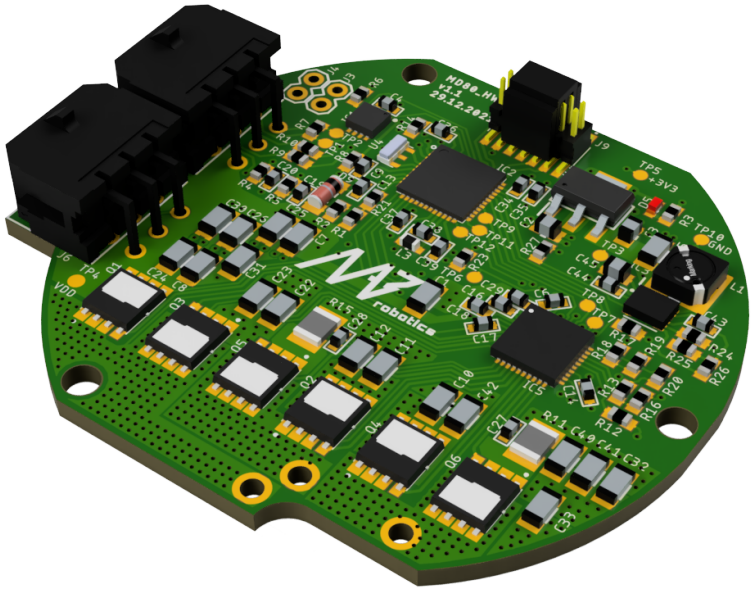
MAB ROBOTICS

MD80 x CANdle User Manual

rev 2.0 - 05.09.2022



www.mabrobotics.pl
contact@mabrobotics.pl

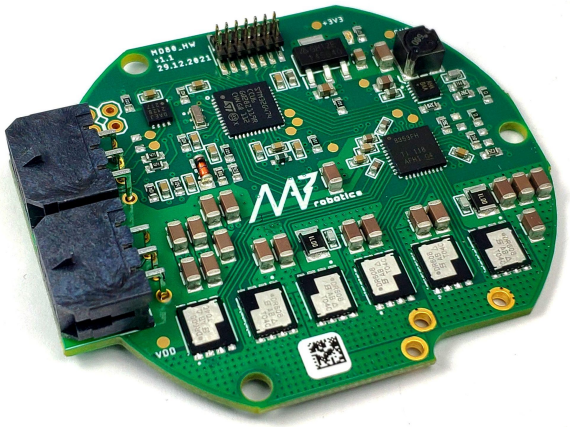
TABLE OF CONTENTS

1. Ecosystem overview

MD80 x CANdle is a system of brushless actuator controllers (MD80) and translator devices (CANdle) used for interfacing with them. MD80-based actuators can be used in advanced robotic projects like quadrupedal robots, robotic manipulators, exoskeletons, gimbals, and many more.

1.1. MD80

MD80 is a highly integrated brushless motor controller. It can be interfaced with a great variety of motors to turn them into advanced servo actuators. MD80 can work with both direct drive (no gearbox) and geared motors.

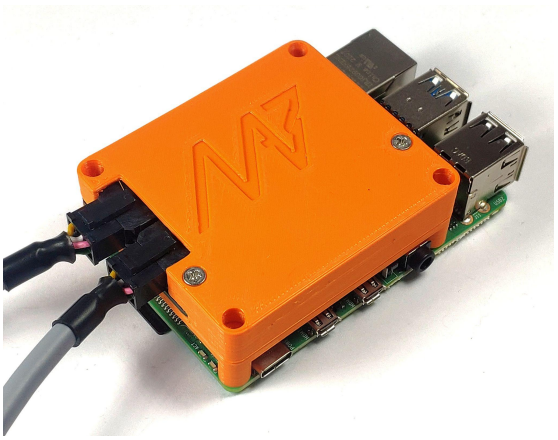


MD80 brushless controller

1.2. CANdle and CANdle HAT

CANdle (CAN + dongle) is a translator device between the host controller and the MD80 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.

CANdle Device



CANdle HAT device

1.3. Safety information

The instructions set out below must be **read carefully before the initial commissioning or installation** to raise awareness of potential risks and hazards, and to prevent injury to personnel and/or property damage.

To ensure safety when operating this servo drive, it is mandatory to follow the procedures included in this manual. The information provided is intended to protect users and their working area when using the device, as well as other hardware that may be connected to it.

Electric servo drives are dangerous: The following statements should be considered to avoid serious injury to individuals and/or damage to the equipment:

- Do not touch the power terminals of the device (supply and phases) as they can carry dangerously high voltages.
- Never connect or disconnect the device while the power supply is ON to prevent danger to personnel, the formation of electric arcs, or unwanted electrical contacts.
- Disconnect the drive from all power sources before proceeding with any wiring change.
- The surface of the device may exceed 100 °C during operation and may cause severe burns to direct touch.
- After turning OFF and disconnecting all power sources from the equipment, wait at least 10 minutes before touching any parts of the device, as it can remain electrically charged or hot.
- Do not remove the casing of the device.

