Y
READ_REGISTER	10-999	X (64 max)	0x41	0x00	reg ID	0x00	reg ID	0x00

When all read operations succeed the 0x00 fields will be filled with appropriate register data when

FRAME NAME	DRIVE ID	LENGTH	BYTE 0 [ID]	BYTE 1	BYTE 2-3	BYTE 4-X	BYTE X+1-X+2	BYTE X+4-X+Y
Response to register read	10-999	X (64 max)	0x41	0x00	reg ID	reg value	reg ID	reg value

Warning

Frame payload length must not exceed 64 bytes.

Skip to main content

Available registers

reg name	address	read/write	size	limits	description
canId	0x001	RW	uint32_t	[10-2000]	FDCAN bus id number
canBaudrate	0x002	RW	uint32_t	[1e6;2e6;5e6;8e6]	FDCAN bus baudrate
canWatchdog	0x003	RW	uint16_t	[0-2500]	FDCAN bus watchdog period in ms
canTermination	0x004	RW	uint8_t	[0-1]	CAN termination (available upon request)
motorName	0x010	RW	char[24]	-	motor name
motorPolePairs	0x011	RW	uint32_t	[2;225]	motor pole pair count
motorKt	0x012	RW	float	>0	motor torque constant
motorKt_a	0x013	RW	float	>0	optional parameter for phase specific torque constant
motorKt_b	0x014	RW	float	>0	optional parameter for phase specific torque constant
motorKt_c	0x015	RW	float	>0	optional parameter for phase specific torque constant
motorIMax	0x016	RW	float	[1 - peak controller current]	maximum phase current
motorGearRatio	0x017	RW	float	-	actuator gear ratio (ex 2:1 should be 0.5) <1 - reducer >1 - multiplier
motorTorqueBandwidth	0x018	RW	uint16_t	[50-2500]	torque bandwidth in Hz
motorFriction	0x019	RO	float32	-	UNUSED
motorStiction	0x01A	RO	float32	-	UNUSED
motorResistance	0x01B	RO	float	[5mOhm-20Ohm]	motor resistance in d axis
motorInductance	0x01C	RO	float	[5nH-100mH]	motor inductance in d axis
motorKV	0x01D	RW	uint16_t	-	motor KV rating [RPM/V]
motorCalibrationMode	0x01E	RW	uint8_t	[0;1]	FULL = 0, NOPPDET = 1
motorThermistorType	0x01F	RW	uint8_t	-	-
outputEncoder	0x020	RW	uint8_t	[0;1;2;3]	NONE = 0, AS5047_CENTER = 1, AS5047_OFFAXIS = 2, MB053SFA17BENT00 = 3, CM_OFFAXIS = 4, M24B_CENTER = 5, M24B_OFFAXIS = 6
outputEncoderDir	0x021	RW	uint8_t	0	RESERVED
outputEncoderDefaultBaud	0x022	RW	uint32_t	115200	optional parameter for setting default output encoder baudrate
outputEncoderVelocity	0x023	RO	float	-	output encoder velocity in rad/s (calculated in a 5kHz loop)
outputEncoderPosition	0x024	RO	float	-	output encoder position in rad (read in 5kHz loop)
outputEncoderMode	0x025	RW	uint8_t	[0;1;2;3;4]	NONE = 0, STARTUP = 1, MOTION = 2, REPORT = 3, MAIN = 4
outputEncoderCalibrationMode	0x026	RW	uint8_t	[0;1]	FULL = 0, DIRONLY = 1
motorPosPidKp	0x030	RW	float	-	position PID proportional gain
motorPosPidKi	0x031	RW	float	-	position PID integral gain
motorPosPidKd	0x032	RW	float	-	position PID derivative gain

Skip to main content

reg name	address	read/write	size	limits	description
motorVelPidKp	0x040	RW	float	-	velocity PID proportional gain
motorVelPidKi	0x041	RW	float	-	velocity PID integral gain
motorVelPidKd	0x042	RW	float	-	velocity PID derivative gain
motorVelPidWindup	0x044	RW	float	-	velocity PID integral windup limit
motorImpPidKp	0x050	RW	float	-	impedance PD proportional gain
motorImpPidKd	0x051	RW	float	-	impedance PD derivative gain
mainEncoderVelocity	0x062	RO	float	-	main encoder velocity in rad/s (calculated in a 40kHz loop)
mainEncoderPosition	0x063	RO	float	-	main encoder position in rad (read in 40kHz loop)
motorTorque	0x064	RO	float	-	motor output shaft torque in Nm (read in 40kHz loop)
runSaveCmd	0x080	WO	uint8_t	other than 0 to run	save non-volatile memory
runTestMainEncoderCmd	0x081	WO	uint8_t	other than 0 to run	runs main encoder test routine
runTestOutputEncoderCmd	0x082	WO	uint8_t	other than 0 to run	runs output encoder test routine
runCalibrateCmd	0x083	WO	uint8_t	other than 0 to run	runs main calibration routine
runCalibrateOutputEncoderCmd	0x084	WO	uint8_t	other than 0 to run	runs output encoder calibration routine
runCalibratePiGains	0x085	WO	uint8_t	other than 0 to run	runs current PI loop calibration routine
runRestoreFactoryConfig	0x087	WO	uint8_t	other than 0 to run	reverts config to factory state
runReset	0x088	WO	uint8_t	other than 0 to run	resets the controller
runClearWarnings	0x089	WO	uint8_t	other than 0 to run	clears all warnings
runClearErrors	0x08A	WO	uint8_t	other than 0 to run	clears non-critical errors
runBlink	0x08B	WO	uint8_t	other than 0 to run	blinks onboard LEDs
runZero	0x08C	WO	uint8_t	other than 0 to run	sets new zero position
runCanReinit	0x08D	WO	uint8_t	other than 0 to run	reinitializes can peripheral
calOutputEncoderStdDev	0x100	RO	float	-	output encoder test result (standard deviation)
calOutputEncoderMinE	0x101	RO	float	-	output encoder test result (min error)
calOutputEncoderMaxE	0x102	RO	float	-	output encoder test result (max error)
calMainEncoderStdDev	0x103	RO	float	-	main encoder test result (standard deviation)
calMainEncoderMinE	0x104	RO	float	-	main encoder test result (min error)
calMainEncoderMaxE	0x105	RO	float	-	main encoder test result (max error)
positionLimitMax	0x110	RW	float	-	maximum valid position
positionLimitMin	0x111	RW	float	-	minimum valid position
maxTorque	0x112	RW	float	-	maximum torque
maxVelocity	0x113	RW	float	-	maximum velocity
maxAcceleration	0x114	RW	float	-	maximum acceleration
maxDeceleration	0x115	RW	float	-	maximum deceleration
profileVelocity	0x120	RW	float	-	profile velocity
profileAcceleration	0x121	RW	float	-	profile acceleration
profileDeceleration	0x122	RW	float	-	profile deceleration

