
Tinymovr

Yannis Chatzikonstantinou

May 08, 2024

CONTENTS:

1	From Unbox to First Spin	3
1.1	Preparation	3
1.2	Installing and Launching Studio	3
1.3	Installing on Raspberry Pi	4
1.4	Calibrating using the Studio GUI	4
1.5	Testing Position Control using the Studio GUI	4
2	Hardware Overview	5
2.1	R5.x	5
2.2	M5.x	8
2.3	R3.x	10
3	Hardware Setup	13
3.1	Requirements	13
3.2	Supported Motor Types	13
3.3	Mechanical Setup	14
3.4	Electrical Setup	15
3.5	Connecting Motor	16
3.6	Connecting Data	17
3.7	Connecting Power	17
3.8	Connecting Multiple Nodes (Daisy-Chaining)	17
4	Studio Installation	19
4.1	Preparation	19
4.2	Using pip	19
4.3	Using git clone	20
5	Studio Usage	21
5.1	Launching Studio	21
5.2	Alternative Adapters/Firmwares	22
5.3	Compatibility	22
5.4	Custom Device Specs	23
5.5	Issuing Commands in CLI	23
5.6	Multiple Instances	23
5.7	Command-line options	24
5.8	Units	25
5.9	Socketcan & Linux	26
6	Integrating Tinymovr	27
6.1	Use with Python	27
6.2	BusRouter API	28

6.3	Error Codes	29
7	Comm Interfaces	31
7.1	CAN Bus	31
7.2	UART	31
8	Sensors and Encoders	37
8.1	Overview	37
8.2	Connector Overview	37
8.3	Hardware Setup	38
8.4	Units	41
8.5	Reference Frames	42
8.6	Onboard Magnetic	43
8.7	Sensor Configuration	44
8.8	Examples	44
8.9	Observer Bandwidth	47
9	Features	49
9.1	Trajectory Planner	49
9.2	Homing	50
9.3	Flux Braking	51
10	Upgrading Firmware	53
10.1	Introduction	53
10.2	Connectivity	54
10.3	Upgrade using DFU and CAN bus	57
10.4	Recovery Mode	58
10.5	Upgrade using ActiveFlashlight and UART	58
10.6	Upgrade using VSCode and J-Link	59
11	Gimbal Motors	61
11.1	Introduction	61
11.2	Enabling Gimbal Mode	61
11.3	Controlling the Motor	63
12	PID Tuning	65
12.1	Preliminaries:	65
12.2	Tuning the Velocity Controller	65
12.3	Tuning the Position Controller	66
12.4	Tips and Considerations:	66
12.5	Final Testing:	66
13	Control Principles	67
13.1	Permanent Magnet Synchronous Motors (PMSMs)	67
13.2	Field Oriented Control (FOC)	67
13.3	Control loop Overview	68
13.4	Further Reading	70
13.5	References	70
14	Troubleshooting	71
15	Developers	73
15.1	Overview	73
15.2	Hardware Connections	73
15.3	Preparation	75

15.4	Using VSCode	75
15.5	Using Eclipse	79
15.6	Setup Studio for Development	79
15.7	Custom Device Definitions	80
16	Hardware Errata	81
16.1	Tinymovr Alpha CAN Bus Connector Erratum	81
16.2	Tinymovr Alpha USB Micro Connector Erratum	82
16.3	Tinymovr R5 UART Silkscreen Reversed	82
17	API REFERENCE	83
17.1	protocol_hash	83
17.2	uid	83
17.3	fw_version	83
17.4	hw_revision	83
17.5	Vbus	84
17.6	Ibus	84
17.7	power	84
17.8	temp	84
17.9	calibrated	84
17.10	errors	85
17.11	warnings	85
17.12	save_config() -> void	85
17.13	erase_config() -> void	85
17.14	reset() -> void	86
17.15	enter_dfu() -> void	86
17.16	config_size	86
17.17	scheduler.load	86
17.18	scheduler.warnings	86
17.19	controller.state	87
17.20	controller.mode	87
17.21	controller.warnings	87
17.22	controller.errors	88
17.23	controller.position.setpoint	88
17.24	controller.position.p_gain	88
17.25	controller.velocity.setpoint	88
17.26	controller.velocity.limit	88
17.27	controller.velocity.p_gain	89
17.28	controller.velocity.i_gain	89
17.29	controller.velocity.deadband	89
17.30	controller.velocity.increment	89
17.31	controller.current.Iq_setpoint	89
17.32	controller.current.Id_setpoint	90
17.33	controller.current.Iq_limit	90
17.34	controller.current.Iq_estimate	90
17.35	controller.current.bandwidth	90
17.36	controller.current.Iq_p_gain	90
17.37	controller.current.max_Ibus_regen	91
17.38	controller.current.max_Ibrake	91
17.39	controller.voltage.Vq_setpoint	91
17.40	calibrate() -> void	91
17.41	idle() -> void	91
17.42	position_mode() -> void	92
17.43	velocity_mode() -> void	92

17.44	current_mode() -> void	92
17.45	set_pos_vel_setpoints(float pos_setpoint, float vel_setpoint) -> float	92
17.46	comms.can.rate	92
17.47	comms.can.id	93
17.48	comms.can.heartbeat	93
17.49	motor.R	93
17.50	motor.L	93
17.51	motor.pole_pairs	93
17.52	motor.type	94
17.53	motor.calibrated	94
17.54	motor.I_cal	94
17.55	motor.errors	94
17.56	sensors.user_frame.position_estimate	95
17.57	sensors.user_frame.velocity_estimate	95
17.58	sensors.user_frame.offset	95
17.59	sensors.user_frame.multiplier	95
17.60	sensors.setup.onboard.calibrated	95
17.61	sensors.setup.onboard.errors	96
17.62	sensors.setup.external_spi.type	96
17.63	sensors.setup.external_spi.rate	96
17.64	sensors.setup.external_spi.calibrated	97
17.65	sensors.setup.external_spi.errors	97
17.66	sensors.setup.hall.calibrated	97
17.67	sensors.setup.hall.errors	97
17.68	sensors.select.position_sensor.connection	97
17.69	sensors.select.position_sensor.bandwidth	98
17.70	sensors.select.position_sensor.raw_angle	98
17.71	sensors.select.position_sensor.position_estimate	98
17.72	sensors.select.position_sensor.velocity_estimate	98
17.73	sensors.select.commutation_sensor.connection	98
17.74	sensors.select.commutation_sensor.bandwidth	99
17.75	sensors.select.commutation_sensor.raw_angle	99
17.76	sensors.select.commutation_sensor.position_estimate	99
17.77	sensors.select.commutation_sensor.velocity_estimate	99
17.78	traj_planner.max_accel	99
17.79	traj_planner.max_decel	100
17.80	traj_planner.max_vel	100
17.81	traj_planner.t_accel	100
17.82	traj_planner.t_decel	100
17.83	traj_planner.t_total	100
17.84	move_to(float pos_setpoint) -> void	101
17.85	move_to_tlimit(float pos_setpoint) -> void	101
17.86	traj_planner.errors	101
17.87	homing.velocity	101
17.88	homing.max_homing_t	101
17.89	homing.retract_dist	102
17.90	homing.warnings	102
17.91	homing.stall_detect.velocity	102
17.92	homing.stall_detect.delta_pos	102
17.93	homing.stall_detect.t	102
17.94	home() -> void	103
17.95	watchdog.enabled	103
17.96	watchdog.triggered	103
17.97	watchdog.timeout	103

Note: This is the documentation for the latest stable version of the Tinymovr Firmware and Studio app. For the legacy 0.x.x documentation, check out the [legacy docs](<https://tinymovr.readthedocs.io/en/attic-legacy/>).

FROM UNBOX TO FIRST SPIN

1.1 Preparation

Welcome to Tinymovr!

If you are using a Tinymovr Servo Kit/Tinymovr Dev Kit, please ensure you have completed [Connecting Data](#) and [Connecting Power](#).

If you are using a Tinymovr board in your own setup, please go through [Hardware Overview](#) and [Hardware Setup](#).

If using Tinymovr on Windows with a CANtact-compatible CAN Bus adapter, such as CANine, you will need to install an .inf file to enable proper device naming. You can [download the inf file here](#). Extract the archive, right click and select Install.

If using CAN bus, you may need to enable at least one termination resistor, either on the CANine adapter, or on one Tinymovr. To enable the termination resistor on CANine, you can take a look at the [CANine connectors diagram](#).

Before proceeding to the next steps, ensure your Tinymovr is powered up.

1.2 Installing and Launching Studio

Tinymovr can be installed using pip. Python 3.10 or greater is required.

Note: We recommend installing Tinymovr in a virtual environment. [Here is a quick tutorial on how to setup a virtual environment using Conda](#).

```
pip3 install 'tinymovr[GUI]'
tinymovr
```

You should now be looking at the Tinymovr Studio GUI interface.

1.3 Installing on Raspberry Pi

Installation on Raspberry Pi requires a few additional steps.

```
sudo apt update
sudo apt install python3-pip python3-numpy libopenjp2-7 libtiff5
pip3 install tinymovr
```

You may also need to append a directory to your PATH variable:

```
echo 'export PATH="/home/pi/.local/bin:$PATH"' >> ~/.bashrc
source ~/.bashrc
```

Now you should be able to run the Tinymovr CLI:

```
tinymovr_cli
```

1.4 Calibrating using the Studio GUI

The Tinymovr Studio GUI offers an overview of the device internals as a tree structure. You can inspect the various values to ensure they are as expected.

The Tinymovr Servo Kit motor and encoder are already calibrated. If you have your own setup, or if you experience problems with prior calibration, you'll need to go through the following brief calibration procedure.

In Tinymovr Studio, navigate to *tmx->controller*. Press the button with the arrow next to the *calibrate* label. Note that after pressing this button, the motor will spin. Ensure the rotor is free of obstructions or loads, and the motor is firmly fixed.

Follow the on-screen prompts. The motor will produce an audible beep and rotate in one direction. Your Tinymovr is now ready for operation. Navigate to *tmx->motor*. This will reveal identified motor parameters, namely: phase resistance, phase inductance, number of pole pairs and encoder ticks.

1.5 Testing Position Control using the Studio GUI

Navigate back to *tmx->controller*. Press the button with the arrow next to the *position_mode* label. Note that after pressing the button, the motor will hold position and may spin. The motor should now be actively holding its position. Try moving it by hand and you should feel resistance.

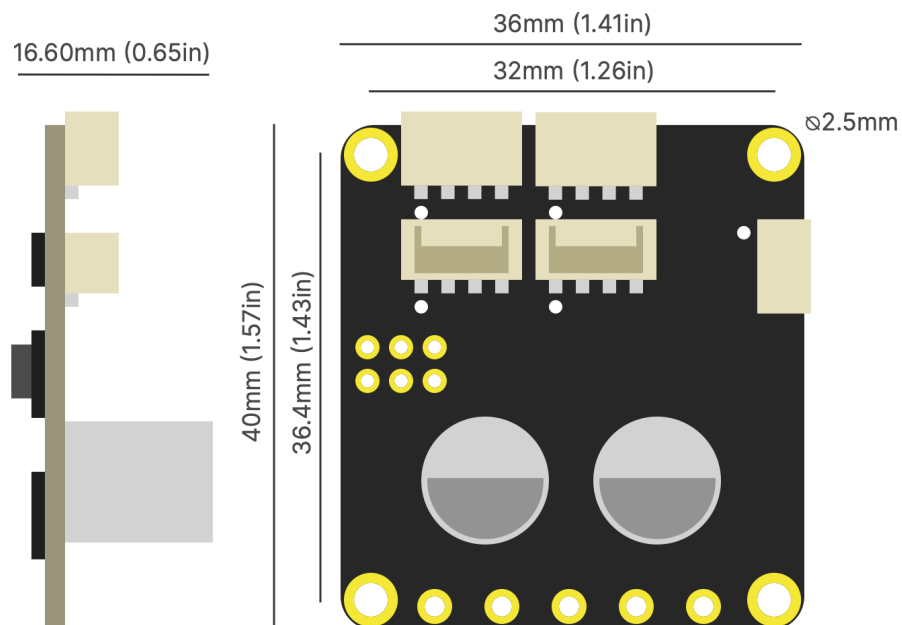
To command a new position, navigate to *tmx->controller->position*, and double-click on the value next to the *setpoint* label. This value is expressed in ticks, a unit that denotes 1/8192 of the circle. Type in a new position followed by Enter. The motor should jump to the commanded position.

HARDWARE OVERVIEW

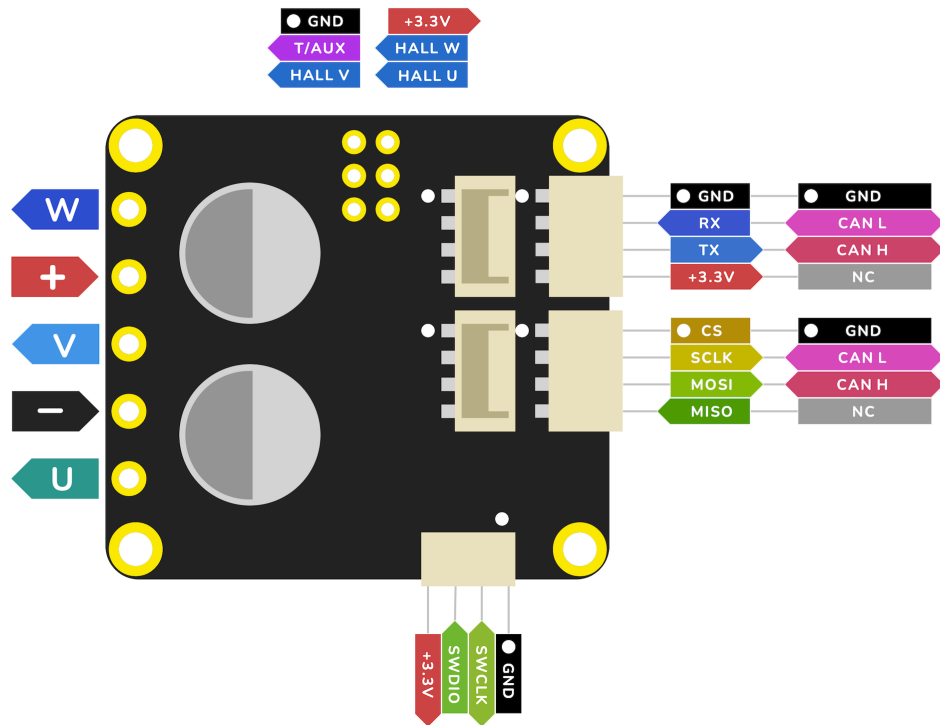
2.1 R5.x

Tinymovr R5.x is the latest Tinymovr revision. It features increased connectivity in a reduced footprint.