The following statements should be considered to avoid serious injury to those individuals performing the procedures and/or damage to the equipment:

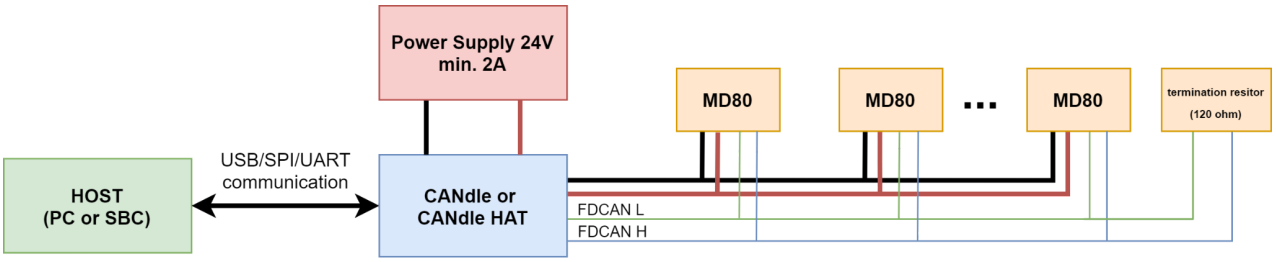
- Always comply with the connection conditions and technical specifications. Especially regarding wire cross-section and grounding.
- Some components become electrically charged during and after the operation.
- The power supply connected to this controller should comply with the parameters specified in this manual.
- When connecting this drive to an approved power source, do so through a line that is separate from any possible dangerous voltages, using the necessary insulation in accordance with safety standards.
- High-performance motion control equipment can move rapidly with very high forces. An unexpected motion may occur especially during product commissioning. Keep clear of any operational machinery and never touch them while they are working.
- Do not make any connections to any internal circuitry. Only connections to designated connectors are allowed.
- All service and maintenance must be performed by qualified personnel.
- Before turning on the drive, check that all safety precautions have been followed, as well as the installation procedures.

1.4. Operating conditions (MD80 and 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 °C
Maximum Humidity (Non-Operating)	up to 95%, non-condensing at 60 °C
Altitude (Operating)	-400 m to 2000 m

1.5. Hardware setup

A typical hardware connection/wiring scheme is presented in the picture below:



CANdle MD80-actuator string (USB bus)

CANdle HAT MD80-actuator string (SPI/UART bus using Raspberry Pi 4)

1.6. Before first use

Here are some things to look out for while playing with the MD80 x CANdle ecosystem for the first time:

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 have the ability to 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.

1.7. Quick startup guide

Please see the quick startup guide on our YouTube channel: Md80 x CANdle - Getting Started Tutorial

2. MD80

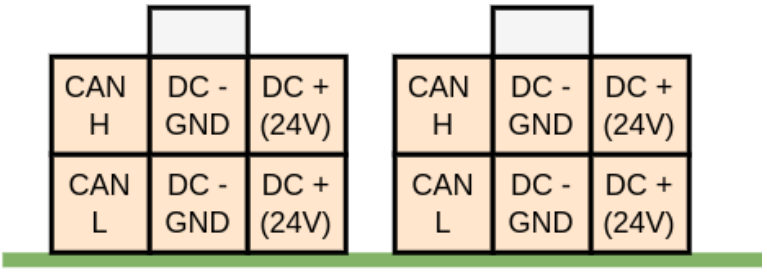
2.1. General parameters

MD80 is a brushless servo drive. It may come with a variety of motors and reductors, 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 servo drives have an integrated high-frequency position PID controller (at 1 kHz), velocity PID controller (at 5 kHz), and impedance controller (at 40 kHz), as well as a direct torque controller. MD80 also features a daisy-chaining mechanism, for easy connection of many drives in a single control network.

Parameter	Value
Input Voltage	18 - 28 VDC
Nominal Input Voltage	24 VDC
Max Input Current	20 A
Max Continuous Phase Current	20 A
Max Peak Phase Current (t = 4 s)	40 A
FDCAN Baudrate (adjustable)	1/2/5/8 Mbps
Position PID Controller Execution Frequency	1 kHz
Velocity PID Controller Execution Frequency	5 kHz
Impedance Controller ExecutionFrequency	40 kHz
Torque Control Execution Frequency	40 kHz
Torque Bandwidth (adjustable)	50 Hz - 2.5 kHz