Skip to main content

reg name	address	read/write	size	limits	description
positionWindow	0x124	RW	float	-	position window within position is considered to be reached
velocityWindow	0x125	RW	float	-	velocity window within velocity is considered to be reached
motionModeCommand	0x140	WO	uint8_t	-	commands a motion mode change: IDLE = 0x00, POSITION_PID = 0x01, VELOCITY_PID = 0x02, RAW_TORQUE = 0x03, IMPEDANCE = 0x04, POSITION_PROFILE = 0x07, VELOCITY_PROFILE = 0x08
motionModeStatus	0x141	RO	uint8_t	-	shows the currently set motion mode
state	0x142	RW	uint16_t	-	returns the internal state machine state of the controller
targetPosition	0x150	RW	float	-	sets target position in rad
targetVelocity	0x151	RW	float	-	sets target velocity in rad/s
targetTorque	0x152	RW	float	-	sets target torque in Nm
userGpioConfiguration	0x160	RW	uint8_t	-	0 - OFF, 1 - AUTO-BRAKE, 2 - GPIO INPUT
userGpioState	0x161	RO	uint16_t	-	GPIO input state
reverseDirection	0x600	RW	uint8_t	-	used to change the direction of the main encoder when using other encoders than the onboard one. Always recalibrate after changing this setting
shuntResistance	0x700	RO	float	[0.001 - 0.01]	Current sense resistor value. Setting this register to a value that is not coherent with the hardware may damage the controller. In this cases warranty is not respected.
buildDate	0x800	RO	uint32_t	-	software build date
commitHash	0x801	RO	char[8]	-	commit hash
firmwareVersion	0x802	RO	uint32_t	-	-
hardwareVersion	0x803	RO	uint8_t	-	-
bridgeType	0x804	RO	uint8_t	-	type of the mosfet driver
quickStatus	0x805	RO	uint16_t	-	quick status vector
mosfetTemperature	0x806	RO	float	-	power stage temperature
motorTemperature	0x807	RO	float	-	motor temperature (if thermistor is mounted)
motorShutdownTemp	0x808	RW	uint8_t	-	temperature at which the MD series motor controller will enter IDLE mode
mainEncoderErrors	0x809	RO	uint32_t	-	main encoder errors
outputEncoderErrors	0x80A	RO	uint32_t	-	output encoder errors
calibrationErrors	0x80B	RO	uint32_t	-	calibration errors
bridgeErrors	0x80C	RO	uint32_t	-	bridge errors
hardwareErrors	0x80D	RO	uint32_t	-	hardware errors
communicationErrors	0x80E	RO	uint32_t	-	communication errors
motionErrors	0x810	RO	uint32_t	-	motion errors

CANopen

MD controllers are capable of supporting CANopen communication protocol according the CiA402 device profile. For further details, please examine the Object Dictionary chapter that contains the communication objects that are used to exchange data.

> **Note**
>
> Make sure you've contacted MABRobotics for the appropriate software update, as the default MD flasher only supports CANdle protocol.

Setting up a new motor

To set up a new motor make sure the drive is in the "switch on disabled" state. After that make sure you set up the following registers:

1. Motor Parameters:

- 0x6075:0 - Rated Current
- 0x6076:0 - Rated Torque
- 0x6072:0 - Max Torque
- 0x6072:0 - Max Current
- 0x6080:0 - Max Speed
- 0x2000:1 - Pole Pairs
- 0x2000:2 - Torque Constant
- 0x2000:5 - Torque Bandwidth
- 0x2000:7 - Motor Shutdown Temp
- 0x2000:8 - Gear Ratio

2. Velocity PID gains:

- 0x2001:1 - Kp

Skip to main content

- 0x2001:3 - Kd
- 0x2001:4 - Integral Limit

3. Position PID gains:

- 0x2002:1 - Kp
- 0x2002:2 - Ki
- 0x2002:3 - Kd
- 0x2002:4 - Integral Limit

4. Profile velocity/position settings:

- 0x6081 - Profile Velocity
- 0x6083 - Profile Acceleration
- 0x6084 - Profile Deceleration
- 0x6085 - Quick Stop Deceleration

> **Note**
>
> Remember to save the parameters - please see the last chapter

Setting up an external encoder

Setup of the external encoder should be done after the motor has been configured, as suggested in previous paragraph. To setup the output encoder the following fields need to be filled:

- 0x2005:1 - Output Encoder Type (please see output encoder)
- 0x2005:2 - Calibration Mode (please see output encoder calibration)
- 0x2005:3 - Mode (please see output encoder)

> **Note**
>
> Remember to save the parameters - please see the last chapter

Skip to main content

Saving and Calibrating

After the parameters have been filled there are two more steps to follow - save to non-volatile memory and calibrate:

1. Run store parameters routine using 0x1010:1:
    - make sure the state machine is in “switch on disabled” state (write 0x8 to controlword 0x6040)
    - write 0x65766173 to 0x1010:1
    - wait for the drive to reboot
2. Run calibration routine using 0x2003:3 (and output encoder calibration routine 0x2003:4 if output encoder is present)
    - make sure the state machine is in “operational” state (write sequentially 0x8, 0x6, 0xf to controlword 0x6040, and check if the statusword 0x6041 is 0x39)
    - make sure the drive is in “service” operation mode (write -2 to Modes Of Operation 0x6060)
    - write 1 to 0x2003:3 to start the calibration
    - wait for the drive to reboot

Object Dictionary

Object dictionary holds CAN objects that can be accessed using SDOs and in some cases by PDOs. There are three main groups in which the address space is divided into:

1. Communication Area
2. Manufacturer Specific Area
3. Profile Specific Area

1. Communication Area

The communication area describes the CANopen objects compliant with CiA301 standard.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x1000	0x00	Device type	UINT32	RO	-	-	-	0x00020192	-

0x1001 - Error Register

Indicates whether an error has occured. Currently, only the 0th bit is implemented, that indicates a general error. For a more verbose error and warning status, please see 0x2004 System Status.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x1001	0x00	Error Register	UINT8	RO	TX	-	-	0x00	-

Bit	Meaning
0	General Error

0x1008 - Manufacturer Device Name

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x1008	0x00	Manufacturer Device Name	STR	RO	-	-	-	"MD80"	-

0x1010 - Store Parameters

Use this object for saving parameters in non-volatile memory. Works only in “switch on disabled” state. To avoid saving parameters by mistake a value of 0x65766173 has to be explicitly written to 0x1010:1.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x1010	0x01	Store Parameters	UINT32	RW	-	-	-	-	-

0x1017 - Producer Heartbeat Time

Defines the period of heartbeat message sent by the MD80.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x1017	0x00	Producer Heartbeat Time	UINT16	RW	-	-	-	-	-

0x1600 - Receive PDO1 mapping

Skip to main content

Index	PDO Index	Name
0x1600	-	Receive PDO1 mapping
0x1600:1	0x6040	ControlWord

0x1601 - Receive PDO2 mapping

Index	PDO Index	Name
0x1601	-	Receive PDO2 mapping
0x1601:1	0x6040	ControlWord
0x1601:2	0x6060	Modes Of Operation

0x1602 - Receive PDO3 mapping

Index	PDO Index	Name
0x1602	-	Receive PDO3 mapping
0x1602:1	0x6040	ControlWord
0x1602:2	0x607A	Target Position