2.1.1 Board Dimensions (R5.0, R5.1)

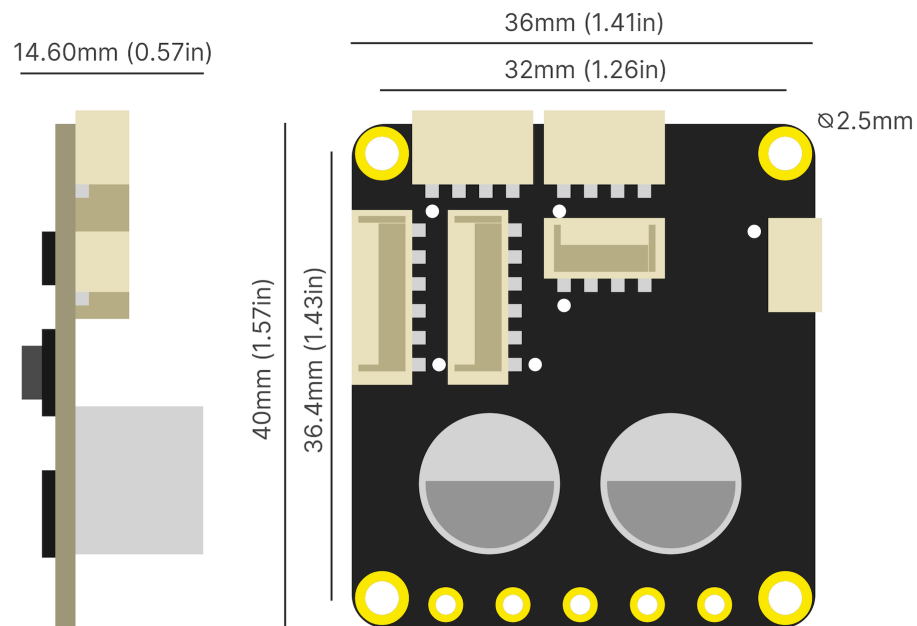


2.1.2 Connectivity (R5.0, R5.1)

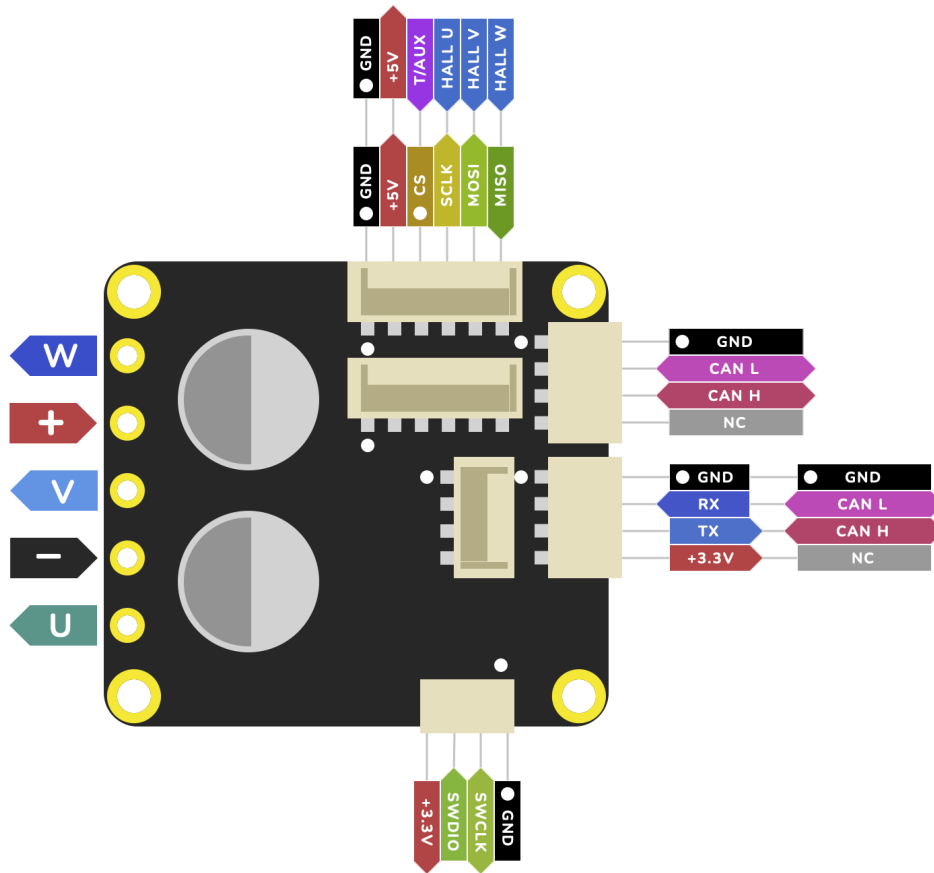


Warning: The UART pins in Tinymovr R5.1 have the silkscreen reversed. If you are planning to use UART with R5.1, consult *Tinymovr R5 UART Silkscreen Reversed*. This only affects R5.1 boards.

2.1.3 Board Dimensions (R5.2)



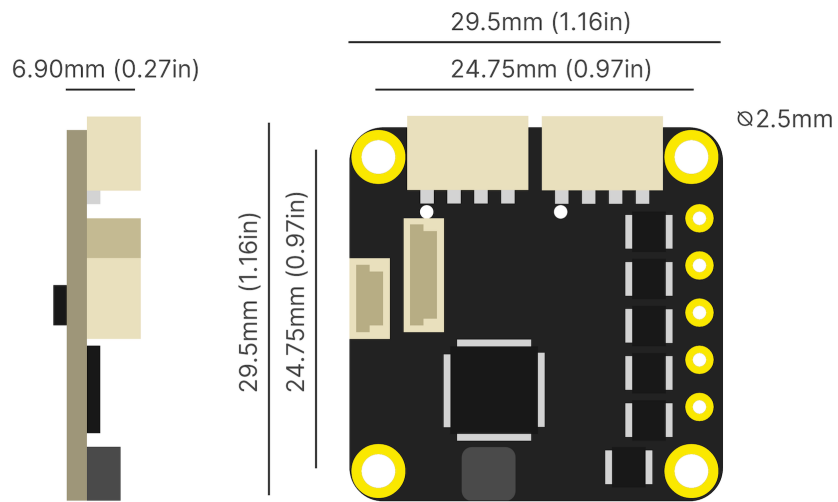
2.1.4 Connectivity (R5.2)



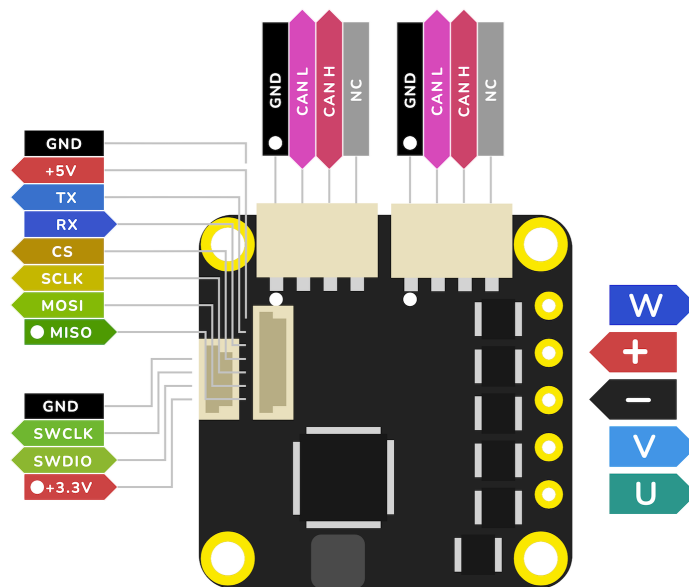
2.2 M5.x

Tinymovr M5.x is our specialized driver for gimbal motors and light robotic joints. It features a very compact footprint and 5A max drive.

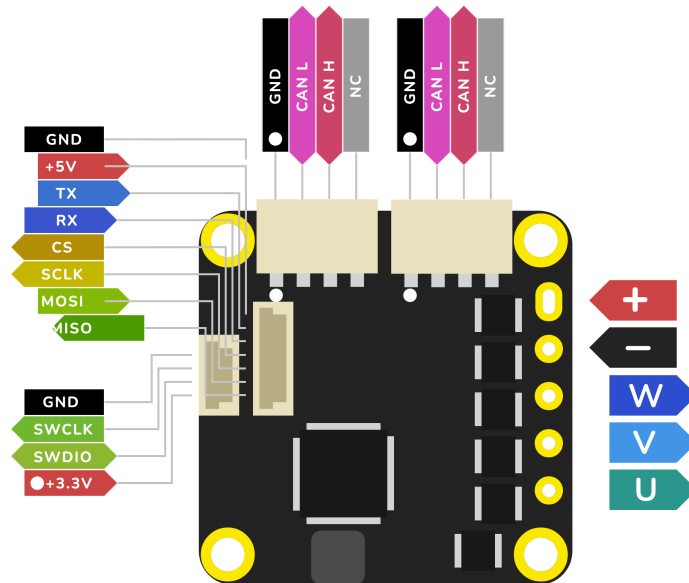
2.2.1 Board Dimensions (M5.1, M5.2)



2.2.2 Connectivity (M5.1)



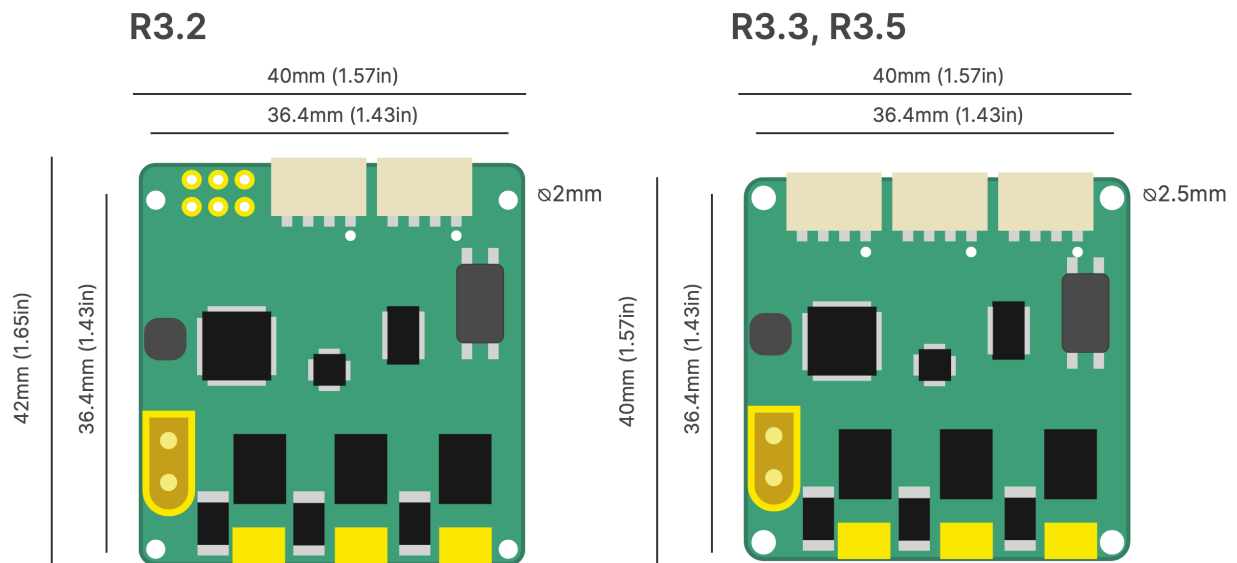
2.2.3 Connectivity (M5.2)



2.3 R3.x

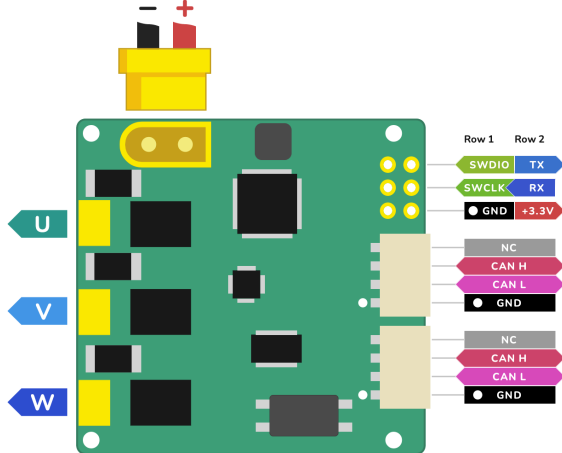
R3.x is the previous Tinymovr revision, with CAN and UART connectivity.

2.3.1 Board Dimensions (R3.x)

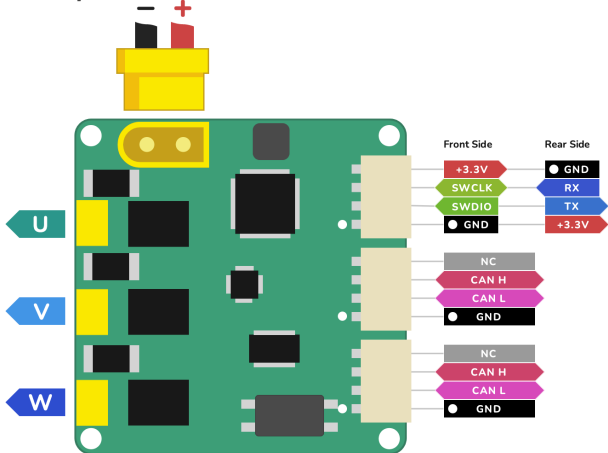


2.3.2 Connectivity (R3.x)

R3.2



R3.3, R3.5



HARDWARE SETUP

3.1 Requirements

1. A 3-phase brushless motor (see below for supported types)
2. A diametrically magnetized sensor magnet (included with Tinymovr), mounted firmly on the motor shaft.
3. A means to talk CAN Bus, such as CANine or a Canable-compatible adapter.
4. A mechanical rig that ensures firm connection between the Tinymovr PCB and the brushless motor. Designs that can be 3D printed are available.

The Tinymovr Servo Kit includes all of the above, except for the CANine adapter, in a ready to use kit.

3.2 Supported Motor Types

3.2.1 Tinymovr R5.x and R3.x

Most three-phase pancake-style outrunners can be used with Tinymovr R5. While there is a lot of variation between motors of even the same size and external appearance, as a general rule-of-thumb motors ranging from 40mm outer diameter to 110mm should work fine with Tinymovr.

For a good quality and affordable option, take a look at our own [Tinymovr Motors](#).

3.2.2 Tinymovr M5.x

Most gimbal motors can be used with Tinymovr M5. Gimbal motors are essentially outrunners with high resistance (they have more turns). You can read more about gimbal motors in [Introduction](#). Note that, to control gimbal motors with Tinymovr M5, you do not have to (in fact, should not) use Gimbal mode. Tinymovr M5 is capable of accurately sensing current at a lower scale, which is compatible with gimbal motors.

Note:

- Tinymovr expects a motor with sinusoidal back-EMF. Most brushless outrunners (including gimbal motors) have sinusoidal back-EMF. If uncertain and you have access to an oscilloscope, you can hook up the motor phases to the scope channels and check out the back-EMF for yourself.
- Motors with trapezoidal back-EMF can also be controlled, but control will be sub-optimal. The trapezoidal back-EMF will appear as residual in the dq frame, as a result it will be much harder for the current controller to regulate phase currents. The tangible result is that the motor may exhibit increased noise and vibration while running.

3.3 Mechanical Setup

3.3.1 Mounting Motor and Tinymovr

The most important aspect of a correct setup is to ensure the controller is properly positioned in relation to the sensor magnet. We assume that the center of the PCB, where the angle sensor IC is located, is located coaxially to the motor shaft, and that the sensor magnet is at the end of the shaft (check out the diagrams below). In addition, the distance from the sensor magnet to the sensor IC should be less than 2mm (less than 1mm if you are mounting the PCB backwards, i.e. the sensor IC is facing away from the magnet).