2.2. Connectors pinout

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



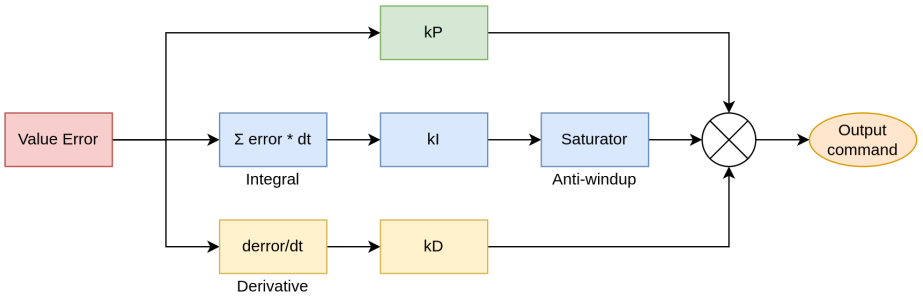
The colors of the corresponding wires in the Molex socket, as supplied by MAB (looking from the side of the wires):

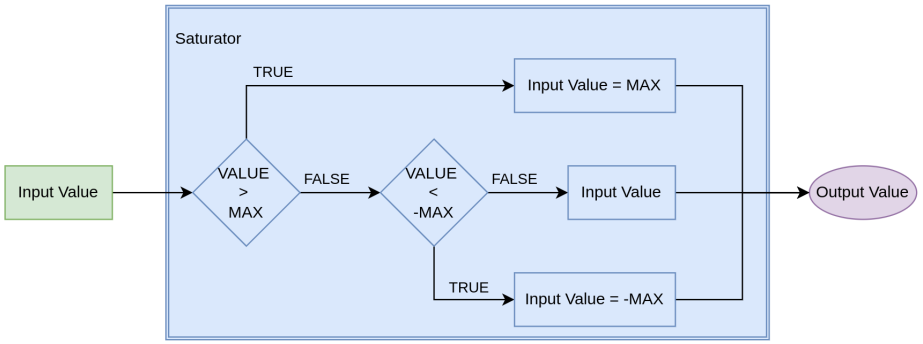
Yellow	White	Pink
Grey	Brown	Green

2.3. Control modes

TL;DR: MD80 x CANdle - motion modes

To control the motor shaft with the user's command MD80 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.



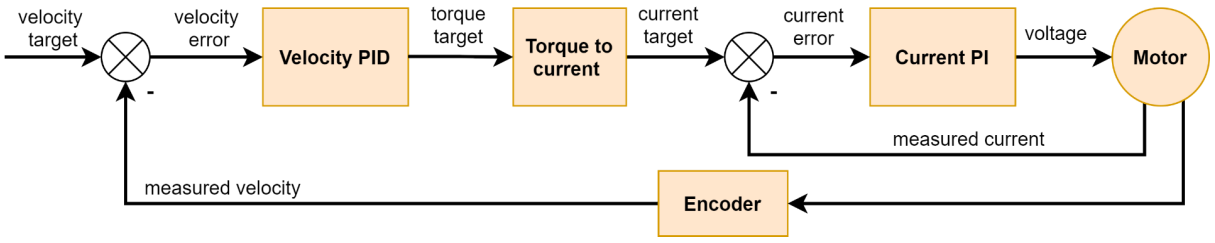


Velocity PID

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])
- Max output (in [Nm])

Velocity PID Controller structure



Position PID

Position PID mode is the most common controller mode used in industrial servo applications. In MD80, it is implemented as a **cascaded PID controller**. This means that the controller is working in two stages, firstly the position error is calculated, it is then passed to the Position PID, which outputs 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

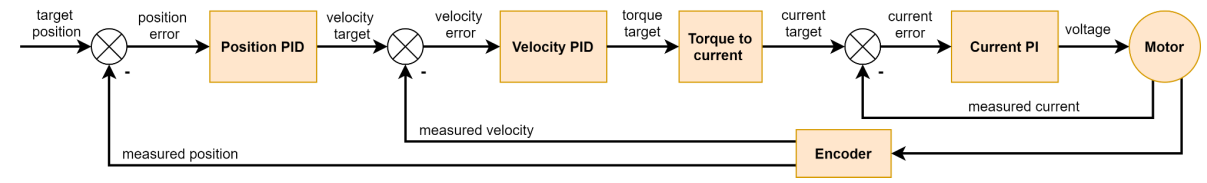
- Position Target (in [rad])
- kP (proportional gain)
- kI (integral gain)
- kD (derivative gain)
- I windup (maximal output of an integral part in [rad/s])
- Max output (in [rad/s])

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])
- Max output (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)

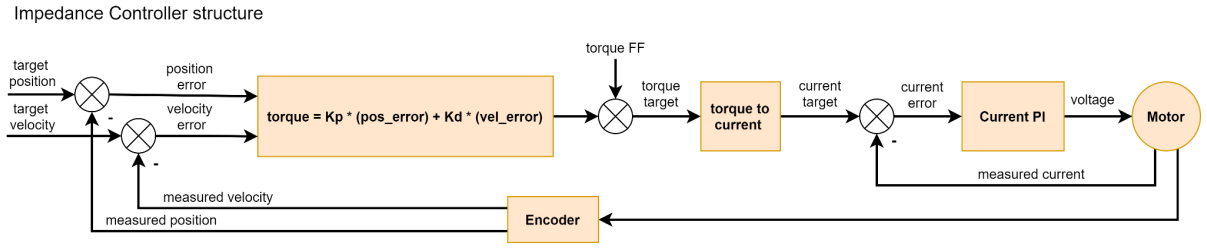
The torque output is proportional to the position error and velocity error and additionally supplemented with a torque command from the user. Here are some of the most common applications for this control mode:

- **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.

