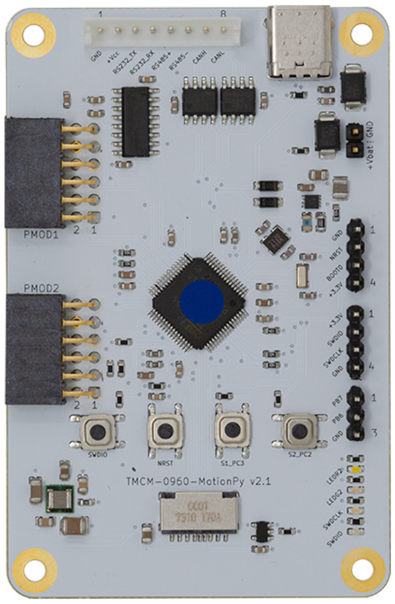
AN061: TMCM-0960-MotionPy V21 with TMCL-Modules

Document Revision V1.10 • 2021-JUL-08

The TMCM-0960-MotionPy is a general purpose MicroPython platform for motion control applications. It comes with various interfaces onboard and is compatible with many Trinamic Modules, making it the swiss-army-knife for engineers. This application note describes how to use the MotionPy together with TMCL-Modules and external TMCL-Masters.



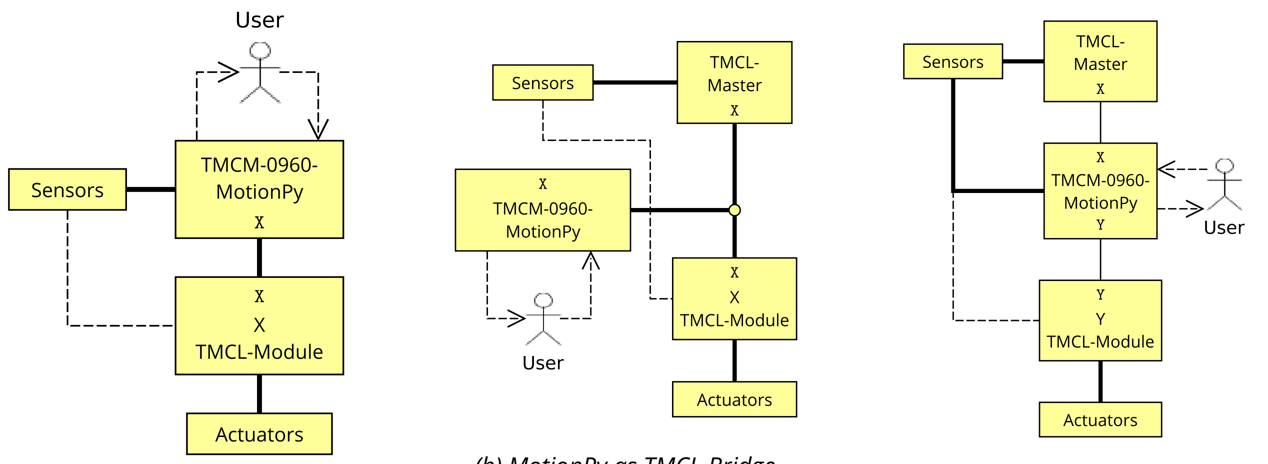
Contents

TRINAMIC
MOTION CONTROL
Now part of Maxim Integrated

1 Introduction

The TMCM-0960-MotionPy is the swiss-army-knife for prototyping, testing and debugging motion control applications. It serves as a MicroPython platform for user-scripts and comes equipped with all the required libraries and hardware peripherals onboard. This way, it is compatible with most TMCL-Modules via RS232, RS485 and CAN bus systems.

It can serve in one of three operation modes within a TMCL system:



(a) MotionPy as TMCL-Master

(b) MotionPy as TMCL-Bridge

(c) MotionPy as TMCL-Slave

Figure 1: Modes of operation

When acting as a TMCL-Master, the MotionPy directly controls one or multiple TMCL-Modules, based on user and sensor input.

When acting as a TMCL-Bridge, the MotionPy bridges between interfaces. By doing so, it will act invisibly for TMCL-Masters and TMCL-Modules on the affected busses. This can be used to forward and analyze TMCL-Commands sent over different bus systems.

When acting as a TMCL-Slave, the MotionPy itself can be controlled by external TMCL-Masters on one end, hiding potentially complex application with multiple TMCL-Modules on the other end. This way, abstract applications can be implemented on the TMCL-Master with the MotionPy handling the individual TMCL-Modules.

2 Getting Started

This is a brief guide on how to get started using the MotionPy with a TMCL-Module.