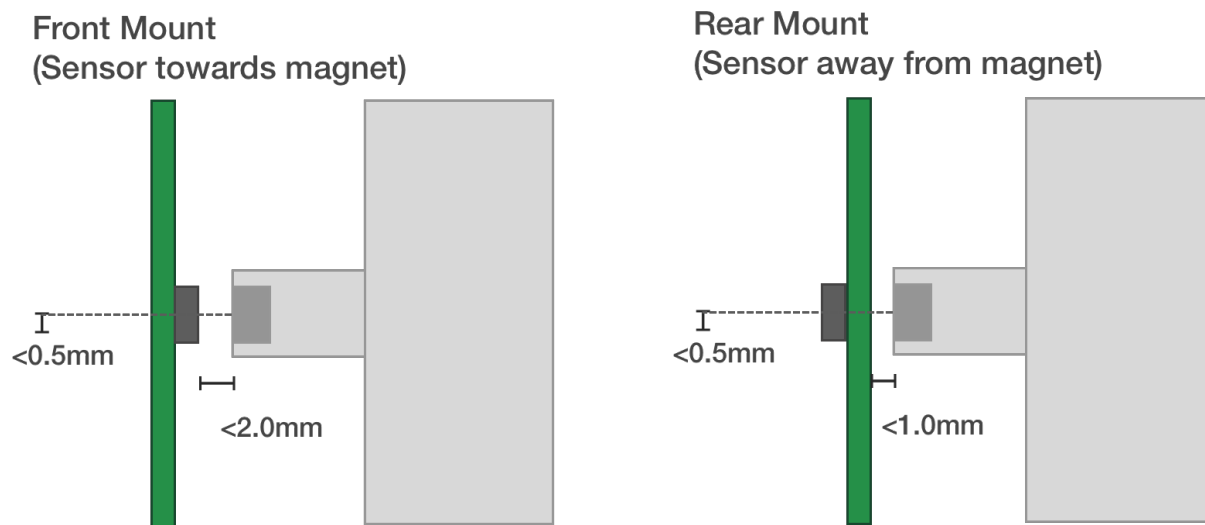


Fig. 1: Tinymovr and motor mechanical mounting

A [3D printable sensor magnet jig](#) is available, suitable for 6mm disc magnets and 14, 19, 25 and 30mm motor hole diameters.

Note: For safety reasons, you should always ensure the motor & controller assembly are secured to a stable surface before operation. The motor rotor may experience high acceleration that may cause damage or injury if not secured properly.

Sensor Magnet and Mounting Tips

- Unless using an external sensor (e.g. Hall effect sensor), Tinymovr requires proximity to the sensor magnet to operate. If the magnet is not in close proximity, an error will be raised upon power up, preventing any further action (calibration, closed loop control).
- Ensure the sensor magnet is firmly attached to the motor shaft, otherwise it may slip out of sync. Use strong adhesive to secure.

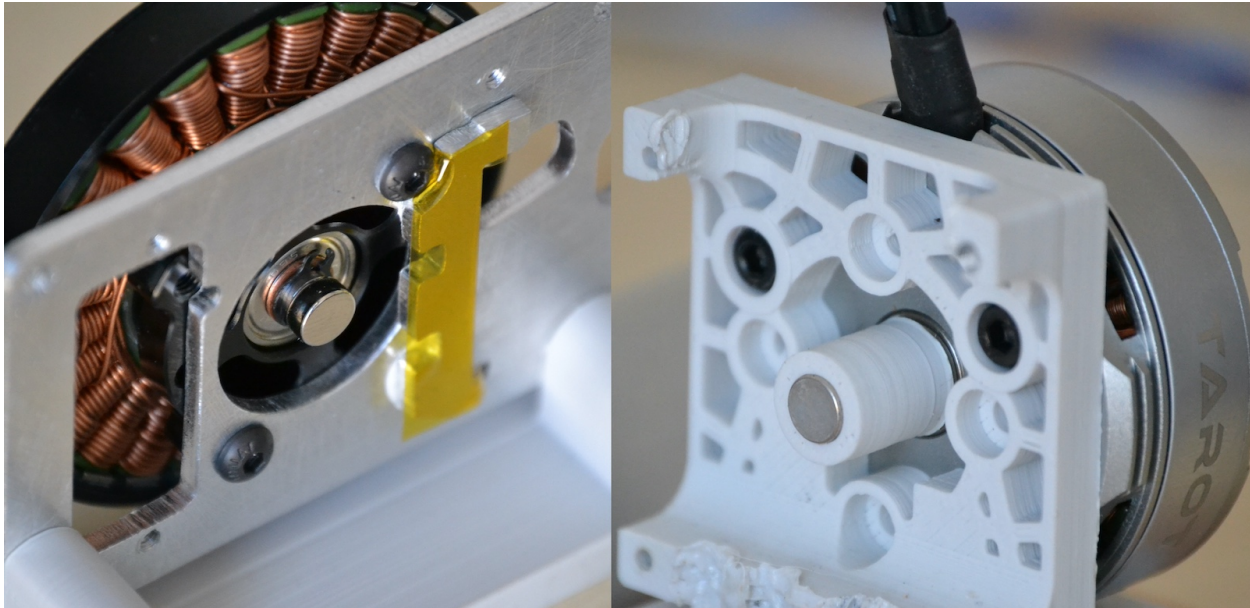


Fig. 2: Left: Magnet mount directly on shaft. Right: Magnet mount using 3d-printed holder.

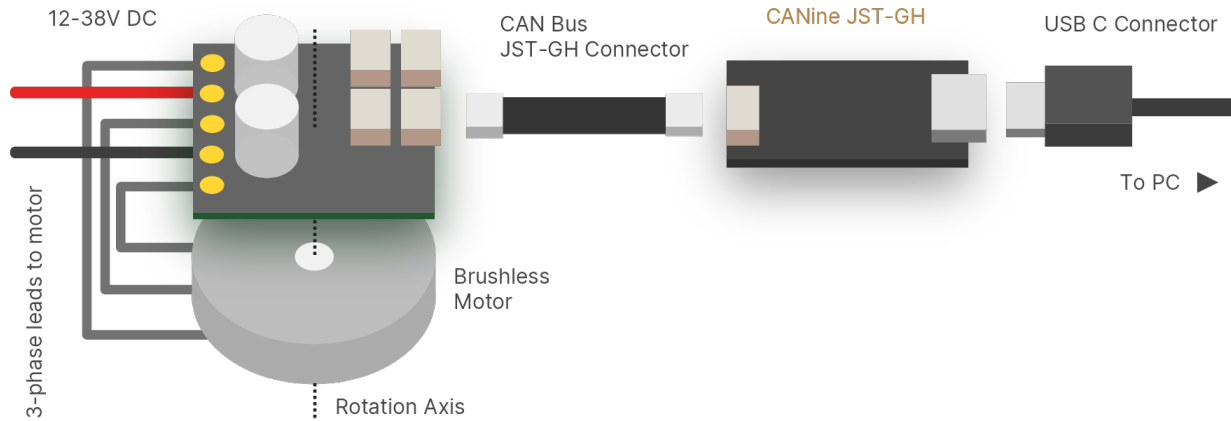
- It is possible to have the magnet on the rear side of the PCB, i.e. opposite of the magnet sensor IC, but the gap needs to be reduced to account for the PCB thickness. This has been verified by MPS in a forum post¹, quoted below: *[...] this type of arrangement is possible, what really matters in the end is that there is enough magnetic field reaching the sensor. Of course the minimum distance is imposed by the thickness of the PCB, so it puts some constraints on the design, that you have to take into account when choosing the magnet (you can use our online simulation tool for that). But as long as the PCB is not acting as a magnetic shield (due to copper plane), then it is fine.*
- Calibration needs to be performed without any loads on the motor. If the motor is coupled to a load, the sensor offset angle may not be determined correctly, leading to a sub-optimal setup.
- For Tinymovr R3.3, adjust your termination resistor DIP switch (if needed) before putting together your actuator, to avoid needing to disassemble it for adjustment later on. See also [Connecting Data](#).

3.4 Electrical Setup

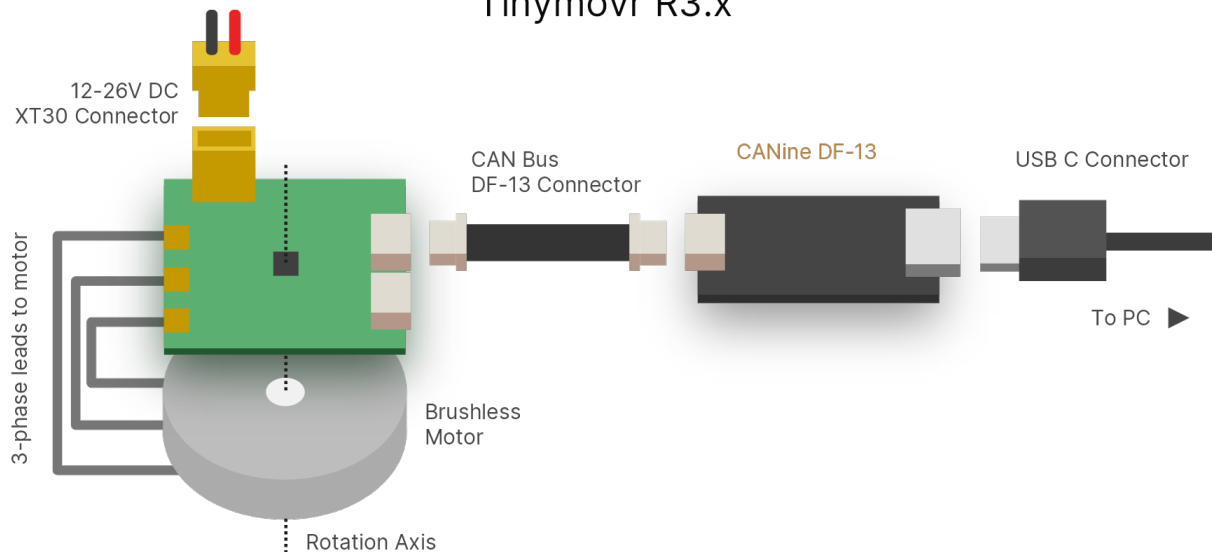
Electrical setup comprises three main parts: Motor connection, data connection and power connection. Below is a diagram with the electrical connection scheme.

¹ MPS forum post on mounting MA702 and magnet on opposite sides of PCB: <https://forum.monolithicpower.com/t/mounting-ma702-and-magnet-on-opposite-sides-of-pcb/1609>

Tinymovr R5.x



Tinymovr R3.x



3.5 Connecting Motor

Connect the three motor phases to the three terminals on Tinymovr. The order of connection is not important, and motor direction will be determined during motor/sensor calibration.

- Tinymovr R3.x: The motor leads can be connected by soldering on the PCB.
- Tinymovr R5.x: The leads can be connected by soldering on the PCB, or you can solder a screw terminal block and secure the motor leads on the block.
- Tinymovr M5.x: The leads can be connected by soldering on the PCB.

3.6 Connecting Data

Connect the CAN bus header to one of the two CAN sockets on the board. It is not important which one you choose. If this is a terminal node in the CAN network, you may need to use a termination resistor, as follows:

- Tinymovr R3.x: flip ONLY the DIP switch labelled “CAN 120R” to on to enable the 120 termination resistor.
- Tinymovr R5.x: you will need to provide an external 120 termination resistor.
- Tinymovr M5.x: you will need to provide an external 120 termination resistor.

In small setups with few nodes and short wires, it is better to enable just a single termination resistor, either on one Tinymovr board or on the CAN adapter. In setups with many nodes and long cables, you may need to enable termination resistors in both terminal nodes. A typical CAN driver has an “open-drain” output structure, meaning that the dominant edge is actively driven and the recessive edge is not. Therefore, properly terminating the bus is very important because it ensures that the recessive edge decays properly, and in time for the next bit’s sample point².

Warning: The UART pins in Tinymovr R5.1 have the silkscreen reversed. If you are planning to use UART with R5.1, consult *Tinymovr R5 UART Silkscreen Reversed*.

3.7 Connecting Power

- Tinymovr R3.x can be powered from a 12-26V (3S-6S) power source.
- Tinymovr R5.x can be powered from a 12-38V (3S-9S) power source.
- Tinymovr M5.x can be powered from a 12-38V (3S-9S) power source.

With the power source off/disconnected, connect the power leads observing correct polarity. Turn on/connect the power source. Upon successful power-up, the onboard LED should light up.

Note: Each Tinymovr board has a capacitance of around 500F (R3.x) / 240F (R5) / 50F (M5). Such capacitance can introduce significant inrush current upon power-on, especially if several boards are connected to the same power supply. To prevent damage to components from overcurrent, the use of an inrush current limiter or a current-limited power supply is advised. We offer a [Power Distribution and protection device](#) which is suitable for this task.

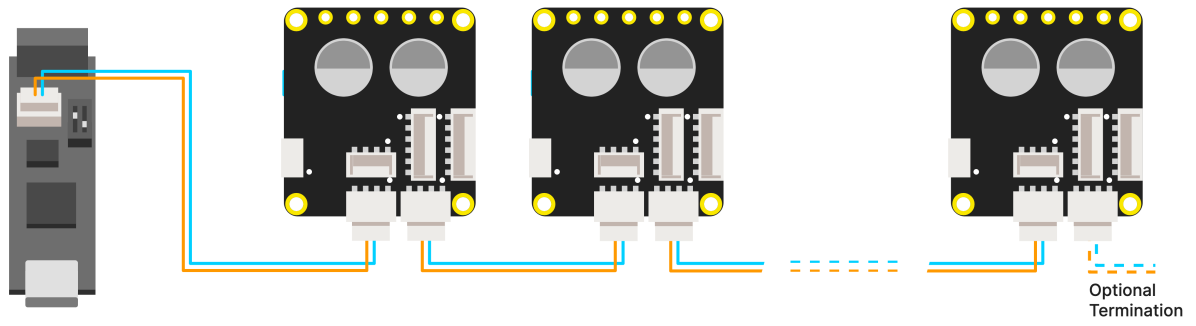
3.8 Connecting Multiple Nodes (Daisy-Chaining)

Multiple nodes can be connected in a single CAN Bus network by means of daisy-chaining. Tinymovr R5 and Tinymovr M5 offer two CAN Bus ports for this purpose, which makes it easy to daisy-chain units. Note that for networks with long cable lengths, you may need termination on both ends. On one end this can be achieved with the onboard termination resistor of CANine, but on the other end you will have to provide your own.

² Split Termination, Texas Instruments. Available at <https://www.ti.com/document-viewer/lit/html/SSZTB40#:~:text=for%20each%20termination,Split%20Termination,typically%20between%201%2D100nF>.

CANine CAN Bus Adapter

Tinymovr Nodes



STUDIO INSTALLATION

Tinymovr Studio is a cross-platform GUI application, CLI application, and Python library that offers easy access to all of Tinymovr's functionality.

Studio requires Python 3.9 or newer.

4.1 Preparation

If using Windows with a CANTact-compatible CAN Bus adapter, such as CANine, you will need to install an .inf file to enable proper device naming. You can [download the inf file here](#). Extract the archive, right click and select Install.