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



2.4. Controller tuning

TL;DR: 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 (link)

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.

Note: 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 MD80-based actuators.

Velocity PID

The velocity PID controller can be used to command different velocity profiles to the motor. This mode uses a regular PID controller architecture and has four user-defined parameters: kP, kI, kD, and windup. Since velocity readout is somewhat noisy, it is recommended to keep the kP value as low as possible and play with kI gain. It is necessary to find the sweet spot that makes the actuator response resistant to disturbances, but also not too noisy (too high kP gain may introduce oscillations). Usually, the kI gain is set to a higher value, however, do not treat it as a rule of thumb. Setting the kI gain to a too high value can cause an overshoot when the target velocity is changed rapidly. The kD gain may be used to partially overcome this issue. The windup parameter is used to limit the integral action which could potentially rise to high values when velocity error is present for longer periods.

Position PID

Position PID is the mode used when high positioning accuracy is required. Usually, no compliance is assumed in this mode, so the actuator will try to hold the read position as close to the commanded value as possible. One must be aware that position mode is actually made out of two PID controllers - the inner loop which is the velocity PID discussed earlier and the position PID working on top of the velocity loop. This is why it is essential to first take care of the velocity controller, considering the highest velocity that may occur in the system between the corresponding position commands in time. When both high and very low velocities are needed in a system it might be necessary to change the velocity PID gains on the fly, depending on the commanded velocity in each trajectory segment. When the velocity PID is ready the next step is to adjust the position PID gains so that a required actuator response is achieved. Since position readout is much less noisy compared to velocity it is recommended to first pick a kP value that will allow the motor to get to a setpoint position. In the next step kI and kD can be varied, together with the kI limiting factor (windup). One should remember the position PID output limit which is called MaxVelocity. This is a parameter that will limit the maximum commanded velocity and thus may limit actuator performance. It should be set to a value that is close to the actual trajectory segment maximum achievable velocity. Making it too high when very low velocities are required may result in oscillations.

Impedance PD

The impedance mode is relatively straightforward to get started with since there are only two main parameters that affect the response of the actuator. The easiest way is to think of a motor as a combination of a torsional spring with a damper, where kP is the spring constant, and kD is the damping coefficient. The higher the kP gain the more accurate positioning is achieved, but also, when it's set too high oscillations may be introduced. This is why a damping coefficient should be introduced. It makes the response more "smooth", usually less aggressive, and minimizes overshoot. It can be thought of as placing the motor in a viscous fluid where the viscosity of the fluid is the damping coefficient. This mode, however, can introduce steady-state error due to the lack of integral term. If high positioning accuracy is needed be sure to read about position PID mode.

Current PI

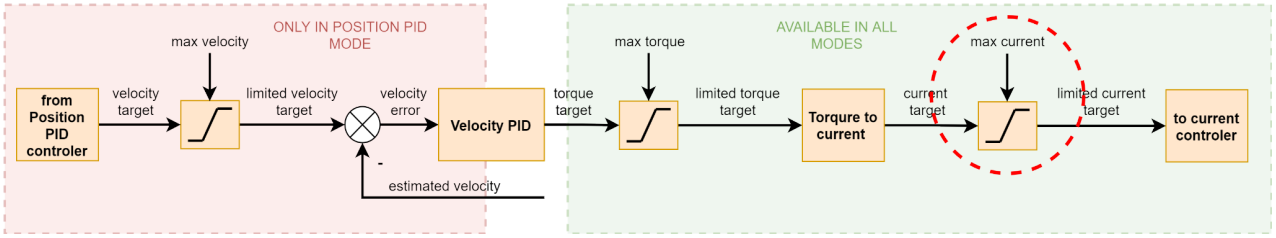
Current/torque PI is the lowest level controller. Its gains are not directly user-configurable, however, they can be modified using the bandwidth parameter. Please see the calibration section for more insight on the topic.

2.5. Safety limits

There are safety limits imposed on the maximum phase current as well as maximum torque and velocity to ensure a safe operation of the drive. Safety limits are there to protect the controller and the motor from overheating and the surrounding environment from too-powerful actuator movements.

Warning: setting the max current limit to above the maximum continuous current may damage the MD80 controller if the maximum torque is commanded for a prolonged period.

Let's start with the max current limit.