0x1603 - Receive PDO4 mapping

Index	PDO Index	Name
0x1603	-	Receive PDO4 mapping
0x1603:1	0x6040	ControlWord
0x1603:2	0x60FF	Target Velocity

0x1A00 - Transmit PDO1 mapping

Index	PDO Index	Name
0x1A00	-	Transmit PDO1 mapping
0x1A00:1	0x6041	StatusWord
0x1A00:2	0x6061	Modes Of Operation Display

0x1A01 - Transmit PDO2 mapping

Index	PDO Index	Name
0x1A01	-	Transmit PDO2 mapping

Skip to main content

0x1A02 - Transmit PDO3 mapping

Index	PDO Index	Name
0x1A02	-	Transmit PDO3 mapping
0x1A02:1	0x6041	StatusWord
0x1A02:2	0x606C	Velocity Actual Value

0x1A03 - Transmit PDO4 mapping

Index	PDO Index	Name
0x1A03	-	Transmit PDO4 mapping
0x1A03:1	0x6041	StatusWord
0x1A03:2	0x6077	Torque Actual Value

2. Manufacturer Specific Area

The manufacturer specific area describes the custom CANopen objects, valid only for MD80s.

0x2000 - Motor Settings

Configures the most important motor settings. This object is especially useful when you want to configure or reconfigure an MD series motor controller for a particular motor. Be sure to save after modification using 0x1010 Store Parameters.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x2000	0x01	Pole Pairs	UINT32	RW	-	yes	[2;255]	0	-
0x2000	0x02	Torque constant	FLOAT32	RW	-	yes	-	0	Nm/A
0x2000	0x03	Phase Inductance	FLOAT32	RW	-	yes	[5nH-100mH]	0	H
0x2000	0x04	Phase Resistance	FLOAT32	RW	-	yes	[5mOhm-20Ohm]	0	Ohm
0x2000	0x05	Torque Bandwidth	UINT16	RW	-	yes	50-2500	0	Hz
0x2000	0x06	Motor Name	STR(20)	RW	-	yes	-	0	-
0x2000	0x07	Motor Shutdown Temperature	UINT8	RW	-	yes	10-80	0	*C
0x2000	0x08	Gear Ratio	FLOAT32	RW	-	yes	-	1	-
0x2000	0x09	Calibration Mode	UINT8	RW	-	yes	FULL = 0, NOPPDET = 1	0	-
0x2000	0x0A	Can ID	UINT32	RW	-	yes	1 - 31	1	-

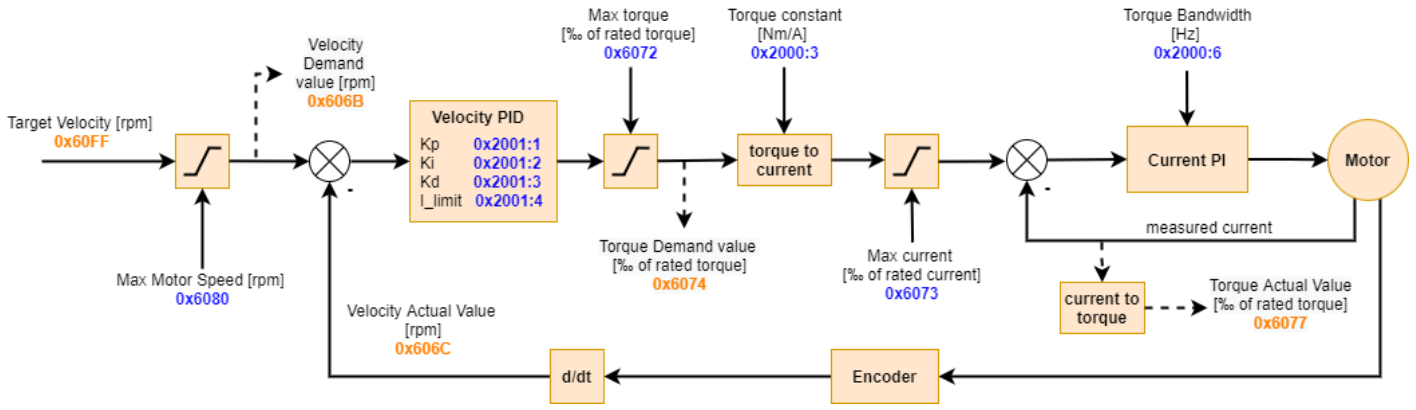
Skip to main content

0x2001 - Velocity PID Controller

Configures the Velocity PID controller gains. Be sure to save after modification using 0x1010 Store Parameters.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x2001	0x01	Kp	FLOAT32	RW	-	yes	-	0	-
0x2001	0x02	Ki	FLOAT32	RW	-	yes	-	0	-
0x2001	0x03	Kd	FLOAT32	RW	-	yes	-	0	-
0x2001	0x04	Integral Limit	FLOAT32	RW	-	yes	-	0	Nm

Velocity PID Controller structure

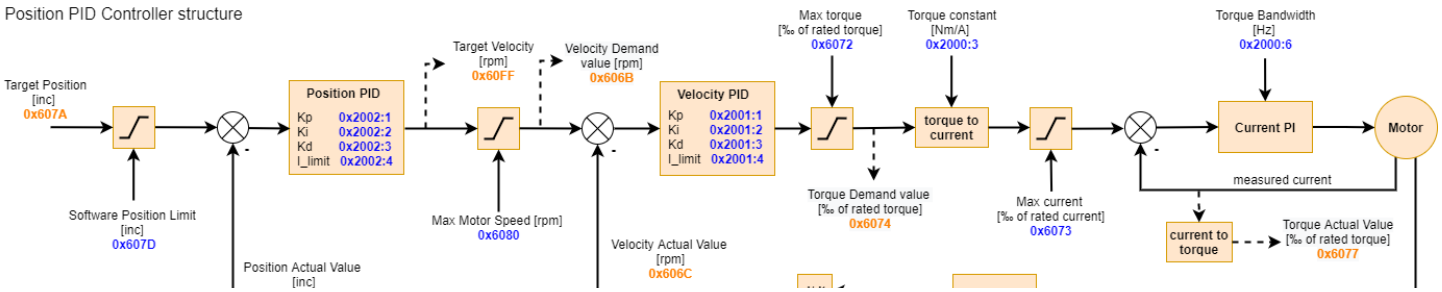


0x2002 - Position PID Controller

Configures the Position PID controller gains. Be sure to save after modification using 0x1010 Store Parameters.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x2002	0x01	Kp	FLOAT32	RW	-	yes	-	0	-
0x2002	0x02	Ki	FLOAT32	RW	-	yes	-	0	-
0x2002	0x03	Kd	FLOAT32	RW	-	yes	-	0	-
0x2002	0x04	Integral Limit	FLOAT32	RW	-	yes	-	0	rad/s