4.2 Using pip

This is the most straightforward method to install Tinymovr studio and have access to hardware. The following command will install Tinymovr with the dependencies required for the Qt-based Tinymovr Studio GUI:

```
pip3 install 'tinymovr[GUI]'
```

If you don't plan to use the GUI, you can skip installing some dependencies using the following installation command instead:

```
pip3 install tinymovr
```

```
tinymovr
```

You should now be looking at the Tinymovr GUI. Alternatively, to start the IPython-based Command Line Interface (CLI):

```
tinymovr_cli
```

4.2.1 Anaconda Installation

Tinymovr can be installed inside a Virtualenv or Anaconda environment.

With Anaconda, create a new environment:

```
conda create --name tinymovr python=3.10 -y
conda activate tinymovr
```

Alternatively you can use any Python version ≥ 3.9 .

Then simply install and run Tinymovr:

```
pip install 'tinymovr[GUI]'
tinymovr
```

4.3 Using git clone

Note: The master branch of the Github repository represents the state of art of development, and it may contain bugs. For a stable version, especially if you are starting with the project, please consider installing Tinymovr Studio using pip as shown above.

First clone the Tinymovr repo to a local directory:

```
git clone https://github.com/yconst/Tinymovr
```

Then cd to the cloned repo directory and install in developer mode:

```
cd Tinymovr/studio/Python
pip3 install -e .
```

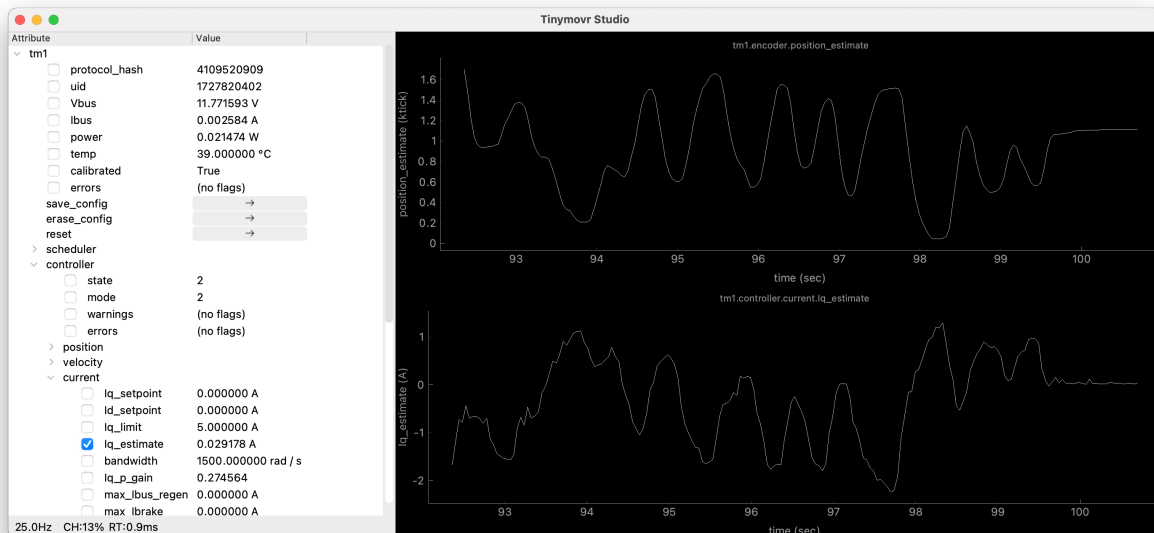
STUDIO USAGE

Tinymovr Studio is a cross-platform GUI application, CLI application, and Python library that offers easy access to all of Tinymovr’s functionality.

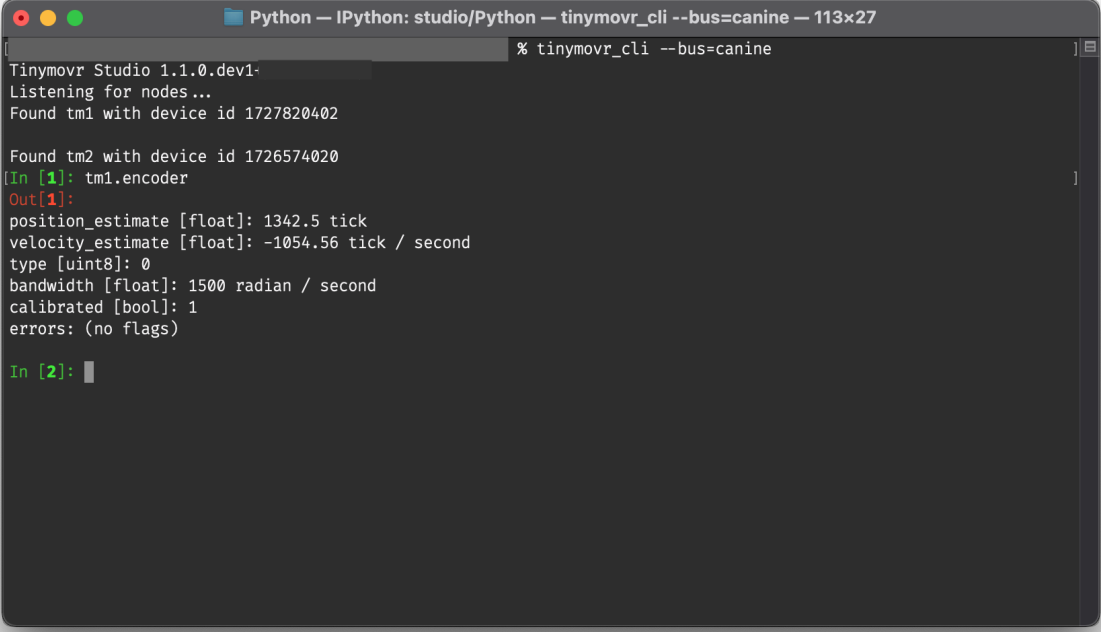
This documentation page includes examples both for the as well as the apps.

5.1 Launching Studio

`tinymovr`



`tinymovr_cli`



```
Python — IPython: studio/Python — tinymovr_cli --bus=canine — 113x27
% tinymovr_cli --bus=canine

Tinymovr Studio 1.1.0.dev1-
Listening for nodes...
Found tm1 with device id 1727820402

Found tm2 with device id 1726574020
[In [1]: tm1.encoder
Out[1]:
position_estimate [float]: 1342.5 tick
velocity_estimate [float]: -1054.56 tick / second
type [uint8]: 0
bandwidth [float]: 1500 radian / second
calibrated [bool]: 1
errors: (no flags)

In [2]:
```

5.2 Alternative Adapters/Firmwares

By default Tinymovr Studio searches for either slcan-compatible CAN bus adapters or adapters using the [CANine firmware](#)¹. To specify an alternative device, use the `--bus` command line argument.

For instance, to work with SocketCAN-compatible adapters in linux, launch Tinymovr Studio as follows:

```
tinymovr --bus=socketcan
```

```
tinymovr_cli --bus=socketcan
```

5.3 Compatibility

Tinymovr Studio includes by default a checksum comparison to determine protocol compatibility between firmware and studio version. This is performed each time a node is discovered, and prior to initializing the tinymovr object. If you see a compatibility-related message, please upgrade to the latest studio and firmware versions.

¹ Note that you need to have completed [setting up CANine](#) before working with CANine.

5.4 Custom Device Specs

You can specify a custom device spec (YAML file) as a command line argument:

```
tinymovr --spec=/path/to/myspec.yaml
```

```
tinymovr_cli --spec=/path/to/myspec.yaml
```

This is useful, for instance, if you have altered the default Tinymovr spec files. Using this parameter with a project developed using [Avlos](#), you can even use Tinymovr Studio to control your own custom devices!

5.5 Issuing Commands in CLI

You can read/write variables and issue commands using the respective Tinymovr handle, e.g.:

```
tm1.encoder
```

or

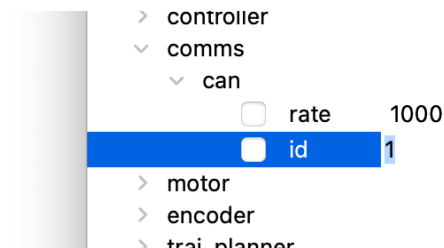
```
tm1.controller.pos_setpoint = 10000
```

Replace “tm1” with the correct device ID if necessary. Full tab completion is available.

5.6 Multiple Instances

In order for multiple Tinymovr instances to coexist in the same CAN network, they need to have unique IDs. The default ID is 1. To assign different IDs to each board, follow the method below:

1. Connect a single Tinymovr to the bus and launch Studio. The board will be assigned the default ID, 1, and will be accessible as tm1.
2. Change the ID



The board will be discovered with the new ID. Studio GUI will rescan, discover the new node, and remove the old instance.

2. Change the ID

```
tm1.comms.can.id = x
```

where x is the desired ID. You can assign IDs in the range 1-1024.

The board will be discovered with the new ID. Relaunch Studio CLI to remove the old board instance.

3. Save configuration.

<input type="checkbox"/>	errors	(no flags)	
<input type="checkbox"/>	save_config	→	
<input type="checkbox"/>	erase config	→	

3. Save configuration.

```
tm1.save_config()
```

4. Power down or reset the board. Tinymovr is now ready to use with the new ID.

5.7 Command-line options

Tinymovr Studio supports the following command line options.

5.7.1 --bus=<bus>

The `--bus` option specifies a CAN bus type to use.

Example:

```
tinymovr --bus=canine
```

```
tinymovr_cli --bus=canine
```

All interfaces offered by python-can are supported.

5.7.2 --chan=<chan>

The `--chan` options specifies a channel to use, optionally together with the `--bus` option.

Example:

```
tinymovr --bus=socketcan --chan=CAN0
```

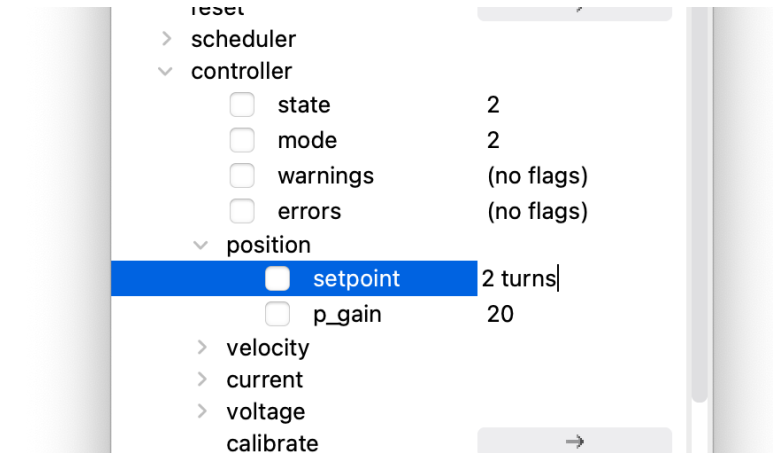
```
tinymovr_cli --bus=socketcan --chan=CAN0
```

By default, Tinymovr Studio will use `slcan` as the interface, and will search for `CANable/CANtact`-type devices with `slcan` firmware. Such is the `CANine` adapter supplied with Tinymovr Servo Kits.

5.8 Units

Tinymovr Studio introduced physical units and quantities since v0.3.0. Units are introduced through the [Pint](#) package. Using units you will see all values that you query associated with a unit, which forms a physical quantity.

In the GUI, units are displayed by default in any quantity that supports them. For instance:



You can set any quantity supporting units by specifying the desired compatible unit next to the quantity you want to set. For instance, to set the position setpoint:

In the CLI, units are displayed whenever a quantity that supports them is printed:

```
In [1]: tm1.encoder.pos_estimate
Out[1]: 0.0 <Unit('tick')>
```

You can also set quantities in any (defined) unit you wish. For instance:

```
In [1]: tm1.controller.pos_setpoint = 2.0 * rad
```

The above will set the rotor position to 2 radians from the initial position. Similarly for velocity:

```
In [1]: tm1.controller.vel_setpoint = 3.0 * rad / second
```

Will set velocity to 3 radians/second. If not unit is used in setting a value, the default units will be assumed, in the above cases ticks and ticks/second.

The ureg object is the unit registry, and it is that which holds all unit definitions. You can use it to do all sorts of cool stuff such as doing conversions, defining your own shortcuts or even new units.

For instance, to define a few frequently used shortcuts in a program:

```
from avlos import get_registry
ureg = get_registry()
mA = ureg.milliamperere
rad = ureg.radian
s = ureg.second
```

Then you can use the defined shortcuts to intuitively set values, such as a position setpoint with velocity and current feed-forwards:

```
tm1.controller.pos_setpoint = 2*PI * rad
tm1.controller.vel_setpoint = PI * rad/second
tm1.controller.cur_setpoint = 1.5 * ampere
```

Take a look at the [API REFERENCE](#) for default units used in each command.

For more information on units and their usage, take a look at [Pint's documentation](#)

5.9 Socketcan & Linux

You can use a socketcan-enabled CAN adapter with Tinymovr Studio. The CANine adapter supplied with Tinymovr Servo Kits supports Socketcan natively with the alternative Candlelight Firmware. To connect to a Socketcan device, run Studio as follows:

```
tinymovr --bus=socketcan --chan=CAN0
```

```
tinymovr_cli --bus=socketcan --chan=CAN0
```

INTEGRATING TINYMOVR

Tinymovr can be integrated into your projects in the form of a Python library. Using the library, you can generate and manipulate one or more Python objects that represent Tinymovr controllers, with the full device tree, as exposed in the Studio GUI and CLI. The communication protocol defined by Avlos and used by Tinymovr comprises a series of read/write endpoints. The endpoints are defined taking into account the capabilities and constraints of the CAN bus, the main communication bus used by Tinymovr, however they are not strictly constrained to CAN, and can be used with other comm channels as well.

The Tinymovr Library allows full hardware control from within Python scripts, using a high-level interface to hardware. For use with embedded devices, the Tinymovr C++ Library is proposed as an alternative.

It is of course possible to interface directly with Tinymovr using the protocol endpoints by generating the message frames yourself. However, this is generally advised against, as the libraries offer robustness and functionality that is not trivial to implement from scratch.

The Tinymovr Python library is part of Tinymovr Studio. For help installing Studio, please take a look at [Studio Installation](#).

For the full Tinymovr protocol reference, take a look at the [API REFERENCE](#)

6.1 Use with Python

Here below is an example using the API from Python scripts and controlling hardware:

```
import can
from tinymovr.bus_router import init_router
from tinymovr.config import get_bus_config, create_device

params = get_bus_config(["canine", "slcan_disco"], bitrate=1000000)
init_router(can.Bus, params)
tm = create_device(node_id=1)

tm.controller.calibrate()
```

The above code block will instantiate a Tinymovr with CAN bus id of 1 and calibrate it. *slcan_disco* is our custom slcan plugin for python-can that allows discovery of slcan-flashed CANine devices. Following the above, you can issue commands such as:

```
tm.controller.position_mode()
tm.controller.position.setpoint = 10000
```

(continues on next page)

```
tm.controller.velocity_mode()  
tm.controller.velocity.setpoint = 80000
```

6.2 BusRouter API

The *BusRouter* class in the Tinymovr library is designed to facilitate message distribution and communication on the CAN bus by distributing incoming messages based on the result of a filter callback.

6.2.1 API Overview

class BusRouter

Initializes and manages the routing of CAN bus messages between various clients. It integrates a simple forwarding mechanism for sending messages to simplify interfacing with CAN bus objects.