This setting limits the maximum current (and thus torque) the motor 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-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:

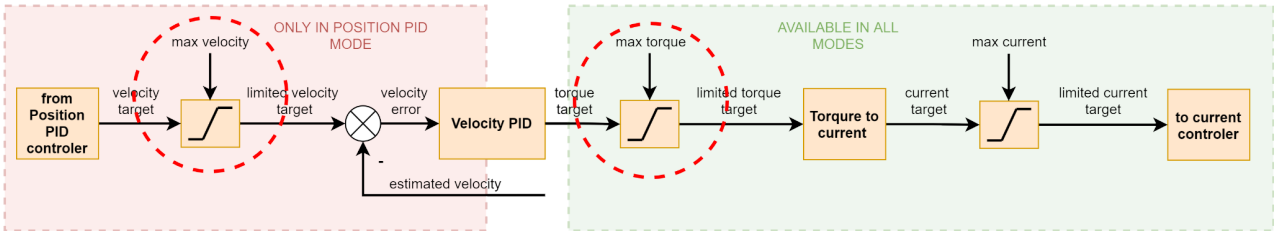
$$\tau = 2Nm$$
$$G_r = 9:1 -> 9$$
$$K_t = 0.091Nm/A_{\text{t}}$$

$$I[A] = \frac{2[Nm] \cdot \frac{1}{9}}{0.091[\frac{Nm}{A}]}$$
$$I[A] \approx 2.44A$$

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 put this value into the MD80 please refer to `mdtool config current command`. Don't forget to save it with the `mdtool config save command`.

The other limits are the max torque and max velocity parameters, which are set from the user script level.



These parameters allow restricting the output of high-level controllers - the position PID, velocity PID, and impedance PD controllers. These limits are applied before the max current limit, so even when set high they will not lead to a hazardous situation until the max current is fixed at a safe level. The max velocity limit is respected only in position PID mode, whereas the max torque limit is respected in all motion modes. Check out the controller tuning section for more information.

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.

2.6. FDCAN Watchdog

MD80 features 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 MD80.

Warning: we do not recommend disabling the CAN watchdog timer.

2.7. Measurements

MD80 is 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 an MD80 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.

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.

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 not as accurate due to excessive friction in the gearbox.

2.8. Calibration

Calibration is performed when the MD80 controller is first mounted to the motor. It has two stages during which it measures specific parameters of the setup.

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 diagnostic` *command. If the failure is essential to the motor's operation the MD80 will remain disabled till the next calibration attempt.*

Encoder eccentricity

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

Torque bandwidth

Even though the torque command on MD80 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 stage of calibration and it manifests itself as an audible sound.

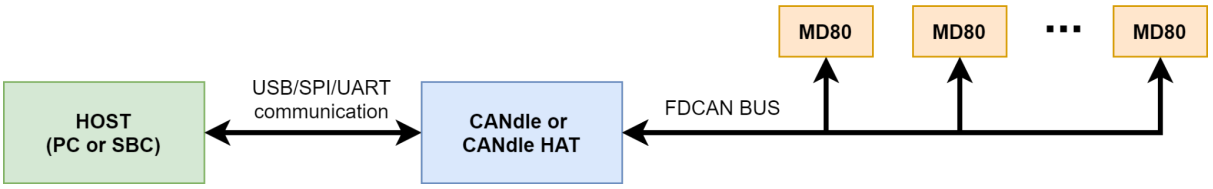
The torque bandwidth is set to 50 Hz by default. It can be set to anywhere from 50 Hz to 2.5 kHz, however it is important to note that higher torque bandwidth causes a higher audible noise level. Please see the `mdtool setup calibration` command for more details on calibrating the actuators.

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.

3. CANdle and CANdle HAT

CANdle is a translator device used to communicate between MD80 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 MD80 controllers is performed using FDCAN bus.



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

Note: currently CANdle supports only Linux operating systems.

3.1. Principle of operation

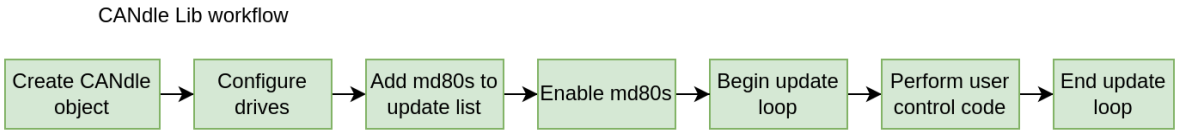
CANdle can work in two different modes: CONFIG and UPDATE. When in CONFIG mode, it works as a traditional translator device between two selected buses - USB/SPI/UART and FDCAN. This mode is used to set up the drives and prepare them for a low latency operation in the UPDATE mode. When the configuration is done the user calls `candle.begin()` which starts a low latency continuous connection with the MD80 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 there are three different speed modes, that select how fast the host should communicate with the CANdle device:

- NORMAL mode - up to 12 actuators on the bus, the slowest mode
- FAST1 mode - up to 6 actuators on the bus, the faster mode
- FAST2 mode - up to 3 actuators on the bus, the fastest mode

Each mode has a different CANdle <> host communication speed, thus the limit on the possible number of drives. Please see the latency section for maximum communication speeds in each mode.