Position PID Controller structure



Skip to main content

0x2003 - System Command

Allows to issue a system command. Write a non-zero value to start a specific action. Actions work only in “switch on disabled” state and “service” (-1) operation mode.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x2003	0x01	Blink LEDs	BOOL	RW	-	-	-	0	-
0x2003	0x02	Reset Controller	BOOL	RW	-	-	-	0	-
0x2003	0x03	Run Calibration	BOOL	RW	-	-	-	0	-
0x2003	0x04	Run Output Encoder Calibration	BOOL	RW	-	-	-	0	-
0x2003	0x05	Set Zero	BOOL	RW	-	-	-	0	-
0x2003	0x06	Calibrate Current PI Gains	BOOL	RW	-	-	-	0	-

0x2004 - System Status

Allows to read System status. Each specific status is a UINT32, where lower bits (0-15) indicate errors, and higher bits (16-31) indicate warnings. Please see the Status section to see how to decode the status to individual fields.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x2004	0x01	Main Encoder Status	UINT32	RW	-	-	-	0	-
0x2004	0x02	Output Encoder Status	UINT32	RW	-	-	-	0	-
0x2004	0x03	Calibration Status	UINT32	RW	-	-	-	0	-
0x2004	0x04	Bridge Status	UINT32	RW	-	-	-	0	-
0x2004	0x05	Hardware Status	UINT32	RW	-	-	-	0	-
0x2004	0x06	DEPRECATED	UINT32	RW	-	-	-	0	-
0x2004	0x07	Motion Status	UINT32	RW	-	-	-	0	-
0x2004	0x08	Communication Status	UINT32	RW	-	-	-	0	-

0x2005 - Output Encoder

Output encoder related record.

Skip to main content

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x2005	0x01	Type	UINT32	RW	-	yes	1 (AS5047_CENTER), 2 (AS5047_OFFAXIS), 3 (MB053SFA17BENT00)	0	-
0x2005	0x02	Calibration Mode	UINT32	RW	-	yes	FULL = 0, DIRONLY = 1	0	-
0x2005	0x03	Mode	UINT32	RW	-	yes	NONE = 0, STARTUP = 1, MOTION = 2, REPORT = 3	0	-
0x2005	0x04	Position	FLOAT32	RO	TX	-	-	0.0	rad
0x2005	0x05	Velocity	FLOAT32	RO	TX	-	-	0.0	rad/s

0x2006 - Temperature

Motor and mosfet temperature readout record.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x2006	0x01	Motor Temperature	FLOAT32	RO	TX	-	-	0.0	-
0x2006	0x02	Mosfet Temperature	FLOAT32	RO	TX	-	-	0.0	-

3. Profile Specific Area

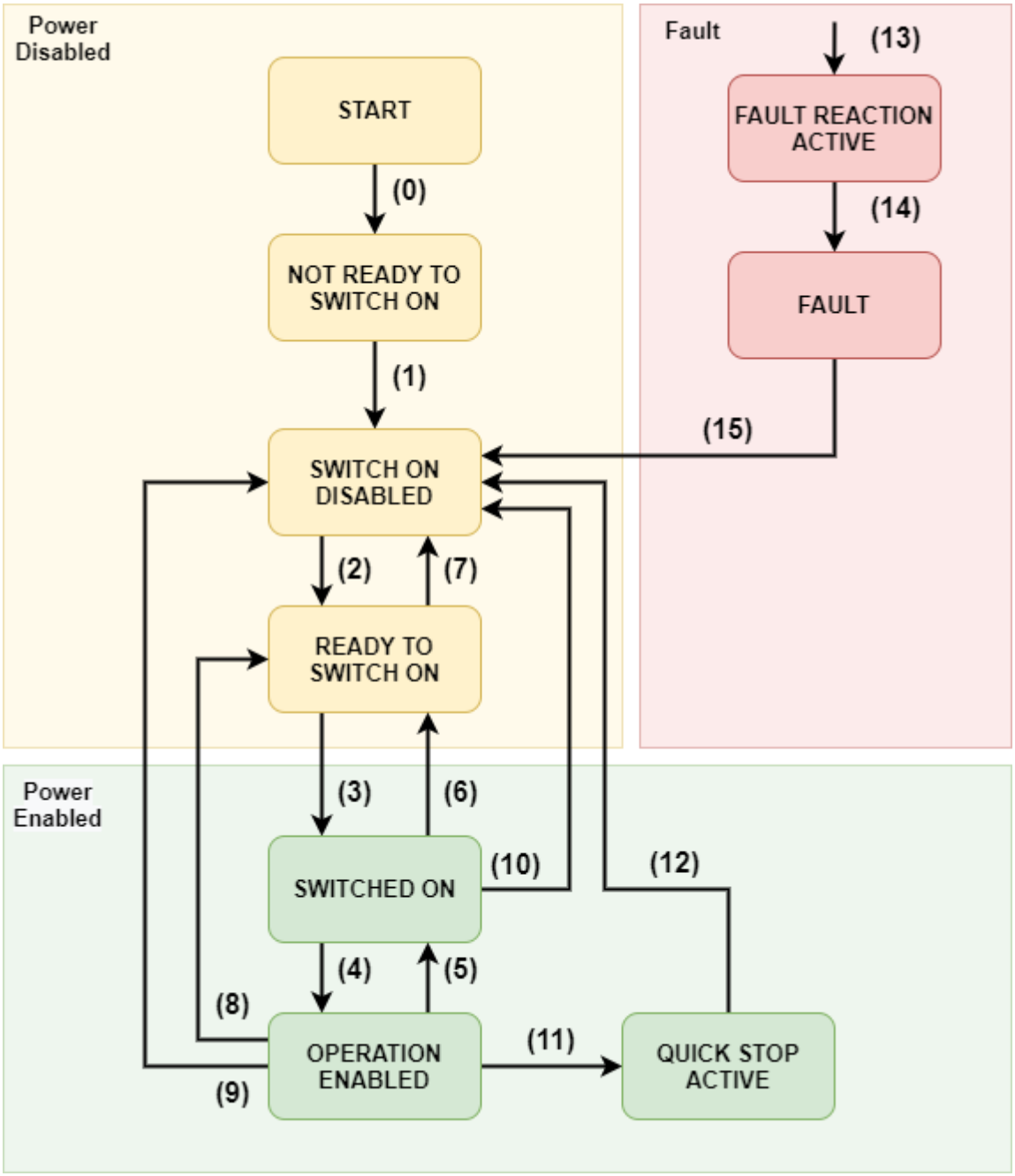
The profile specific area describes the CANopen objects compliant with CiA402 standard.

0x6040 - Control Word

Control word is used to change the state of the internal CiA402 state machine implemented on the drive.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x6040	0x00	Control Word	UINT16	RW	RX	-	-	0x00	-

The state machine is defined as follows:



All the transitions are based on the control word. The current state can be read using the status word (0x6041).