__init__(*bus_manager, timeout, logger*)

Constructor for the BusRouter. Initializes the bus manager, timeout, and logger.

Note: This constructor should not be used directly. Instead, use the *init_router()* function.

Parameters

- **bus_manager** – Instance of BusManager to handle low-level bus interactions.
- **timeout** – Timeout in seconds for receiving messages.
- **logger** – Logger instance for logging information.

run()

Main thread function that continuously listens for incoming frames and dispatches them to the appropriate clients based on filter results.

add_client(*filter_cb, recv_cb*)

Registers a new client to receive messages that meet specific conditions.

Parameters

- **filter_cb** – Callback function that determines if the incoming frame should be passed to the client.
- **recv_cb** – Callback function that handles the received frame.

stop()

Stops the router's main running thread, effectively stopping the routing of messages.

shutdown()

Performs a complete shutdown by stopping the router and closing all bus manager activities.

send(*frame*)

Sends a frame via the bus manager.

Parameters

frame – The CAN frame to be sent.

6.2.2 Bus Router Management Functions

These helper functions manage the lifecycle of the *BusRouter* instance.

init_router(*bus_class*, *bus_params*, *logger*, *timeout=0.1*)

Initializes a bus router using a python-can bus instance. This function creates a singleton instance of *BusRouter*.

Parameters

- **bus_class** – The bus class from python-can to be used.
- **bus_params** – Parameters for the bus initialization.
- **logger** – Logger instance for logging activities.
- **timeout** – Timeout in seconds for receiving messages.

Returns

An instance of *BusRouter*.

destroy_router()

Destroys the existing bus router and stops its associated thread.

get_router()

Returns the current instance of the bus router if it exists.

6.2.3 Deprecations

The following functions are deprecated and will be removed in future versions:

init_tee()

Deprecated since version 2.1: Use *init_router()* instead.

destroy_tee()

Deprecated since version 2.1: Use *destroy_router()* instead.

get_tee()

Deprecated since version 2.1: Use *get_router()* instead.

6.3 Error Codes

Tinymovr uses error codes to indicate faults in operation. Each component presents their own error states, which are summarized in the [API REFERENCE](#).

COMM INTERFACES

7.1 CAN Bus

CAN Bus is the primary interface for communicating with Tinymovr. The physical and data link interface adheres to the CAN 2.0 standard. Tinymovr exposes to CAN all communication endpoints defined in firmware.

7.1.1 Data rate

By default, the CAN baud rate is set to 1Mbit/s. This is customizable both through CAN as well as through UART. See *comms.can.rate*. Possible values are 125kbit/s, 250kbit/s, 500kbit/s and 1Mbit/s.

7.1.2 Addressing

Tinymovr uses extended CAN frames for communication. The 29-bit identifier of the extended CAN frame is used for device, endpoint and message sequence identification. The 12 least significant bits of the identifier are reserved for endpoint identification, the next 9 for message sequence identification, and the 8 most significant bits, for device identification. The total number of addressable endpoints in a single device are 4096, the number of addressable devices is 256, and a message sequence can include up to 512 messages.

7.1.3 Protocol

For a detailed description, please see *Integrating Tinymovr* and *API REFERENCE*.

7.2 UART

As an alternative to CAN Bus, Tinymovr offers UART-based (serial) communication. The protocol is much simpler than CAN and mainly designed for troubleshooting or testing in the absence of CAN hardware.

Warning: The UART port on Tinymovr is NOT 5V tolerant. Applying 5V voltage will immediately damage the onboard PAC5527 controller. Please use only 3.3V for UART communication.

Warning: The UART port on Tinymovr offers a 3.3v output for driving very light loads (30mA absolute max). Tinymovr cannot be powered by this pin. In addition, most UART adapters offer 5V power, which is incompatible with Tinymovr. **In short: If in doubt, leave this pin disconnected.**

7.2.1 Protocol Description

The UART port is TTL at 115200 baud. A regular FTDI-style USB to UART adapter should be sufficient.

UART communication is based on a simple human-readable protocol dubbed the “dot protocol”, because the dot is the command starting character. The protocol is response-only, meaning that Tinymovr will only respond to commands initiated by the client, it will never initiate a transmission on its own.

The command template is as follows:

```
.Cxxxx
```

The command begins with a dot. The next single character identifies the command. The characters following the second one are optional values to pass to write commands. Read commands only include the dot and command character. The command is finalized with a newline character (n, not shown above).

For instance, to get the current position estimate:

```
command: .p
response: 1000
```

To set the velocity estimate in encoder ticks:

```
command: .V10000
(no response)
```

The values passed or returned are always integers scaled by the mentioned factor (see command reference below).

Note that command characters are case-sensitive, i.e. capitals and small represent different commands. As a convention, capital letters are setters and small are getters, where applicable.

7.2.2 Command Reference

.Z

Transition to idle state.

Example

```
.Z
0
```

.Q

Transition to calibration state.

Example

```
.Q
0
```


.A

Transition to closed loop control state.

Example

```
.A  
0
```

.e

Get the error code.

Example

```
.e  
0
```

.p

Get position estimate (ticks).

Example

```
.p  
1000
```

.v

Get velocity estimate (ticks/s).

Example

```
.v  
-200
```

.i

Get current (Iq) estimate (mA).

Example

```
.i  
2000
```

.P

Get/set position setpoint (ticks).

Example

```
.P  
1000
```

```
.P1000
```

.V

Get/set velocity setpoint (ticks/s).

Example

```
.V  
-10000
```

```
.V-10000
```

.I

Get/set current (Iq) setpoint (mA).

Example

```
.I  
1000
```

```
.I1000
```

.W

Get/set current (Iq) limit (mA).

Example

```
.W  
10000
```

```
.W15000
```

.Y

Get/set position gain.

Example

```
.Y  
25
```

```
.Y25
```

.F

Get/set velocity gain (x0.000001).

Example

```
.F  
20
```

```
.F20
```

.G

Get/set velocity integrator gain (x0.001).

Note that high values (e.g. above 10) may cause instability.

Example

```
.G  
2
```

```
.G2
```

.H

Get/set motor phase resistance (mOhm).

Example

```
.H  
200
```

```
.H 200
```

.L

Get/set motor phase inductance (H).

Example

```
.L  
2000
```

```
.L 2000
```

.R

Reset the MCU.

Example

```
.R
```

.S

Save board configuration.

Example

```
.S
```

.X

Erase board configuration and reset.

Example

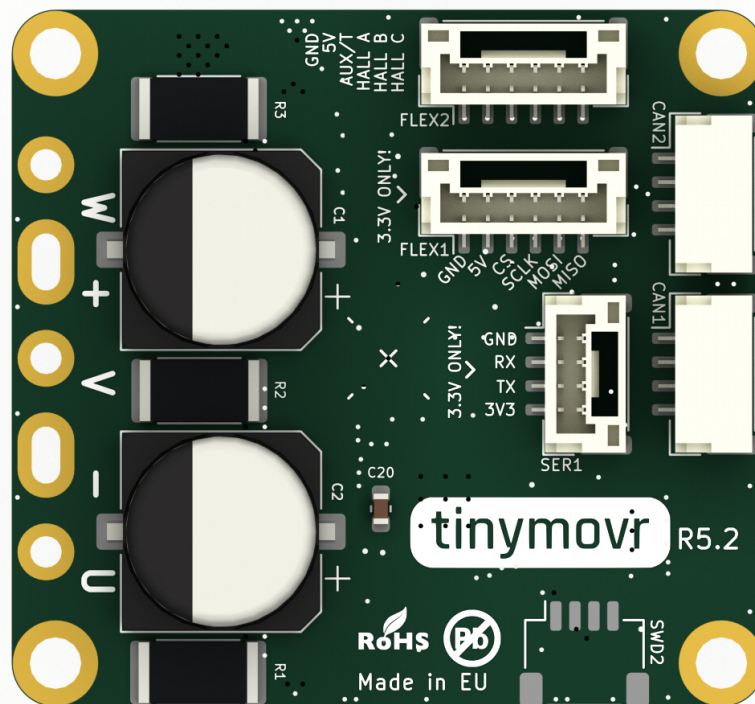
```
.X
```

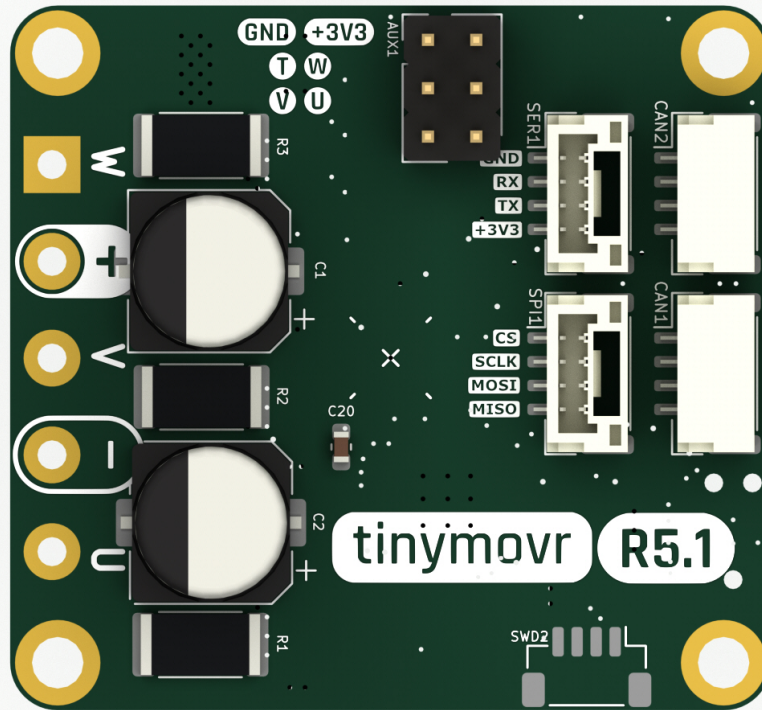
SENSORS AND ENCODERS

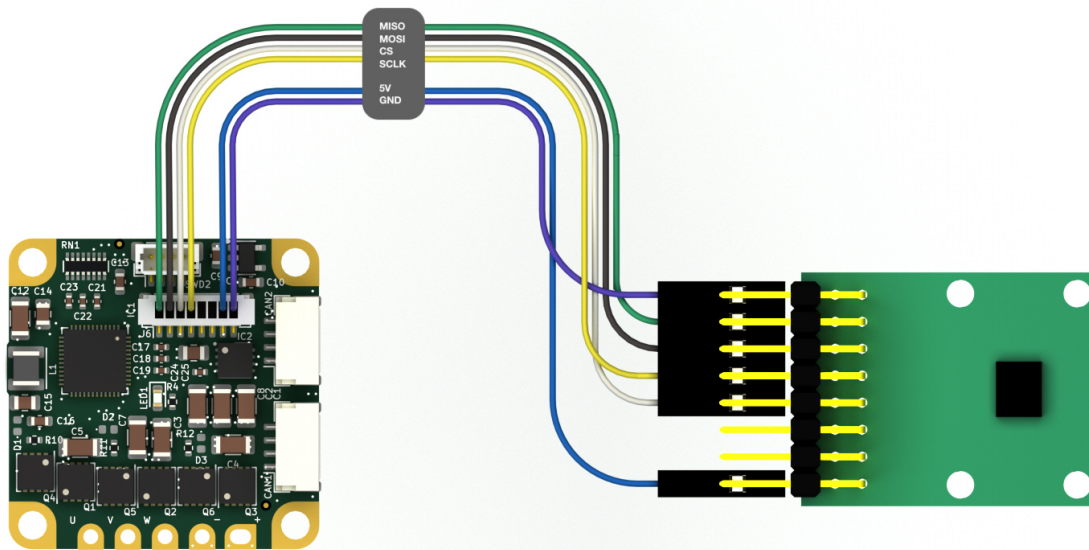
8.1 Overview

This document provides information and guides for using the various angle sensor and encoder types supported by Tinymovr.

8.2 Connector Overview



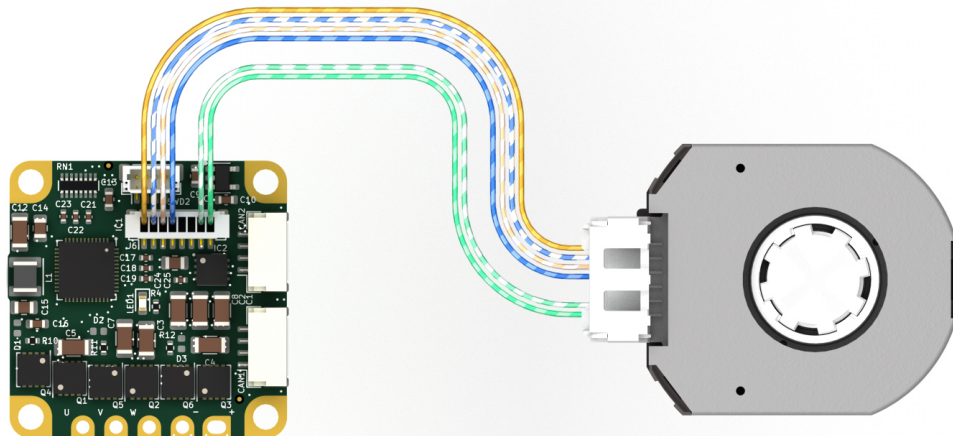




A total of six wires need to be connected: 5V, GND, MISO, MOSI, SCLK and CS.

CUI AMT22

The CUI AMT22 is an absolute angle, 12 or 14-bit capacitive sensor. It is compatible with Tinymovr M5.x.



A total of six wires need to be connected: 5V, GND, MISO, MOSI, SCLK and CS. If using the AMS AMT-06C-1-036 prototype cable, you can additionally connect the cable shield (black wire) to one GND pin on the CAN bus ports or

the SWD port.