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 md80 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.

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. Please note that scanning of the entire ID range will take about 2.5 seconds.

The next step is adding md80s 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 md80 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 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 md80s. 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 md80 vector will be updated with the most recent data from MD80s 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 md80 control mode. Latest data from md80's responses can be accessed with Md80::getPosition(), Md80::getVelocity(), Md80::getTorque(), Md80::getErrorVector().

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.

3.2. USB bus

The USB bus is the most common one, used in both CANdle and CANdle HAT. This is 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 realtime applications due to random host delays, the MD80 baudrate is not the limiting factor - you can set it to 1/2/5/8 Mbps and there will be no difference in the update rate.

Note: We highly recommend using the USB bus setup for the first run.

3.3. 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 MD80 controllers using CANdle HAT. Together with the RT-PATCHED kernel of the system, you will get the best performance.

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

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.

3.4. UART bus

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.

3.5. 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:

```
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
```

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
```

3.6. Latency

The latency was measured in a real scenario to get the most accurate results. A special flag was embedded into the MD80 command which the MD80 should return in the next response it sends. This way the whole route from the host, through CANdle, MD80 and back was profiled in terms of the delay. 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

VM PC Ideapad Gaming 3 AMD Ryzen 7 4800H							
50 samples test		mean [us]	stdev [us]	min delay [us]	max delay [us]	mean [Hz]	stdev [Hz]
priority normal	NORMAL	15976.8	2681.34	12615	30137	62.5907	8.99483
	FAST1	11778.2	3743.81	5923	19175	84.9029	20.4781
	FAST2	12706.3	4897.71	3855	22295	78.7009	21.8958
priority high	NORMAL	12924.6	995.128	11641	16025	77.372	5.53137
	FAST1	6884.62	1255.78	5210	10773	145.251	22.4073
	FAST2	5806.88	2658.18	2891	18838	172.21	54.0768

RPI 3b+ RT PATCH (4,19,71-rt24-v7+)							
50 samples test		mean [us]	stdev [us]	min delay [us]	max delay [us]	mean [Hz]	stdev [Hz]
priority normal	NORMAL	16203,7	475,925	15746	18418	61,7144	1,76092
	FAST1	9037,86	1046,21	7741	12605	110,646	11,4793
	FAST2	6909,86	1328,09	4444	10699	144,721	23,3313
priority high	NORMAL	15828,6	562,806	14909	16800	63,1769	2,16921
	FAST1	7526,66	356,193	7090	8751	132,861	6,00343
	FAST2	4565,58	221,033	4214	5885	219,03	10,1142

UART

RPI 3b+ RT PATCH (4,19,71-rt24-v7+)							
50 samples test		mean [us]	stdev [us]	min delay [us]	max delay [us]	mean [Hz]	stdev [Hz]
priority normal	NORMAL	NOT STABLE	NOT STABLE	NOT STABLE	NOT STABLE	NOT STABLE	NOT STABLE
	FAST1	NOT STABLE	NOT STABLE	NOT STABLE	NOT STABLE	NOT STABLE	NOT STABLE
	FAST2	NOT STABLE	NOT STABLE	NOT STABLE	NOT STABLE	NOT STABLE	NOT STABLE
priority high	NORMAL	12663,5	203,012	12538	13476	78,9669	1,24596
	FAST1	7695,86	100,132	7608	8030	129,94	1,66895
	FAST2	4241,02	97,3461	4117	4581	235,792	5,29081

SPI							
RPI 3b+ RT PATCH (4,19,71-rt24-v7+)							
50 samples test		mean [us]	stdev [us]	min delay [us]	max delay [us]	mean [Hz]	stdev [Hz]
priority normal	NORMAL	2891,26	818,706	1780	4955	**345,87**	76,3257
	FAST1	1375,6	371,937	780	2155	**726,956**	154,721
	FAST2	1100,54	329,765	630	2523	**908,645**	209,493
priority high	NORMAL	2311,58	600,697	1704	4611	**432,605**	89,2306
	FAST1	1378,22	333,762	789	2096	**725,574**	141,455
	FAST2	936,48	225,68	520	1321	**1067,83**	207,362

Each mode was tested with the maximum number of actuators i.e NORMAL mode with 12 actuators, FAST1 mode with 6 actuators, and FAST2 with 3 actuators. As can be seen, the SPI bus gives the best results, reaching over 1 kHz of communication speed in FAST2 mode and 3 actuators. The UART and USB buses generally give similar results. The division between priority normal and priority high was accomplished using a script that changes the scheduler priority of the running test program:

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

This script changes the priority only when the program is already running (otherwise it will not work). It can be used when your program cannot be run directly with sudo - for example it is useful when dealing with ROS nodes.
You can also embed the following snippet in your C++ code if you can run it with sudo directly:

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

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.

4. Software Pack

The MD80 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.

4.1. 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 MD80 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 MD80 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:

```
cmake .. -DCANDLE_BUILD_STATIC=TRUE
```

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 found on PyPi: https://pypi.org/project/pyCandleMAB/ and installed using pip:

```
sudo 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.