Skip to main content

Command	Reset Fault (bit 7)	Enable Operation (bit 3)	Quick Stop (bit 2)	Disable Voltage (bit 1)	Switch On (bit 0)	Decimal value
Shutdown	0	X	1	1	0	6
Switch on	0	0	1	1	1	7
Switch on and Enable Operation	0	1	1	1	1	15
Disable Voltage	0	X	X	0	X	0
Quick Stop	0	X	0	1	X	2
Disable operation	0	0	1	1	1	7
Enable operation	0	1	1	1	1	15
Fault reset	0->1	X	X	X	X	128

X means “do not care”

Example:

To put the drive into operational mode set:

1. Control word = 6 (dec) (Shutdown cmd)
2. Control word = 15 (dec) (Switch on and Enable Operation cmds)

The events and respective transitions are gathered in the table below:

Transition	Event	Internal action
0	Automatic transition after power up	Drive internal initialization
1	Automatic transition after drives internal initialization	Object dictionary is initalized with NVM data
2	Shutdown command received	None
3	Switch on command received	None
4	Enable operation command received	Current controllers are on, power is applied to the motor
5	Disable operation command received	Current controllers are off, power is not applied to the motor
6	Shutdown command received	Current controllers are turned off
7	Quick stop command received	Current controllers are turned off
8	Shutdown command received	Current controllers are turned off
9	Disable voltage command received	Current controllers are turned off
10	Disable voltage / Quick stop command received	Current controllers are turned off
11	Quick stop command received	Quick stop action is enabled. The drive decelerates and transits to SWITCH ON DISABLED
12	Automatic transition when quick stop is completed	Current controllers are turned off
13	Fault ocurred	Current controllers are turned off
14	Automatic transition to fault state	None
15	Fault reset command received	Fault is cleared if not critical

0x6041 - Status Word

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x6041	0x00	Status Word	UINT16	RO	TX	-	-	0x00	-

Status Word	State Machine State
xxxx xxxx x0xx 0000	Not ready to switch on
xxxx xxxx x1xx 0000	Switch on disabled
xxxx xxxx x01x 0001	Ready to switch on
xxxx xxxx x01x 0011	Switched on
xxxx xxxx x01x 0111	Operation Enabled
xxxx xxxx x00x 0111	Quick stop active
xxxx xxxx x0xx 1111	Fault reaction active
xxxx xxxx x0xx 1000	Quick stop active

bit 10 of the statusword indicates the current target has been reached (1) or not (0). This bit is motion mode - dependent, meaning for example in position mode it indicates the position has been reached (within a 0x6067 Position Window margin), and in velocity mode that a velocity target has been reached (within 0x606D Velocity Window)

Skip to main content

bit 11 of the statusword indicates whether any of the internal limits was active during current power up - for more information on which limit is active, check the 0x2004:7 Motion Status.

0x6060 - Modes Of Operation

Use this object to request a motion mode change. The actual mode is reflected in 0x6061 Modes Of Operation Display.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x6060	0x00	Modes Of Operation	INT8	RW	RX	-	-	0x00	-

Value	Mode
-1	Service
0	Idle
1	Profile Position
3	Profile Velocity
8	Cyclic Sync Position
9	Cyclic Sync Velocity

Service

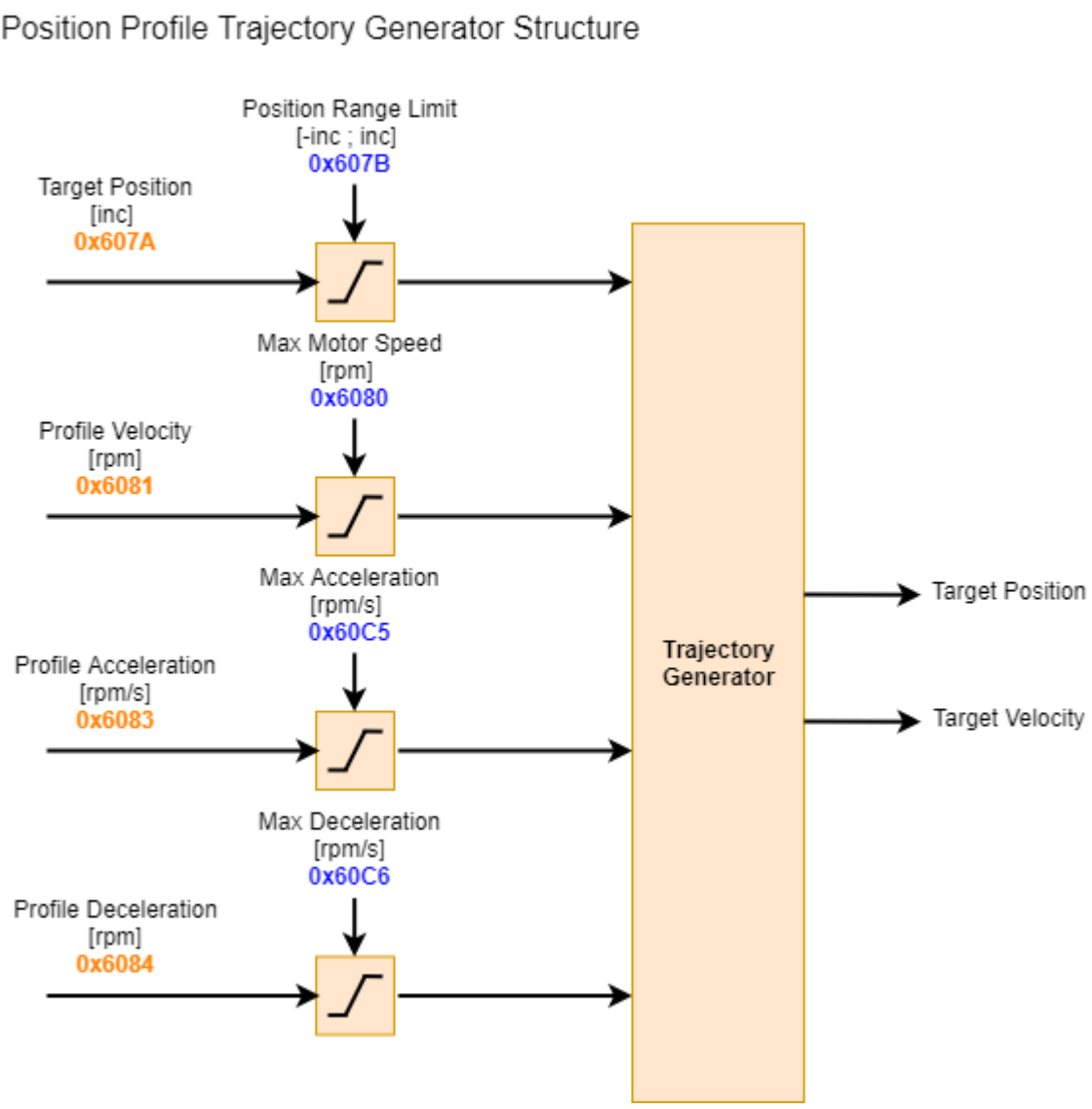
Mode in which System Commands can be issued.

Idle

Default state of the drive. No torque is produced, motor phases are shorted to GND which causes a damping sensation on the shaft.