8.3.2 Hall Effect Sensor

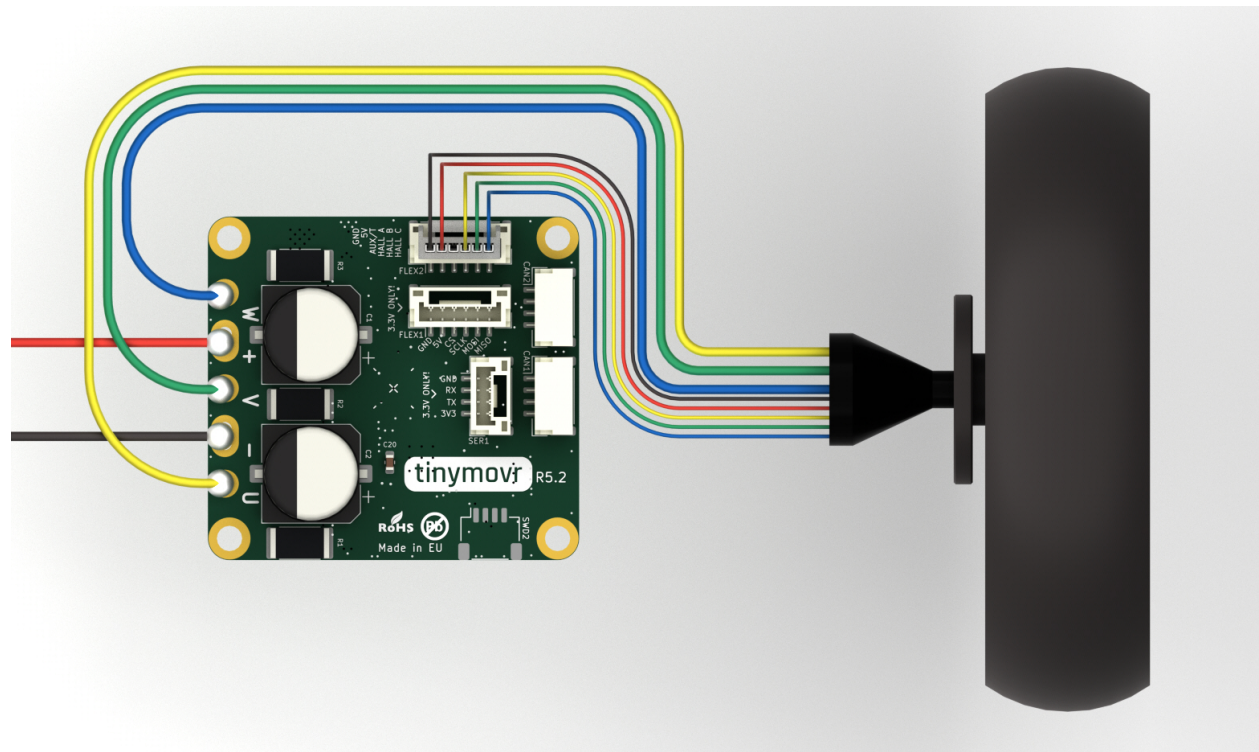
To use Hall effect sensors, you need to connect the sensor's power supply, phases and ground to the correct pins on the FLEX1 header of the Tinymovr R5.2 or greater, or the AUX header of the Tinymovr R5.1. Note the U, V and W pins. These need to be connected to the respective pins of the sensor. The pin labeled AUX/T is an input for a thermistor, but is currently not in use. In addition, power supply and GND pins need to be connected to the sensor.

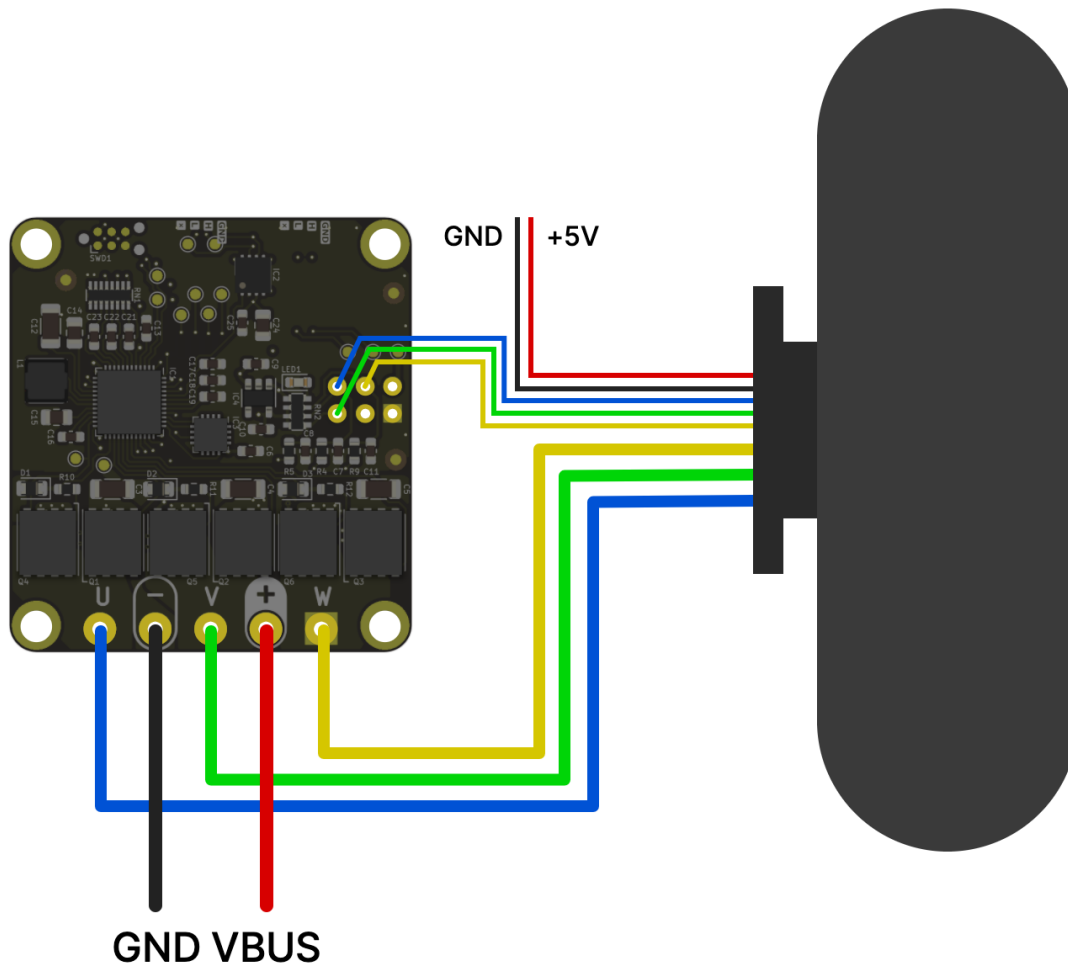
Note: Tinymovr R5.2 and above supply 5V on the FLEX1 power supply pin. You can safely connect this to the Hall effect sensor + terminal.

Tinymovr R5.1 supplies 3.3V on the AUX power supply pin. If your sensor uses 5V, or if it needs more than 50mA, you'll need to provide power externally, e.g. through a dedicated buck converter.

Example

The figures below shows an example of wiring a hubwheel motor to Tinymovr R5.2 and R5.1 respectively, using the embedded Hall effect sensors of the motor for commutation.





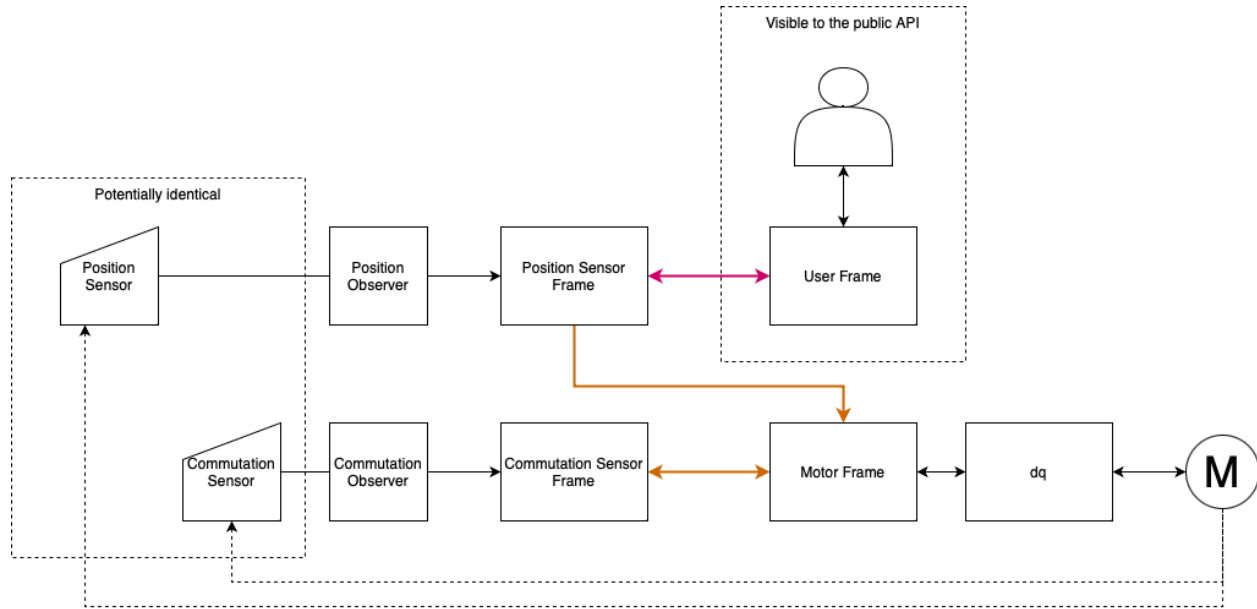
8.4 Units

In Tinymovr, a 'tick' traditionally represents $1/8192$ of a full mechanical rotation. The system utilizes floating-point values, thereby allowing resolution beyond the granularity of a single tick — down to the precision defined by the IEEE754 standard. This means that even when high-resolution sensors (with 16, 18, 20 or more bits) are employed, their precision is fully retained. Internally, sensor measurements are scaled to conform with the 8192-tick representation. In addition, using the Tinymovr client library, you can define commands in any angle unit you wish, such as turns, rads, degrees etc. This gives you freedom in your application beyond the tick representation.

8.5 Reference Frames

In the context of Tinymovr motor control, reference frames are essential for understanding the transformation of sensor data and user setpoints into motor control signals. The following diagram depicts the reference frames and their interconnections:

The diagram below illustrates the flow of data from the physical sensors through various observers and frames, finally reaching the motor.



8.5.1 Position Sensor Frame (PSF)

The Position Sensor Frame (PSF) corresponds to the filtered position sensor data. The main function of this frame is to provide feedback on the estimated position and velocity of the rotor, and therefore provide feedback to the position and velocity control loops. As the homing and trajectory planners also rely on position and velocity estimates, this frame also affects those functions.

8.5.2 Commutation Sensor Frame (CSF)

The Commutation Sensor Frame (CSF) corresponds to the filtered commutation sensor data. In this simplest scenario, the position and commutation sensors are the same, as such the PSF and CSF are identical. The main function of this frame is to provide the estimated rotor angle to the current control loop, so that the electrical angle is derived in the Motor Frame, for Field Oriented Control.

Note: The transform between MF and CSF always has a scale factor of 1. In other words, the commutation sensor is always assumed to be mechanically attached directly to the motor shaft, without reduction.

8.5.3 Motor Frame (MF)

The origin of the Motor Frame (MF) corresponds to the zero electrical angle of the electrical cycle energized during calibration. This is the frame used by current control, and related dq, inverse Park and SVPWM transforms.

8.5.4 User Frame (UF)

The User Frame is the interface exposed to the Tinymovr API, allowing the user to command the motor using position, velocity and current setpoints. This frame is related to the PSF, so that the user commands are predominantly based on the position data, with the commutation aspect being internally managed by the firmware's observer algorithms.

8.5.5 Frame Transforms

Data from the Position and Commutation Sensors is forwarded to their respective observers. The observers are responsible for filtering the sensor readings and providing position and velocity estimates. This processed data is then translated into two separate frames:

1. The Position Sensor Frame, which carries the filtered position data.
2. The Commutation Sensor Frame, which ensures the motor's proper electrical commutation.

These frames are then employed to inform the Motor Frame, which is the final reference before actuating the motor.

As a summary, the following transforms are derived during calibration and are stored in the Tinymovr firmware:

1. UF <-> PSF
2. PSF <-> CSF
3. CSF <-> MF
4. UF <-> MF

Tinymovr makes use of the XF1 library, which has been developed for this purpose and offers convenience functions to perform transforms, derive transforms from data, as well as inverse and constrained transforms.

8.6 Onboard Magnetic

All Tinymovr controllers feature an onboard magnetic absolute angle sensor that allows high precision angle measurement for efficient commutation and highly dynamic motor control. This is enabled by default and does not require any specific setup, apart from initial reference frame calibration.

The onboard angle sensor is enabled by default, so no special configuration is necessary. Should you need to switch to the onboard sensor, use the following commands:

```
tmx.encoder.type = 0
tmx.encoder.bandwidth = 300
tmx.save_config()
tmx.reset() # sensor type change is applied after reset
```

The value of 300Hz in bandwidth is the default value configured based on the characteristics of the onboard angle sensor.

8.7 Sensor Configuration

The sensor configuration consists of two steps. The first step concerns the setup of the individual sensors being used, and the second step concerns sensor selection. The corresponding sections in the device spec are *tmx.sensors.setup*, and *tmx.sensors.select*.

8.7.1 Sensor Setup

Onboard Magnetic Sensor

The Onboard Magnetic Sensor does not require any configuration. In this section the calibration state and any sensor errors can be seen.

External SPI Sensor

The External SPI Sensor requires the correct sensor type to be set before enabling it. Three sensors are currently supported, the MPS MA7xx series, the AMS AS504x series, and the CUI AMT22 series. In addition, here you can see the calibration state and sensor errors.

Hall Effect Sensor

Hall effect sensors generate a specific sequence in the 3 phase Hall effect sensor signal as the rotor moves. By reading this sequence, the rotor position is determined in one of six 60 degree sectors along the electrical cycle.

The Hall Effect Sensor does not require any configuration. In this section the calibration state and any sensor errors can be seen.

8.7.2 Sensor Selection

Sensor selection can be performed for positioning and for commutation. In both cases, the selection should be performed after hardware setup and any sensor setup has been fully completed, namely if using external sensors, the selection of the sensor type. The selection is among ONBOARD, EXTERNAL_SPI and HALL sensors. Once selection is complete, the Tinymovr needs to undergo calibration.

8.8 Examples

8.8.1 External AS5047 Sensor for Commutation and Positioning

Note: This is only supported on the Tinymovr M series, and upcoming Tinymovr R versions

Ensure the hardware is properly connected.

Then, configure the external sensor type as follows:

```
tmx.sensors.setup.external_spi.type = tmx.sensors.setup.external_spi.type.AS5047
```

Then select the *EXTERNAL_SPI* sensor for each of the position and commutation sensors:

```
tmx.sensors.select.commutation_sensor.connection = tmx.sensors.select.commutation_sensor.
↳connection.EXTERNAL_SPI
tmx.sensors.select.position_sensor.connection = tmx.sensors.select.position_sensor.
↳connection.EXTERNAL_SPI
```

At this point, you are ready to perform motor/sensor calibration. This will measure the R and L values of the motor, derive frame transforms and eccentricity compensation tables.

```
tmx.controller.calibrate()
```

After calibration finishes, you should be able to control the motor:

```
tmx.controller.velocity_mode()
tmx.controller.velocity.setpoint = 8192 # 60 rpm
```

The motor should now move at a constant velocity.

Once you have determined that the motor behaves as expected, set to idle and perform another config save to persist the configuration:

```
tmx.controller.idle()
tmx.save_config()
```

8.8.2 External AMT22 Sensor for Positioning and Onboard MA702/704 for Commutation

Note: This is only supported on the Tinymovr M series, and upcoming Tinymovr R versions

Ensure the hardware is properly connected.

Then, configure the external sensor type as follows:

```
tmx.sensors.setup.external_spi.type = tmx.sensors.setup.external_spi.type.AMT22
```

Then select the *EXTERNAL_SPI* sensor for each of the position and commutation sensors:

```
tmx.sensors.select.commutation_sensor.connection = tmx.sensors.select.commutation_sensor.
↳connection.ONBOARD
tmx.sensors.select.position_sensor.connection = tmx.sensors.select.position_sensor.
↳connection.EXTERNAL_SPI
```

At this point, you are ready to perform motor/sensor calibration. This will measure the R and L values of the motor, derive frame transforms and eccentricity compensation tables.

```
tmx.controller.calibrate()
```

After calibration finishes, you should be able to control the motor:

```
tmx.controller.velocity_mode()
tmx.controller.velocity.setpoint = 8192 # 60 rpm
```

The motor should now move at a constant velocity.