First, check out the required hardware:

- TMCM-0960-MotionPy Board
- USB-C cable
- TMCL-Module of choice (e.g. TMCM-1270, as used in this application note)
- Cable harness for power and bus signals

- (optional) SWD interface

Secondly, check out the required software on the workstation:

- Python 3
- Git
- Windows: PuTTY
- Linux: screen (installable via package manager)

2.1 Wiring up the MotionPy with the TMCL-Module

TMCL-Modules with various interfaces and the power lines are supposed to be connected on the onboard 8-pin JST-EH connector. It consists of the following pins:

- *GND*: Ground
- *+VCC*: Positive voltage up to +36V
- *TX*: TX signal for RS232
- *RX*: RX signal for RS232
- *RS485+*: Positive differential signal for RS485
- *RS485-*: Negative differential signal for RS485
- *CANH*: High CAN signal
- *CANL*: Low CAN signal

However, when using just a single bus interface, the connectors for the other interfaces can remain disconnected. For details, take a look into the hardware manual. For this application note, we are specificially describing the setup with a TMCM-1270 via CAN interface. Figure 2 shows an example wiring for that case. Note that the power supply is shared between the driver module and the MotionPy and should not exceed 50 V. For signal stability on the CAN bus, a termination resistor of 120 Ohm should be used.

2.2 Preparing the MicroPython firmware

If you already have a fully prepared MotionPy and do not want to build and flash the firmware on your own, you can skip the following steps.

1. Build the PYBv11 STM32 firmware. The detailed guide is available in the MicroPython repository.

```
cd micropython
PYTHON=python make -C mpy-cross
cd ports/stm32
PYTHON=python make submodules
PYTHON=python make BOARD=PYBV11
```

   Assuming `python` is the linked Python binary.

2. Connect your SWD interface as labeled on the PCB and flash the `micropython/ports/stm32/build/firmware.hex` file.

3. Reset the board with a `low` signal on the `NRST` pin or power cycle.

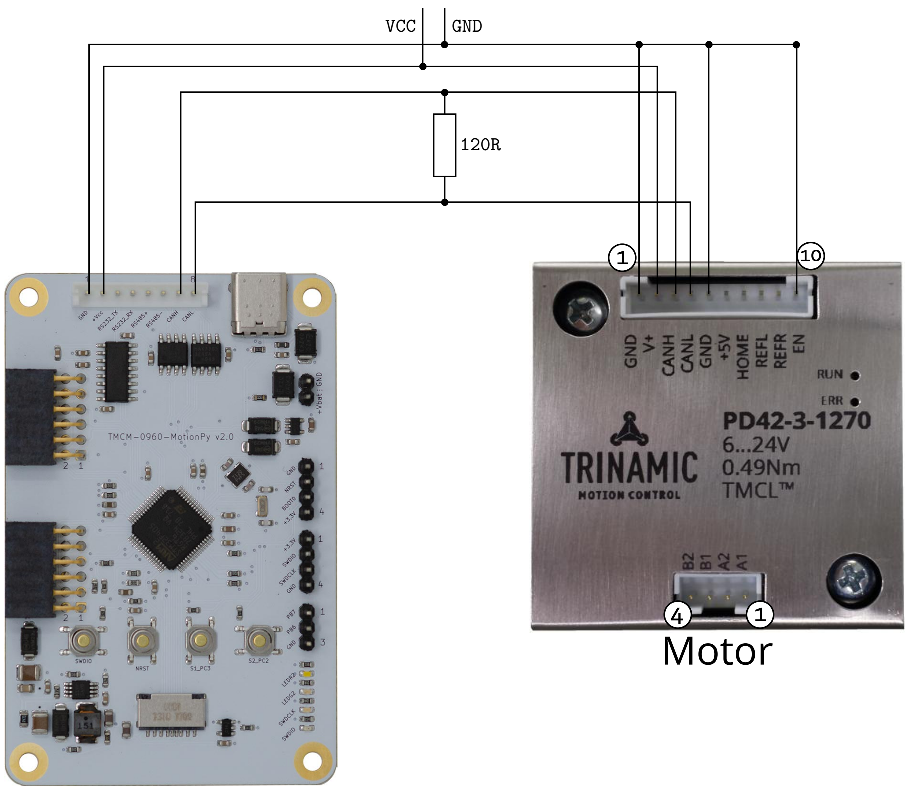


Figure 2: Example wiring with TMCM-1270

If you are developing on Windows, the following additional steps are required:

1. (Re-)attach the board to the PC, without the SD card inserted in it.
2. The boards internal flash should be mounted as mass storage device directly. Install the Serial Port Drivers `pybcdc.inf` from the root of the attached storage to your system.
3. Insert the SD card in the board and reattach it.