Profile position

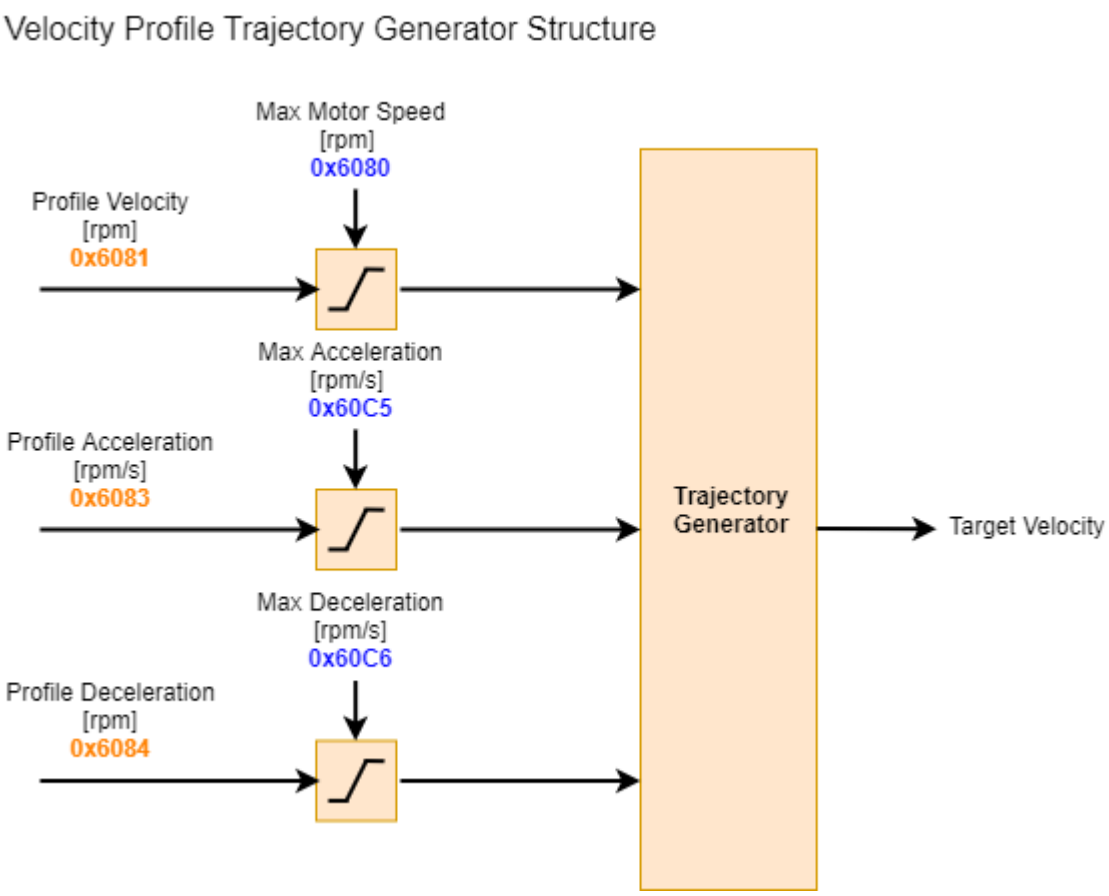
Profile position mode uses a trapeziodal trajectory generator on top of the Position PID controller. Allows to perform smooth point-to-point movements.



Profile velocity

Profile velocity mode uses a trapeziodal trajectory generator on top of the Velocity PID controller. Allows to reach a certain velocity with a constant acceleration / deceleration.

Skip to main content



Cyclic Sync Position

Raw position PID controller. Target position is reached as fast as possible, respecting the position range limits, max velocity, and max torque limit. To achieve smooth trajectories new setpoints need to be sent with high frequency.

Cyclic Sync Velocity

Raw velocity PID controller. Target velocity is reached as fast as possible, respecting the max velocity limit, and max torque limit. To achieve smooth acceleration new velocity setpoints need to be sent with high frequency.

0x6061 - Modes Of Operation Display

Use this object to read current motion mode

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units

Skip to main content

Value	Mode
-1	Service
0	Idle
1	Profile Position
3	Profile Velocity
8	Cyclic Sync Position
9	Cyclic Sync Velocity

0x6062 - Position Demand Value

This object provides the target position for Position PID controller, after the limits have been applied. Expressed in encoder increments.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x6062	0x00	Position Demand Value	INT32	RO	-	-	-	0x00	INC

0x6064 - Position Actual Value

Provides the actual position value read from the encoder, expressed in the output shaft reference frame. Expressed in encoder increments.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x6064	0x00	Position Actual Value	INT32	RO	TX	-	-	0x00	INC

0x6067 - Position Window

Sets the size of the position window within which the target position is considered to have been reached. This value is symmetrically added to both sides of the target position value. Expressed in encoder increments.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x6067	0x00	Position Window	INT32	RW	TX	-	-	0x00	INC

0x606B - Velocity Demand Value

Provides the target velocity value for the Velocity PID controller, after the limits have been applied. Expressed in RPM

Skip to main content

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x606B	0x00	Velocity Demand Value	INT32	RO	TX	-	-	0x00	RPM

0x606C - Velocity Actual Value

Provides the actual velocity value read from the encoder, expressed in the output shaft reference frame. Expressed in RPM.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x606C	0x00	Velocity Actual Value	INT32	RO	TX	-	-	0x00	RPM

0x606D - Velocity Window

Sets the size of the velocity window within which the target velocity is considered to have been reached. This value is symmetrically added to both sides of the target velocity value. Expressed in RPM.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x606D	0x00	Velocity Window	INT32	RW	TX	-	-	0x00	RPM

0x6072 - Max Torque

Configures the maximum allowed torque in the motor. The value is expressed in permille of rated torque. Example: rated torque of the motor is 1Nm, and the maximum is 2Nm. The Max Torque object should equal to 2000.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x6072	0x00	Max Torque	UNT16	RW	RX	yes	32767	0x00	-

0x6073 - Max Current

Configures the maximum allowed phase current in the motor. The value is expressed in permille of rated current. Example: rated current of the motor is 15A, and the maximum is 30A. The Max Current object should equal to 2000.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units

Skip to main content

0x6075 - Motor Rated Current

Configures the motor rated current expressed in mA. This object is a reference for parameters such as 0x6073 Max Current. The value should be taken from the motor's datasheet.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x6075	0x00	Motor Rated Current	INT32	RW	-	yes	1000000	0x00	mA

0x6076 - Motor Rated Torque

Configures the motor rated torque expressed in mNm. This object is a reference for parameters such as 0x6072 Max Torque. The value should be taken from the motor's datasheet.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x6076	0x00	Motor Rated Torque	INT32	RW	-	yes	1000000	0x00	mNm

0x6077 - Torque Actual Value

Provides the actual velocity value read from the encoder, expressed in the output shaft reference frame. Expressed in permille of 0x6076 Motor Rated Torque.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x6077	0x00	Torque Actual Value	INT16	RO	TX	-	-	0x00	-

0x6079 - DC Link Circuit Voltage

Provides the bus voltage measured by the motor controller in mV.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x6079	0x00	DC Link Circuit Voltage	UNT32	RO	RX	-	-	0x00	mV

0x607A - Target Position

Sets the target position for all motion modes.