Once you have determined that the motor behaves as expected, set to idle and perform another config save to persist the configuration:

```
tmx.controller.idle()
tmx.save_config()
```

8.8.3 Hall Effect Sensor

Note: This is only supported on the Tinymovr R series.

Ensure the hardware is properly connected.

Then select the *HALL* sensor for each of the position and commutation sensors, and configure the observer bandwidth as follows:

```
tmx.sensors.select.commutation_sensor.connection = HALL
tmx.sensors.select.position_sensor.connection = HALL
tmx.sensors.select.commutation_sensor.bandwidth = 200
tmx.sensors.select.position_sensor.bandwidth = 20
```

This sets the type to Hall effect sensor, and each of the commutation and position observer bandwidths. The commutation observer is set to a higher bandwidth value, in order to ensure that commutation is accurate and a runoff scenario is avoided.

Next, you need to set the motor pole pairs:

```
tmx.motor.pole_pairs = 15
```

Next comes tuning of gains. Gains are determined on the tick count of a full mechanical turn of the motor. When using the an absolute sensor, the tick count is fixed to 8192 ticks (the resolution can be higher as the tick count is a floating point value).

When using the Hall effect sensor, the tick count is defined as 8192 ticks in an electrical cycle. Thus, your mechanical cycle tick count is variable, depending on the pole pair count of your motor. Because of this it is possible that the gains need to be updated. Below we present an example of values that work well with a 15 pp hoverboard motor:

```
tmx.controller.position.p_gain = 5
tmx.controller.velocity.p_gain = 0.00001
```

For your own motor, you need to determine these experimentally. Take a look at [PID Tuning](#) for more information.

At this point, you are ready to perform motor/sensor calibration. This will measure the R and L values of the motor, as well as the hall effect sensor sequence.

```
tmx.controller.calibrate()
```

After calibration finishes, you should be able to control the motor. Note that the default reference frame for the hall sensors maps to 8192 ticks per motor electrical cycle. You can change this by modifying the

```
tmx.controller.velocity_mode()
tmx.controller.velocity.setpoint = 100 # around 60 rpm for a 15 pp motor
```

The motor should now move at a constant velocity.

Once you have determined that the motor behaves as expected, set to idle and perform another config save to persist the configuration:

```
tmx.controller.idle()  
tmx.save_config()
```

8.9 Observer Bandwidth

Tinymovr uses a second order observer that filters readings from the sensors, and maintains a position and velocity state. The bandwidth value corresponds to the desired observer bandwidth. It is a configurable value and depends on the dynamics that you wish to achieve with your motor. Keep in mind that high bandwidth values used with motors with fewer pole pairs will make the motors oscillate around the setpoint and have a rough tracking performance (perceivable “knocks” when the rotor moves). On the other hand, too low of a bandwidth value may cause the motor to lose tracking in highly dynamic motions. If you are certain such motions will not be possible (e.g. in heavy moving platforms) you may reduce the bandwidth to ensure smoother motion.

FEATURES

9.1 Trajectory Planner

The trajectory planner can generate acceleration- and velocity-limited trajectories, as well as time-limited trajectories for multi-axis synchronization. The planner executes at the motor control rate for smooth trajectory interpolation.

9.1.1 Trapezoidal Trajectories

Trapezoidal trajectories are named after the shape of the velocity profile, which is a trapezoid. As such, it comprises a constant acceleration phase, where the velocity increases linearly, a constant velocity (cruise) phase, and a constant deceleration phase, where the velocity linearly decreases.

9.1.2 Acceleration and Velocity-Limited Trajectory

The acceleration- and velocity-limited planner generates trajectories that respect constraints on acceleration cruise velocity and deceleration set by the user. The corresponding time windows for each phase are computed by the planner.

Before using the planner, the desired acceleration, deceleration and max velocity limits need to be set. This can be achieved as follows:

```
tm1.traj_planner.max_accel = {max_acceleration} # ticks/sec^2
tm1.traj_planner.max_decel = {max_deceleration} # ticks/sec^2
tm1.traj_planner.max_vel = {mac_velocity} # ticks/sec
```

Once you set the desired acceleration and deceleration parameters, they do not need to be re-set. The parameters can be saved in NVRAM using `tmx.save_config()`.

Once the parameters are set, you can execute a plan to a target position:

```
tm1.traj_planner.move_to(pos_target)
```

As an example, a trajectory is defined and executed as follows:

```
tm1.traj_planner.max_accel = 20000
tm1.traj_planner.max_decel = 20000
tm1.traj_planner.max_vel = 80000
tm1.traj_planner.move_to(100000)
```

9.1.3 Time-Limited Trapezoidal Velocity Trajectory

The time-limited trajectory planner generates trajectories whose acceleration, cruise and deceleration phases are defined by the user. The corresponding acceleration, velocity and deceleration values are computed by the planner. In this sense, it can be thought as the ‘inverse’ process of the acceleration and velocity limited planner.

Similarly to the velocity-limited planner, the plan parameters need to be set first:

```
tm1.traj_planner.t_acc = 2 # seconds
tm1.traj_planner.t_cruise = 10 # seconds
tm1.traj_planner.t_dec = 2 # seconds
```

Now the time-limited trajectory is executed as follows:

```
tm1.traj_planner.move_to_tlimit(1000000)
```

9.1.4 Multi-axis Synchronization

Time-limited trajectories are useful for synchronizing the acceleration, cruise and deceleration phases for multiple axes. For instance, to synchronize three axes with different setpoints, you would configure the same `t_acc`, `t_cruise`, `t_dec` values. This guarantees that the trajectory phases are synchronized. Then you would issue the `move_to_tlimit` commands to each of the three different controllers in sequence:

```
tm1.traj_planner.move_to_tlimit(1000000)
tm2.traj_planner.move_to_tlimit(50000)
tm3.traj_planner.move_to_tlimit(2000000)
```

This will generate one trajectory for each controller, which will start and stop at the same time.

9.2 Homing

The homing feature enables sensorless homing, endstop identification and retraction. The homing feature relies on detecting the mechanical resistance when an endstop is reached. During homing the position error is continuously monitored. Upon exceeding a preset threshold for more than a preset time duration, the motor is considered stalled, and the endstop found.

9.2.1 Configuration

Because the homing planner relies on mechanical resistance of the structure, it is important to set up the corresponding parameters correctly, otherwise the endstop sensing performance can be compromised, and damage to the structure can occur.

There are six parameters in total that control the homing behavior:

- `tmx.homing.velocity`: The velocity at which the motor performs homing
- `tmx.homing.max_homing_t`: The maximum time the motor is allowed to travel before aborting homing
- `tmx.homing.retract_dist`: The retraction distance the motor travels after the endstop has been found
- `tmx.homing.stall_detect.velocity`: The velocity below which (and together with `stall_detect.delta_pos`) stall detection mode is triggered

- `tmx.homing.stall_detect.delta_pos`: The position error above which (and together with `stall_detect.velocity`) stall detection mode is triggered
- `tmx.homing.stall_detect.t`: The time to remain in stall detection mode before the motor is considered stalled

In addition to the above, the phase current while the motor is stopped, until `stall_detect.t` time passes is the maximum allowed phase current, as defined in `tmx.controller.current.Iq_limit`. It is advisable to set this value so that mechanical damage or fatigue is avoided.

9.2.2 Operation

Following proper configuration, the homing operation is initiated as follows:

```
tm1.homing.home()
```

9.3 Flux Braking

Flux braking is an advanced motor control technique used in electric motor applications to achieve controlled deceleration and stopping while minimizing reverse current due to back electromotive force (back-EMF).

FOC decouples the torque-producing and magnetizing currents by aligning the stator current vector with the rotor's magnetic field axis. This alignment allows independent control of both the torque and the magnetic flux components. When braking, the motor controller reverses the direction of the torque-producing current in the q-axis, generating a counter-torque that opposes the rotor's motion, thus slowing it down. This deceleration would typically cause a reverse current due to the reduction in kinetic energy and the presence of back electromotive force. The flux braking technique effectively converts the motor's kinetic energy into heat by increasing the d-axis current, which allows for controlled deceleration while minimizing the impact of back-EMF and preventing reverse current.

Two parameters control flux braking:

1. `tmx.controller.current.max_Ibus_regen`: The maximum current (in amperes) allowed to be fed back to the power source before flux braking activates. By adjusting this value, you can control the regenerative braking threshold and determine when flux braking should take effect.
2. `tmx.controller.current.max_Ibrake`: The maximum current (in amperes) allowed to be dumped to the motor windings during flux braking. By setting this value to zero, you can deactivate flux braking. Adjusting this parameter allows you to manage the braking torque and the heat generated during the braking process.

UPGRADING FIRMWARE

10.1 Introduction

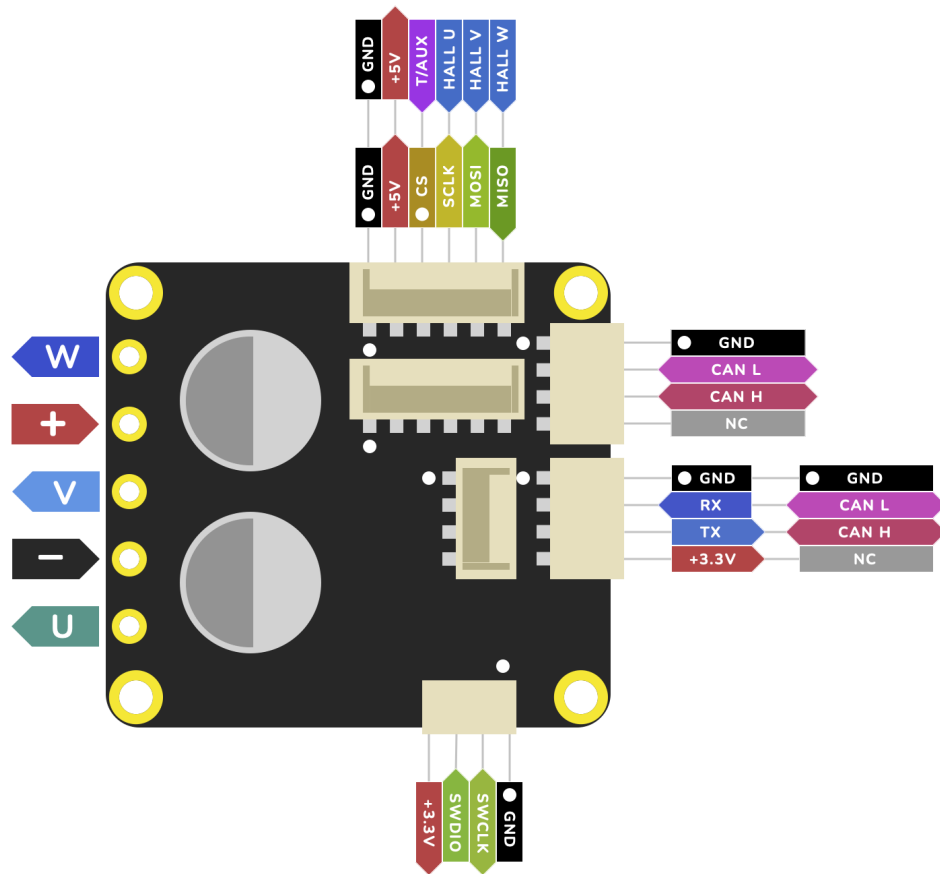
It is possible to upgrade the firmware of your Tinymovr device in different ways, depending on the board you have:

1. *Upgrade using DFU and CAN bus* connection (fw 1.6 or later, boards shipped starting 8th Sep. 2023)
2. *Upgrade using ActiveFlashlight and UART* connection (fw 1.5 or earlier, boards shipped until the 7th Sep. 2023)
3. *Upgrade using VSCode and J-Link* (all boards)

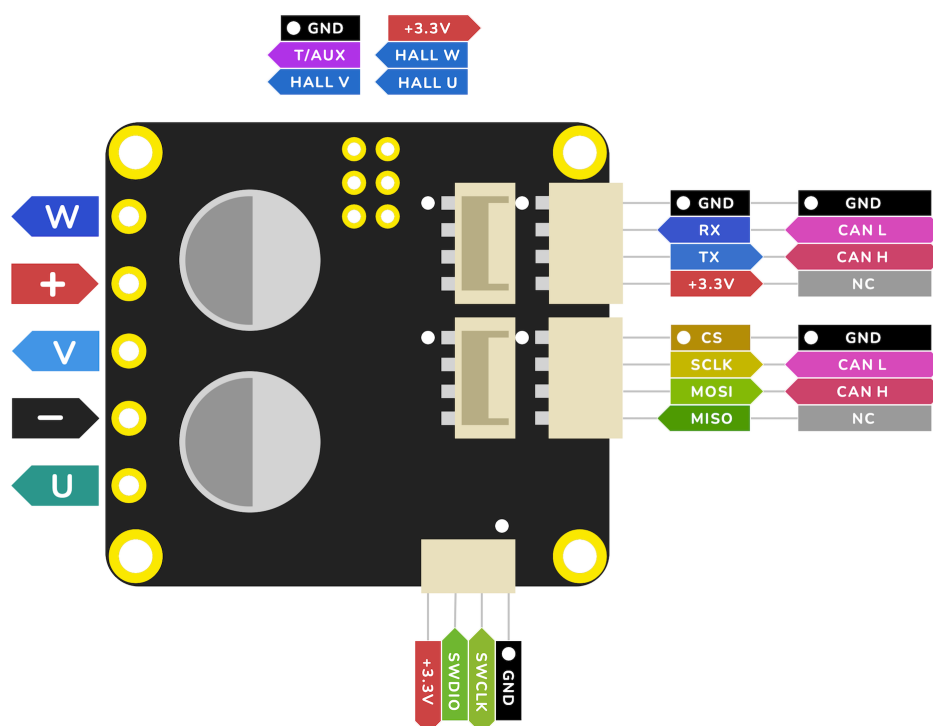
A revision of the hardware connectivity and each of the above methods is provided below.