2.3 Installing the libraries

1. Recursively clone the PyTrinamicMicro library.

```
git clone https://github.com/trinamic/PyTrinamicMicro.git --recurse-
    ↪ submodules
```

2. Invoke the installation script install.py. For full installation, run the command

```
python install.py D:\
```

assuming `python` is properly linked to your python binary and `D` is the mounted flash / SD card. As installing incrementally is not supported at the moment, invoke

```
python install.py D:\ -c
```

if you have already an installation in that target directory.

3. Move the `main.py` and `boot.py` scripts for the MotionPy platform manually out of `PyTrinamicMicro/platforms/motionpy` to the root of the mounted storage.

4. Soft-Reset MicroPython by restarting it, or hard-reset the microcontroller with a `low` signal on the `NRST` pin, or power cycle.

2.4 Connecting and first steps

Connecting to an attached MotionPy on standard operating systems is simple.

1. Connect to the serial port of the attached MotionPy via terminal.
   Windows: Connect to `COMX` (determine `X` in the Device Manager) via PuTTY → Serial
   Linux: Connect to `/dev/ttyACMX` (determine `X` in the mounted devices in `/dev`) via

```
chmod 777 /dev/ttyACMX
screen /dev/ttyACMX
```

2. Now, being in the Python shell, any MicroPython compatible statement can be interpreted. The PyTrinamicMicro package can be used. By default, all example scripts can be executed via standard commands, i.e.

```
exec(open("PyTrinamicMicro/platforms/motionpy/examples/io/blinky.py").
    ↪ read())
```

   Example scripts can also be linked manually by the user in the `MotionPy` configuration class, and executed via shortcut, i.e.

```
exec(MP.script("blinky"))
```

3. Review the core configuration classes `PyTrinamicMicro` (for platform independent configuration) and `MotionPy` (for platform dependent configuration) and change them how you like. Various logging and quality-of-life options are possible there.

3 Structure

The software used to work with TMCL-Modules on the MotionPy is structured in multiple open-source libraries:

- MicroPython
- PyTrinamic
- PyTrinamicMicro
- MotionPy (part of PyTrinamicMicro)

PyTrinamic and PyTrinamicMicro for abstract motion control logic, MicroPython as the base platform and MotionPy as the actual implementation for the specific board.

4 Modes of operation

As introduced in the introduction, the MotionPy is designed to be used in the three use cases depicted in Figure 1.

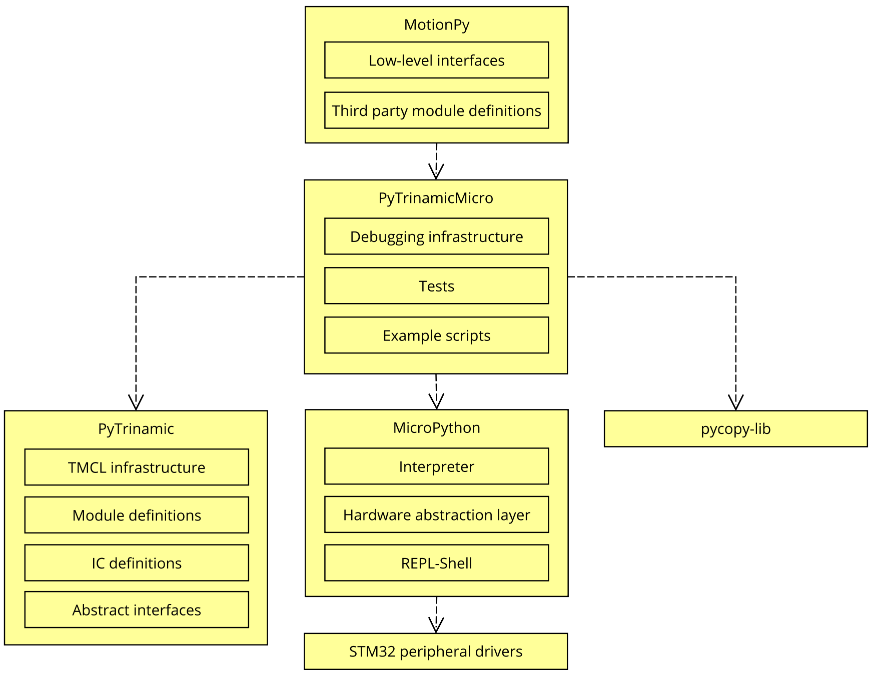


Figure 3: Libraries and its dependencies

4.1 TMCL-Master

The TMCL-Master functionality is already given by the `PyTrinamic` library via the central `tmcl_interface`, handling all the requests and replies when communicating with TMCL-Slaves.