Skip to main content

0x607D - Software Position Limit

Configures software limits that each new target position is checked against, and clipped if needed.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x607D	0x01	Min Position Limit	INT32	RW	RX	yes	-	0x00	inc

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x607D	0x02	Max Position Limit	INT32	RW	RX	yes	-	0x00	inc

0x6080 - Max Motor Speed

Sets the maximum allowed velocity of the actuator’s output shaft in both directions.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x6080	0x00	Max Motor Speed	UINT32	RW	RX	-	-	1000	RPM

0x6081 - Profile Velocity

Configures the target velocity for the profile position mode. If this value is greater than 0x6080 Max Motor Speed, it will be limited to Max Motor Speed.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x6081	0x00	Profile Velocity	UINT32	RW	RX	yes	-	0	RPM

0x6083 - Profile Acceleration

Configures the acceleration for profile position and profile velocity modes.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x6083	0x00	Profile Acceleration	UINT32	RW	RX	-	-	0	RPM/s

0x6084 - Profile Deceleration

Configures the deceleration for profile position and profile velocity modes.

Skip to main content

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x6084	0x00	Profile Deceleration	UINT32	RW	RX	-	-	0	RPM/s

0x60FF - Target Velocity

Sets the target velocity for all motion modes.

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x60FF	0x00	Target Velocity	INT32	RW	RX	-	-	0	RPM

0x6502 - Supported Drive Modes

Indicates the supported drive modes (binary value).

Index	Sub Index	Name	Data Type	SDO	PDO	NVM	Range	Default	Units
0x6502	0x00	Supported Drive Modes	INT32	RO	-	-	-	647	-

Migration to and from CANopen

Preparation

Download flasher programs from here:

- MD CANopen
- MD FDCAN Protocol

The migration is performed via flashing appropriate firmware onto the MD device. Necessary components for this procedure are:

- Candle or Candle HAT device
- MD driver
- PC or SBC with mdtool installed

Connect your candle device to the driver and power it. Make sure that candle device is connected via the USB to the host.

Skip to main content

> **Important**
>
> Only one device can be connected on the can line during the procedure, so all the drivers need to be updated individually.

Migration to CANopen

First step

When switching to CANopen the driver ID must be within the range of valid nodeIDs in CANopen. To set this up we usually use the lowest valid nodeID in our protocol which is 10. So the first command issued should be:

```
mdtool config can <previous_id> <new_id> <baudrate> <watchdog>
```

Example:

```
mdtool config can 100 10 1M 200
```

Second step

It is necessary to save this new ID to the persistent memory using save command as follows:

```
mdtool config save <id>
```

Example:

```
mdtool config save 10
```

Third step

Use flasher to flash firmware onto the board:

Skip to main content

```
<path_to_canopen_flasher> --id <id> --baud <baudrate>
```

Example:

```
./MAB_CAN_Flasher_CANopen_7fd0626 --id 10 --baud 1M
```

Migration from CANopen

Using the recovery procedure, the driver can be reverted to version with MD FDCAN protocol. The recovery command looks like this:

```
<path_to_fdcan_protocol_flasher> --id 9 --baud 1M --wait
```

Example:

```
./MAB_CAN_Flasher --id 9 --baud 1M --wait
```

After starting this command, the driver needs to be restarted manually and then the recovery re-flashing procedure will begin. The driver should then operate in MD FDCAN protocol mode.

If the nodeID was lower than 10 the driver should automatically assign itself an ID of 10.

Common Issues and FAQ

MDxx is not detected on “mdtool ping all” command

There might be several reasons why MD controllers are not showing up in the mdtool ping all command output. Please ensure you’ve check the things listed below:

1. Check the power supply - when powered the MDxx’s onboard LEDs should be blinking. The LEDs are located under the connectors near the PCB edge.
2. Check the LED blink pattern - if red and green LEDs are blinking one after another quickly in 4s

`id 9` and `wait` flag, ex.: `./MAB_CAN_Flasher --id 9 --baud 1M --wait`. If the red led is on and the green one is blinking in 1s intervals this means there is an error in the setup that can be checked using mdtool setup info. If only the green LED is blinking in 1s intervals the MD8xx should operate and be discovered without issues. In case it's not please check the cabling one more time.

How to check if my motor is operating properly

First thing to check is the `mdtool setup info` command output. If there are no errors (meaning the error field are empty or show 'ALL OK" message) the drive did not detect any issues by itself. The other thing is to make sure that the actuator runs smoothly - such that there is no excessive cogging torque when rotating. You can check it using `mdtool test move` command - for example by commanding 0 position and moving the rotor by hand - such test makes easy to determine if there is excessive cogging torque. You can also use the Python/C++ examples to rotate the motor. The last thing to check is the motion parameters - position velocity and torque. You can check them by looking at the `mdtool encoder` command output. If any of these quantities look suspicious feel free to contact us using: contact@mabrobotics.pl.

Motor terminals not soldered properly

In case you have ordered the MDxx controllers without the MAB assembly option you will have to make sure the controller is soldered correctly to the motor. Usually, hobby motors have multiple wires wound in parallel on each motor phase, and it is crucial to solder ALL wires to the controller. Leaving a single string of wire can lead to an imbalance between the phases, which in the best scenario will cause the calibration to fail and in the worst will cause large torque variations (large cogging torque).

> **Warning**
>
> Operating such an improperly configured motor can lead to hazardous situations for both the operator and the driver.

Failed calibration

The calibration can fail for several reasons, yet the most common one is just improperly soldered motor wires. In this case, you'll see the ERROR_CALIBRATION general error or ERROR_CALIBRATION and ERROR_PARAM_IDENT. These two errors will also show up when automatic parameter identification fails. In this case, rerunning the calibration should fix the issue. ERROR_POLE_PAIR_DET error is shown in case the automatic pole pair detection algorithm detected a different pole pair number (compared to the one form the *.cfg file) or it failed due to high rotor friction/external load, which stopped the rotor during the process.

The other most common reason is that the eccentricity calibration is interrupted by either a large load on the motor shaft or the encoder placed non-axially in regard to the magnet mounted on the motor shaft. In this case, you'll see the ERROR_CALIBRATION general error. To fix it be sure to unload the motor shaft completely, make it run smoothly, and make sure the controller is placed axially with respect to the magnet placed on the motor shaft.

Lack of FDCAN termination

Proper termination on the FDCAN bus is crucial, especially when the string of actuators is long. In case you see some communication errors, or the drives connected to your FDCAN bus string are not discovered correctly using MDtool be sure to check if the termination is present and working (the resistance between CANH and CANL lines should be 60 Ohms - two 120 Ohm resistors in parallel). The termination resistor is embedded in the CANdle device, and can be turned on/off with a physical switch.

> **Hint**
>
> Since version HW2.0 the termination resistor can be mounted on demand on the MD controllers. Please check out the `mdtool config can` command for more information.

Different FDCAN speeds between actuators

MD x CANdle ecosystem is not adopted for working with actuators of different FDCAN baudrates. Trying to control actuators with different baud rates on a common FDCAN bus can cause the communication to fail or not start at all. This is why it is crucial to make sure when you call the `mdtool ping all` command, all discovered MD controllers lie in a single baudrate category. If that's not the case, use the `mdtool config can` command to fix it.