4.2. MDtool

MDtool is a console application for configuring and performing basic diagnostics on MD80 drives via CANdle. The application is available as a standalone executable and as a .deb package. The program 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. To install after the download simply call:

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

Be sure to call `./mdtool bus <SPI/UART/USB>` to configure MDtool for a desired communication bus before first use.

mdtool bus command <bus>

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.

mdtool ping command <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 command <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 <ID> command.

mdtool config can command <current ID> <new ID> <baud> <watchdog period>

This command is used to change MD80's parameters such as CAN ID, CAN speed, and CAN watchdog. For more information on CAN watchdog please refer to section FDCAN Watchdog Timer.

Note: this setting has to be saved to be preserved after power down! Please see the config save <ID> command.

mdtool config current command <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. The absolute maximum value is 40 A.

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

Note: this setting has to be saved to be preserved after power down! Please see the config save <ID> command.

mdtool config save command <ID>

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 command <ID> <torque bandwidth>

The calibration is always performed in the factory, however, if for some reason you'd like to rerun it the calibration command can be used to do just that. 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 diagnostic command <ID>

This command returns basic diagnostic information:

- Drive's ID
- Current CAN speed
- Current Position
- Current Velocity
- Current torque
- Current temperature (0* if the thermistor is not mounted)
- Errors

Reading the errors is the easiest way of debugging possible problems with the drive.

Error code	Description
ERROR_BRIDGE_OCP	Overcurrent was detected by the MOSFET driver
ERROR_BRIDGE_FAULT	A general fault of the MOSFET driver
ERROR_OUT_ENCODER_E	Output encoder general error
ERROR_OUT_ENCODER_COM_E	Output encoder communication error
ERROR_PARAM_IDENT	Error during calibration of the drive, during the resistance and inductance measurements
ERROR_UNDERVOLTAGE	Under voltage detected
ERROR_OVERVOLTAGE	Overvoltage detected
ERROR_TEMP_W	Overheat warning
ERROR_TEMP_SD	Overheat shutdown
ERROR_CALIBRATION	A general error during calibration
ERROR_OCD	Overcurrent detected
ERROR_CAN_WD	CAN watchdog triggered

mdtool blink command <ID>

This command is mostly used to find an MD80 drive on a long CAN bus using its ID – the command makes the drive flash its onboard LEDs for easy identification.

mdtool test command <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 encoder command <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.

4.3. CANdle ROS/ROS2 nodes

TL;DR: MD80 x CANdle - ROS/ROS2 startup guide

While C++ API is the most flexible way of interfacing with CANdle/MD80, 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
```

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 MD80 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
[ INFO] [1661492668.839748424]: candle_ros_node has started.
```

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:

```
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
[ INFO] [1661493518.632029451]: candle_ros_node has started.
[CANDLE] Added Md80.  [OK]
[CANDLE] Added Md80.  [OK]
[CANDLE] Set current speed mode to FAST2
```

According to the status messages we have added two MD80 actuators. As you can see the speed mode (NORMAL/FAST1/FAST2) is selected automatically based on the number of added drives.

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", "IMPEDANCE"]}"
```

Should produce:

```
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 md80 mode set before enabling.

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

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

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, 800], kp:[0.25, 1.0], kd:[0.1, 0.05], max_output:[2.0, 2.0]}"
```

Example set up of custom gains for POSITION PID mode:

```
rostopic pub /md80/position_command candle_ros/PositionPidCommand "{drive_ids: [200, 800], position_pid: [{kp: 40.0, ki: 0.5, kd: 0.0, i_windup: 10, max_output: 3.0},{kp: 20.0, ki: 0.5, kd: 0.0, i_windup: 10, max_output: 3.0}], velocity_pid: [{kp: 0.2, ki: 0.3, kd: 0.0, i_windup: 2.0, max_output: 2.0}, {kp: 0.1, ki: 0.1, kd: 0.0, i_windup: 1, max_output: 2.0}]}"
```

Example set up of custom gains for VELOCITY PID mode:

```
rostopic pub /md80/velocity_command candle_ros/VelocityPidCommand "{drive_ids: [200, 800], velocity_pid: [{kp: 0.2, ki: 0.3, kd: 0.0, i_windup: 2.0, max_output: 2.0}, {kp: 0.1, ki: 0.1, kd: 0.0, i_windup: 1, max_output: 2.0}]}"
```

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], target_position:[3.0, -3.0], target_velocity:[0.0, 0.0], target_torque:[0, 0]}"
```

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 youre going to send commands related to the node.