Additionally, the module definitions of PyTrinamic can be used as a wrapper when working with known modules. That way, setting and getting axis parameters as well as rotating motors becomes a trivial task. Just wire up the TMCL-Module with the required power and bus signal cables to the MotionPy, import and initialize the required modules and feel free to control the module.

Listing 1: Rotate a TMCM-1270 via CAN

```
from PyTrinamic.modules.TMCM1270.TMCM_1270 import TMCM_1270
from PyTrinamicMicro.platforms.motionpy.connections.can_tmcl_interface
    ↪ import can_tmcl_interface

con = can_tmcl_interface()
module = TMCM_1270(con)

module.rotate(0, 1000)
time.sleep(5)
module.stop(0)

con.close()
```

4.2 TMCL-Bridge

When using the MotionPy as a TMCL-Bridge, TMCL-Datagrams can be forwarded between different bus systems, and even analyzed on the way. This makes the MotionPy applicable as USB-to-X adapter and X-to-Y bus analyzer.

Listing 2: Forward TMCL-Datagrams from USB to CAN, analyze through callbacks

```
def request_callback(request):
  global request_command
  request_command = request.command
  return request

def reply_callback(reply):
    if(request_command != TMCL.COMMANDS["GET_FIRMWARE_VERSION"]):
        reply.calculate_checksum()
    return reply

host = usb_vcp_tmcl_interface()
module = can_tmcl_interface()
bridge = TMCL_Bridge(host, [{"module":module, "request_callback":
    ↪ request_callback, "reply_callback":reply_callback}])

while(not(bridge.process(request_callback=request_callback, reply_callback=
    ↪ reply_callback))):
    pass
```

Listing 2 shows a code snippet out of the USB-to-CAN example implementation. After initialization of the single interface towards the external TMCL-Master and the probably multiple module interfaces (here it is just one), `bridge.process` is called periodically, handling and forwarding newly fetched requests and replies to the correct destinations. In the intermediate steps, e.g. when a new request comes in, additional actions can be taken in the callback function defined in the main application (e.g. logging of the TMCL-Datagram).

4.3 TMCL-Slave

As a third operational mode, the MotionPy can act as a single TMCL-Slave, making additional modules of the next level transparent against external TMCL-Masters.

Listing 3: Act as TMCL-Slave via USB

```
con = usb_vcp_tmcl_interface()
slave = TMCL_Slave_Bridge(MODULE_ADDRESS, HOST_ADDRESS, VERSION_STRING,
    ↪ BUILD_VERSION)

while(not(slave.status.stop)):
    if(con.request_available()):
        request = con.receive_request()
        if(not(slave.filter(request))):
            continue
        reply = slave.handle_request(request)
        con.send_reply(reply)
```

Listing 3 shows a code snippet out of the TMCL-Slave example implementation via USB. The general workflow of the TMCL request handling is as follows:

1. Initialization of the used interface, the instance of the slave implementation and the used hardware.
2. If a TMCL-Request is available: Handle it in the TMCL-Slave implementation and return the answer. Parameters and status get updated during the procession in the TMCL-Slave.
3. Check the updated parameters and flags and take the corresponding actions.
4. Update all parameters from external sources.

5. Go to 2. Repeat until stop-flag is set.
6. Deinitialize used components and interfaces.

This loop and external module handling is supposed to be done in the main application. Take the `tmcl_slave_usb` example script for reference. The actual parameter handling is done in the TMCL-Slave implementation, which is generally a descendant of the `TMCL_Slave` class. Therefore, please use `tmcl_slave_motionpy` as a reference.

5 Disclaimer

TRINAMIC Motion Control GmbH & Co. KG does not authorize or warrant any of its products for use in life support systems, without the specifc written consent of TRINAMIC Motion Control GmbH & Co. KG. Life support systems are equipment intended to support or sustain life, and whose failure to perform, when properly used in accordance with instructions provided, can be reasonably expected to result in personal injury or death.

Information given in this application note is believed to be accurate and reliable. However, no responsibility is assumed for the consequences of its use nor for any infringement of patents or other rights of third parties that may result from its use.

Specifications are subject to change without notice. All trademarks used are property of their respective owners.

6 Revision History

Version	Date	Author	Description
V1.00	2021-MAY-07	LK	Initial version.
V1.10	2021-JUL-09	SK	Changes for V21 board version: +VCC changed to +36V max, board image updated

Table 1: Document Revision