Too-low torque bandwidth setting

When the torque bandwidth is set to a too low value it can cause the motor to behave improperly in highly dynamic scenarios, for example, impacts. Because with low torque bandwidth, the torque controller gains are set so that the controller is slow, it might not be able to keep up with the changing setpoint value. In order to fix this issue, you can calibrate the motor for a higher torque bandwidth frequency using `mdtool config bandwidth` command. This has a disadvantage connected to it - the higher the bandwidth the more audible noise you will hear coming from the motor.

Downloads

3D models

Simplified (lightweight) 3D *.STEP models of MABRobotics products can be found here.

Software (stable)

Main stable releases:

Date	CANdle lib	MDtool	pyCANdle	CANdle ROS	CANdle ROS2	CANdle device	MD Firmware	Manual
earlier	3.0	1.1.0	1.2.0	1.1.0	1.1.0	1.4	1.0	1.0
12.12.2022	3.1	1.2.1	1.3.1	1.2.0	1.2.0	2.0	2.0	1.1.0
05.04.2023	3.3.0	1.3.0	1.4.0	1.3.0	1.3.0	2.2.0	2.2.0	1.2.0

Skip to main content

MD update tool - MAB CAN Flasher

MAB_CAN_Flasher is a console application used to update the MD controller software using CANdle that can be downloaded at the top of this page. The MD firmware is contained in the MAB_CAN_Flasher application itself. To update the firmware connect the CANdle to the PC and the MD controller(s), and apply the power supply. You can make sure all the controllers are functional using MDtool and the `mdtool ping all` command before you proceed to update the controllers. After that, you are ready to run the update tool. We highly advise you to call `./MAB_CAN_Flasher --help` command on the first use to get acquainted with the available options.

> **Note**
>
> Make sure the MAB_CAN_Flasher can be executed. If not use the `sudo chmod +x ./MAB_CAN_Flasher` command.

Example use cases

`./MAB_CAN_Flasher --id 150 --baud 1M` - update the MDxx controller with id equal to 150, which current CAN speed is 1M (the default CAN speed is 1M). Example output of this command for an ak80-64 motor:

```
MAB@tutorial:~$ ./MAB_CAN_Flasher_ak80-64 --id 150 --baud 1M
[CANDLE] CANdle library version: v2.1
[CANDLE] Creating CANdle object.
[CANDLE] Reset successfull!
[CANDLE] CANdle ready.
[CANDLE] Entering bootloader mode...
[CANDLE] Detecting bootloader mode...
[CANDLE] Bootloader detected!
[CANDLE] 100% [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||] Downloading...
[CANDLE] Flashing Complete.
```

`./MAB_CAN_Flasher --all --baud 1M` - update all available MDxx controllers, whose current CAN speed is 1M (all controllers need to have the same speed). Example command output for two MD controllers:

Skip to main content

```
MAB@tutorial:~$ ./MAB_CAN_Flasher_ak80-64 --all --baud 1M
[CANDLE] CANdle library version: v2.1
[CANDLE] Creating CANdle object.
[CANDLE] Reset successfull!
[CANDLE] CANdle ready.
[CANDLE] Starting pinging drives...
[CANDLE] Found drives.
[CANDLE] 1: ID = 150 (0x96)
[CANDLE] 2: ID = 250 (0xfa)
[CANDLE] Flashing drive [150] [CANDLE] Entering bootloader mode...
[CANDLE] Detecting bootloader mode...
[CANDLE] Bootloader detected!
[CANDLE] 100% [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||] Downloading...
[CANDLE] Drive [150] updated successfully
[CANDLE] Flashing drive [250] [CANDLE] Entering bootloader mode...
[CANDLE] Detecting bootloader mode...
[CANDLE] Bootloader detected!
[CANDLE] 100% [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||] Downloading...
[CANDLE] Drive [250] updated successfully
[CANDLE] Summary:
[CANDLE] Update successful for drives: [150] [250]
```

Important

In case the update process is interrupted or the MD controller seems to be not functioning (fast periodical flashing of the LEDs), you can disconnect the power supply, call:

```
./MAB_CAN_Flasher --id 9 --baud 1M --wait
```

While the command is running connect the power supply. This command will wait for the bootloader response and try to recover the firmware. If the flashing does not occur in the first power cycle you can repeat it until the bootloader is detected. If using that procedure **only one driver** may be present on the can bus. An example output of the wait option for the ak80-64 motor is shown below:

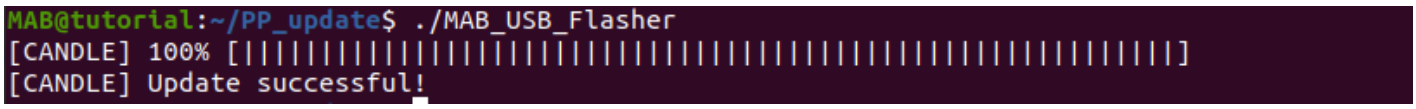
```
MAB@tutorial:~$ ./MAB_CAN_Flasher_ak80-64 --id 69 --baud 1M --wait
[CANDLE] CANdle library version: v2.1
[CANDLE] Creating CANdle object.
[CANDLE] Reset successfull!
[CANDLE] CANdle ready.
[CANDLE] Detecting bootloader mode...
[CANDLE] Bootloader mode could not be entered!
[CANDLE] Please power-cycle the drive...
[CANDLE] Detecting bootloader mode...
[CANDLE] Bootloader detected!
[CANDLE] 100% [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||] Downloading...
[CANDLE] Flashing Complete.
```

Skip to main content

CANdle update tool - MAB USB Flasher

MAB_USB_Flasher is a console application used to update the CANdle software using USB bus. Currently, only updates over USB are supported (updates over SPI and UART are not supported). To update, first turn off all applications that may be using CANdle, and simply run `./MAB_USB_Flasher`.

> **Note**
>
> Make sure the MAB_USB_Flasher can be executed. If not use the `sudo chmod +x ./MAB_USB_Flasher` command.



After a successful update, the CANdle device is ready.

CANopen Flashers

Those flashers are intended to be used with CANopen controllers.

> **Important**
>
> Candle and CandleHAT devices do not support CANopen protocol.

Versions:

Date	MD Firmware (CANopen)
22.08.2024 (latest)	2.4.0

Skip to main content

Changelogs

Latest [22.08.2024]

CANdle lib

- Changed license to MIT
- Minor bug-fixes

MDtool

- Changed license to MIT
- Minor bug-fixes

[23.07.2024]

MD Firmware

- Removed homing feature for safety, will be brought back in future releases
- Added GPIO general functionality - `GPIO Input`
- Relabeled Brake Mode as a GPIO mode, address and values stayed the same
- Added support for MD20 driver

CANdle lib

- Removed homing feature support for safety, will be brought back in future releases
- Added GPIO general functionality support
- Relabeled Brake Mode as a GPIO mode, address and values stayed the same

    - `runCalibrateOutpuEncoderCmd` → `runCalibrateOutputEncoderCmd`
    - `motorTorgueBandwidth` → `motorTorqueBandwidth`
- Added support for MD20 driver

MDtool

- Added compatibility with the newest CANdle lib functionalities