First let's run the node with arguments that fit your MD80 x CANdle setup. The general syntax is:

```
ros2 run candle_ros2 candle_ros2_node <BUS> <FDCAN 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: [200,800]}"
```

Should produce the following output:

```
response:
candle_ros2.srv.AddMd80s_Response(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:

```
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]
[CANDLE] Set current speed mode to FAST2
```

According to the status messages we have added two MD80 actuators. As you can see the speed mode (NORMAL/FAST1/FAST2) is selected automatically based on the number of added drives.

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 "{drive_ids: [200, 800], mode:["IMPEDANCE", "IMPEDANCE"]}"
```

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 "{drive_ids:[200,800]}"
```

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.

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 md80 mode set before enabling.

```
ros2 service call /candle_ros2_node/enable_md80s candle_ros2/srv/GenericMd80Msg "{drive_ids:[200, 800]}"
```

```
ros2 service call /candle_ros2_node/disable_md80s candle_ros2/srv/GenericMd80Msg "{drive_ids:[200, 800]}"
```

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: [200, 800], kp: [1.0,1.0]], kd: [0.001,0.001], max_output: [1.0, 1.0]}"
```

Example set up of custom gains for POSITION PID mode:

```
ros2 topic pub /md80/position_pid_command candle_ros2/msg/PositionPidCommand "{drive_ids: [150, 350], position_pid: [{kp: 40.0, ki: 0.5, kd: 0.0, i_windup: 10, max_output: 3.0},{kp: 20.0, ki: 0.5, kd: 0.0, i_windup: 10, max_output: 3.0}], velocity_pid: [{kp: 0.2, ki: 0.3, kd: 0.0, i_windup: 2.0, max_output: 2.0}, {kp: 0.1, ki: 0.1, kd: 0.0, i_windup: 1, max_output: 2.0}]}"
```

Example set up of custom gains for VELOCITY PID mode:

```
ros2 topic pub /md80/velocity_pid_command candle_ros2/msg/VelocityPidCommand "{drive_ids: [200, 800], velocity_pid: [{kp: 0.2, ki: 0.3, kd: 0.0, i_windup: 2.0, max_output: 2.0}, {kp: 0.1, ki: 0.1, kd: 0.0, i_windup: 1, max_output: 2.0}]}"
```

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], target_position: [3.0, 3.0], target_velocity: [0.0, 0.0], target_torque: [0.0, 0.0]}"
```

4.4. MD80 update tool - MAB CAN Flasher

MAB_CAN_Flasher is a console application used to update the MD80 controller software using CANdle. When an update is released our engineers will prepare a MAB_CAN_Flasher application and send it to you. The MD80 firmware is contained in the MAB_CAN_Flasher application itself, thus it can upload only one firmware version for one motor type. In case you have multiple motor types in your application you will receive multiple MAB_CAN_Flasher programs (one for each motor type). The application can update a single MD80 device, or multiple, same motor-type devices.

First, connect the CANdle to the PC and the MD80 controller(s), and apply the power supply. You can make sure all the controllers are functional using MDtool and the ping 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: In case the motor config (motor type) is changed with new firmware using MAB CAN Flasher it is mandatory to call the `mdtool setup calibration` command. Only after recalibrating the new motor config will be loaded.

Example use cases

`./MAB_CAN_Flasher --id 150 --baud 1M` - update the md80 controller with id equal to 69, 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 md80 controllers, whose current CAN speed is 1M (all controllers need to have the same speed). Example command output for two md80 controllers:

```
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]
```

In case the update process is interrupted and the md80 controller seems to be broken, you can disconnect the power supply, call:

```
./MAB_CAN_Flasher --id 69 --baud 1M -wait
```

and 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. The 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.
```

4.5. 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). When an update is released our engineers will prepare a MAB_USB_Flasher application and send it to you. To update, first turn off all applications that may be using CANdle, and simply run `./MAB_USB_Flasher`.

```
MAB@tutorial:~/PP_update$ ./MAB_USB_Flasher
[CANDLE] 100% [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||]
[CANDLE] Update successful!
```

After a successful update, the CANdle device is ready.

5. Common Issues and FAQ

5.1. How to check if my motor is operating properly

First thing to check is the mdtool setup diagnostic command output. If there are no errors (meaning the error field shows 0x00, or it shows only the CAN watchdog error) 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 command). The last things to check are the motion parameters - position velocity and torque. You can check them by printing from C++ or Python script. If any of these quantities look suspicious feel free to contact us using: contact@mabrobotics.pl.

5.2. Motor not soldered properly

In case you have ordered the MD80 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.

5.3. 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.
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.

5.4. 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 series). Remember, you only need to place a single termination resistor on the end of the string when using CANdle. The other resistor is embedded in the CANdle device.

5.5. Different FDCAN speeds between actuators

MD80 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 MD80 controllers lie in a single baudrate category. If that's not the case, use the `mdtool config can` command to fix it.

5.6. 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. This has a disadvantage connected to it - the higher the bandwidth the more audible noise you will hear coming from the motor.

Revision history

Revision	Release date	Changes
v1.0	December 2021	Preliminary release
v2.0	September 2022	CANdle HAT release + software pack update