10.2 Connectivity

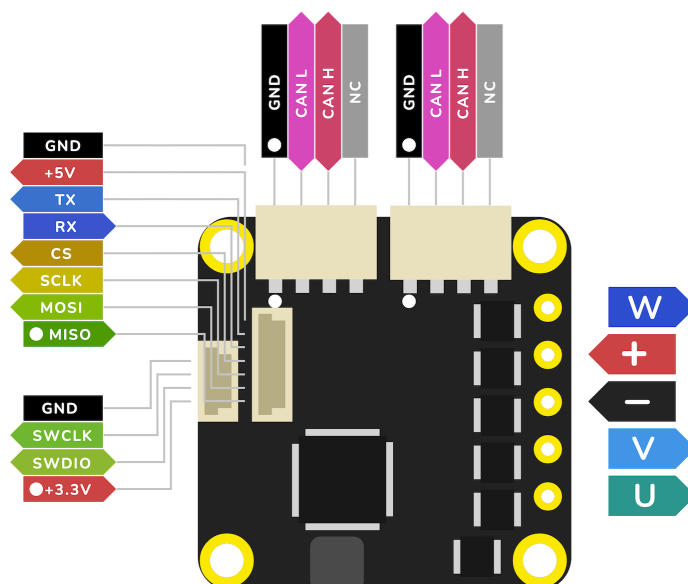
10.2.1 R5.2



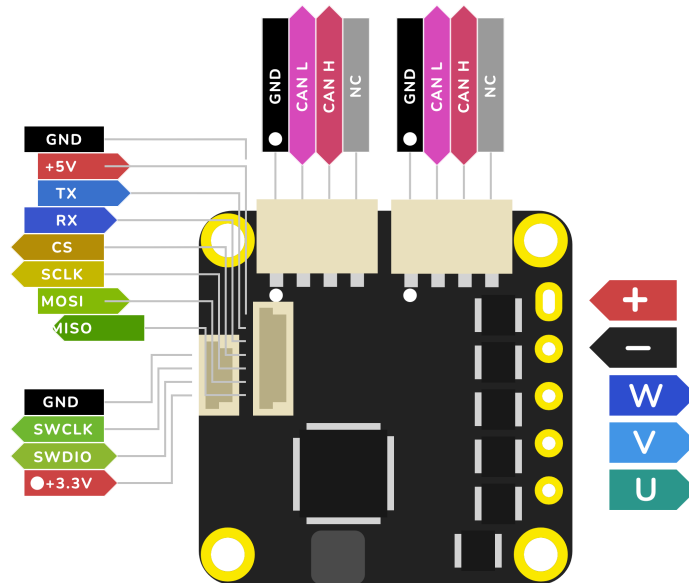
10.2.2 R5.0, R5.1



10.2.3 M5.1

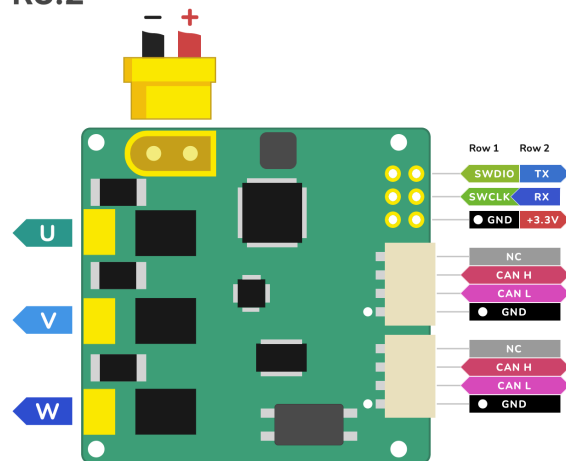


10.2.4 M5.2

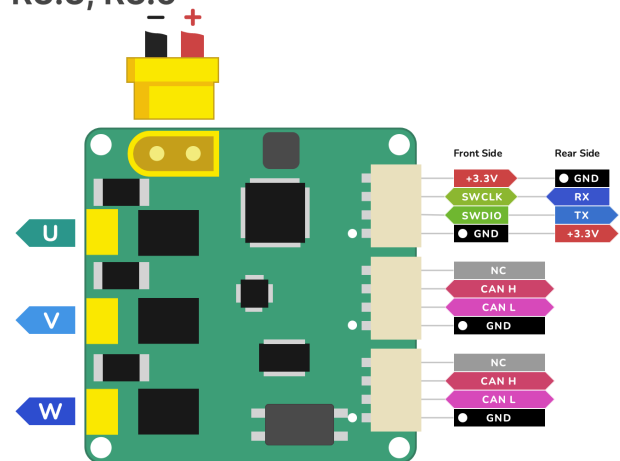


10.2.5 R3.x

R3.2



R3.3, R3.5



10.3 Upgrade using DFU and CAN bus

Since firmware 1.6, we have streamlined the firmware update process with a bootloader that supports DFU through the native CAN bus interface. This guide will walk you through the steps to upgrade your Tinymovr's firmware.

10.3.1 Prerequisites

1. Install the Tinymovr Package:

If you haven't already, install the Tinymovr package. This package also comes with the required dependencies:

```
pip install tinymovr
```

2. Download the Firmware:

Head to the [Tinymovr Releases Page](#) and download the precompiled binary of the firmware that matches your board's hardware revision. Firmware binaries follow the format: `tinymovr<firmware_version>-<hardware_revision>.bin`. For instance, you might come across a file named `tinymovr1.5.0-M5.1.bin`.

10.3.2 Flashing the Firmware

1. Set Up Your CAN Interface:

Ensure your CAN interface is connected and configured correctly with Tinymovr.

2. Enter DFU Mode:

Launch the Tinymovr CLI:

```
tinymovr_cli
```

Once inside the CLI, type the following to enter DFU Mode:

```
tmx.enter_dfu()
```

Then exit the CLI by typing `exit`.

If you are getting an exception after this command, it means that either your Tinymovr does not come with the DFU bootloader, so you will have to [Upgrade using ActiveFlashlight and UART](#), or your board is already in DFU mode.

2. Run the Script:

With the firmware .bin file you downloaded, execute the DFU script:

```
tinymovr_dfu --node_id=<your node id> --bin=path/to/your/downloaded/tinymovr
-><version>-<revision>.bin
```

Replace the placeholders in the path with the appropriate values based on where you've saved the .bin file and its name.

Example:

```
tinymovr_dfu --node_id=<your node id> --bin=~/.Downloads/tinymovr1.5.0-M5.1.bin
```

To forgo the automatic reset post-flash, append the `--no-reset` flag:

```
tinymovr_dfu --node_id=<your node id> --bin=~/.Downloads/tinymovr1.5.0-M5.1.bin --no-  
↪reset
```

3. Follow the Script's Prompts:

The script initially checks if the device's current firmware matches the .bin file. If they're identical, it will bypass the flashing process. If not, it will erase the prior firmware, flash the new version, and validate the update.

4. Concluding Steps:

If you skipped using the `--no-reset` flag, the device will reboot automatically after the update. If you used the flag, you'd need to power off and on the Tinymovr by hand.

You're all set! Your Tinymovr is updated and primed for use with the fresh firmware.

Note: Before updating, it's a wise move to backup your current firmware and settings. Always pore over any version-specific instructions or release notes accompanying fresh firmware updates to stay informed.

10.4 Recovery Mode

The DFU mode can be triggered upon device power up, which is useful to deal with problems resulting from inaccessible user firmware (e.g. mismatch between firmware and hardware revision). This feature will put the device into DFU mode immediately after power up. To use recovery mode, run the following command:

```
tinymovr_dfu --node_id=<your node id> --recovery
```

Then follow the instructions to trigger DFU mode. After this step, you will be able to *Upgrade using DFU and CAN bus*. You will need to know the device node ID to use this feature.

10.5 Upgrade using ActiveFlashlight and UART

10.5.1 Prerequisites

For this method you will need a USB to UART adapter or similar device to connect to Tinymovr's UART interface, such as an FTDI interface or similar. Such devices are very common and inexpensive. Please follow the diagram below to correctly setup the UART interface.

Warning: The UART port on Tinymovr is NOT 5V tolerant. Applying 5V voltage will immediately damage the onboard PAC5527 controller. Please use only 3.3V for UART communication.

Warning: The UART port on Tinymovr offers a 3.3v output for driving very light loads (30mA absolute max). Tinymovr cannot be powered by this pin. In addition, most UART adapters offer 5V power, which is incompatible with Tinymovr. **In short: If in doubt, leave this pin disconnected.**

10.5.2 Flashing the Firmware

Warning: This is a legacy upgrade approach that is maintained here to facilitate existing users. New users should *Upgrade using DFU and CAN bus*

Qorvo, the maker of the PAC chip used in Tinymovr, provides an application to interface with the bootloader environment using UART and enable firmware upgrades. It is available through the [Qorvo website](#) as an archive named “PAC55xx ActiveFlashLight Bootloader”. Please download and extract the archive and follow these steps:

1. Inside the “resources” folder run the ActiveFlashLight application
2. Power on Tinymovr
3. Under ‘COMMUNICATION’ click on ‘Connect’
4. Power off and on Tinymovr. The application status should now indicate that it is connected to the device.
5. Under ‘PROGRAMMING’ click on ‘Browse...’ and find the latest ‘tinymovr<firmware_version>-R<board_revision>.bin’ file for the board being used, which can be downloaded from [releases](#). Also take note of any version specific instructions in the release notes.
6. Under ‘PROGRAMMING’ click on ‘Auto Program’. The application will erase, flash and verify the device with the new firmware. Wait for the process to complete
7. Once complete, power off Tinymovr and quit the application

You should now be able to use Tinymovr as usual with the new firmware.

10.6 Upgrade using VSCode and J-Link

Please see *Using VSCode*.

Warning: Depending on the hardware revision and the batch, either a vertical or a sideways JST-SH connector is used for SWD. In any case, the pin order shown in the above figures is correct. Please consider this when connecting your SWD adapter.

GIMBAL MOTORS

11.1 Introduction

Tinymovr can drive gimbal-style brushless motors in open loop current control, by disabling current sensing.

Warning:

- This mode is not recommended for Tinymovr M5, which is a specialized gimbal driver and can do closed loop current control with gimbal motors.

What is a gimbal motor anyway?

A gimbal motor is a 3-phase brushless motor whose stator is wound with many turns, and as such exhibits much larger resistance and inductance compared to the high-current brushless motors used to provide lift to drones, rc planes etc. Gimbal motors are used in... well... camera gimbals mainly. Similar motors can be found in light robotic joints. Such motors offer smooth motion and require small currents to produce torque compared to the ‘high current’ brushless motors. This in turn can help minimize the size of the motor driver and associated wires etc. Note that we refer to reduction of current through the stator windings and not the power converted to heat as a result of Joule heating of stator windings, which is still the same for the same amount of torque.

To achieve closed-loop current control, motor controllers such as Tinymovr use current measurement resistors in each phase (usually in series with the low-side mosfet) to estimate the current in the motor windings. These resistors have low resistance (in the range of a few milliOhms at most), to limit power dissipation in the resistor and allow operation in wide current ranges (up to several tens of Amps, even hundreds). The drawback is that current measurements exhibit noise of 100s of mA, and as such do not offer accurate measurements of small currents.

The gimbal mode of Tinymovr disables closed-loop current control. The commanded currents are converted to voltages using basic resistance and inductance formulas.

11.2 Enabling Gimbal Mode

Warning:

- DO NOT perform calibration on a gimbal motor without setting gimbal mode first! There is a risk of damaging the motor and board.
- Using arbitrary resistance and inductance settings can damage your motor and board.

To enable gimbal mode, set the motor configuration as follows:

```
>>>tm1.motor.type = 1 # gimbal type
>>>tm1.motor.R = {motor_resistance}
>>>tm1.motor.L = {motor_inductance}
>>>tm1.motor.I_cal = {calibration_current}
```

Example

```
>>>tm1.motor.type = 1
>>>tm1.motor.R = 5
>>>tm1.motor.L = 2e-3
>>>tm1.motor.I_cal = 0.5

>>>tmx.set_motor_RL(5, 0.002)
```

This specifies a motor with 5 Ohms resistance, 2 Millihenries inductance and 0.5Amps calibration current. Alternatively, using the units interface:

```
>>>tm1.motor.type = 1 # gimbal type
>>>tm1.motor.R = 5 * Ohm
>>>tm1.motor.L = 2e-3 * Henry
>>>tm1.motor.I_cal = 0.5 * Amps
```

Control that the settings are correct:

```
>>>tm1.motor
```

Ensure the values above are correct. You can now calibrate the motor:

Set Gimbal Motor Parameters

>	comms	
▼	motor	
<input type="checkbox"/>	R	1.000000 Ω
<input type="checkbox"/>	L	0.002000 H
<input type="checkbox"/>	pole_pairs	7
<input type="checkbox"/>	type	GIMBAL
<input type="checkbox"/>	offset	0
<input type="checkbox"/>	direction	1
<input type="checkbox"/>	calibrated	False
<input type="checkbox"/>	I_cal	0.500000 A
<input type="checkbox"/>	errors	(no flags)

```
>>>tm1.controller.calibrate()
```

Because the motor is set as gimbal, calibration will bypass resistance and inductance measurement, and will only calculate pole pairs, offset and direction. After calibration *tmx.motor.calibrated* should have a value of True:

Calibrate Motor

▼ controller	
<input type="checkbox"/> state	0
<input type="checkbox"/> mode	0
<input type="checkbox"/> warnings	(no flags)
<input type="checkbox"/> errors	(no flags)
> position	
> velocity	
> current	
> voltage	
calibrate	→
idle	→
position_mode	→
velocity_mode	→
current_mode	→
set_pos_vel_setpoints	→

```
>>>tm1.motor.calibrated
True
```

11.3 Controlling the Motor

Gimbal mode has identical functionality as the default high-current mode. Position, velocity and current control modes are supported. Depending on your motor characteristics, you may have to tune the control gains to achieve optimal performance. In addition, due to the fact that current control is open loop, high angular velocities may not be available.

PID TUNING

Tuning PID gains can be an iterative process, and it's essential to approach it systematically to achieve the desired performance. Here's a basic guide for tuning the PID gains for your Tinymovr:

12.1 Preliminaries:

- Ensure the motor won't cause damage or injury if it starts to oscillate or behave unexpectedly. Fasten the stator assembly to avoid loosening in case of excessive oscillations.
- If the default gains cause oscillations, reduce all gains until the system is stable.

12.2 Tuning the Velocity Controller

12.2.1 Proportional Gain (P)

1. Gradually increase the proportional gain until the system starts to oscillate when given a step change in velocity command.
2. Note this gain value down. This is the critical gain.
3. Set the velocity proportional gain to roughly half of the previous value for a balance of performance and stability.

12.2.2 Integral Gain (I)

1. With the proportional gain set as above, slowly increase the integral gain.
2. If the motor starts to oscillate, decrease the integral gain slightly until stability is achieved.
3. The role of the integral gain is to eliminate steady-state errors. Ensure that with the selected gain, errors in the steady state (like due to friction or minor disturbances) are reduced.

12.3 Tuning the Position Controller

12.3.1 Proportional Gain (P):

1. Start increasing the proportional gain. The system will use the already-tuned velocity controller as an intermediate step.
2. Increase until you notice overshoot in the position response. If overshooting is excessive or oscillations start, reduce the gain slightly until satisfactory performance is achieved without instability.
3. Ideally, the response should be quick to reach the desired position but without excessive oscillations.

12.4 Tips and Considerations:

- Always make changes in small increments.
- If the system becomes unstable at any point, revert to previous settings and proceed more cautiously.
- Remember that real-world systems have nonlinearities and disturbances. Your tuning might need tweaking under different conditions or loads.

12.5 Final Testing:

Once you believe you've set the gains appropriately:

1. Test across the entire range of operational conditions to ensure stability and performance.
2. Introduce disturbances if possible (like changes in load) to verify robustness.
3. Adjust as necessary based on real-world performance.

CONTROL PRINCIPLES

This document provides a high-level introduction to the essential control principles in Tinymovr. Additional documentation regarding specific high-level control concepts is available in the [Features](#) section.

13.1 Permanent Magnet Synchronous Motors (PMSMs)

PMSMs is a category which includes the majority of hobby-grade brushless motors. PMSMs produce torque through the interaction of the magnetic field of the rotor permanent magnets with the magnetic field generated by the stator coils. The stator magnetic field “rotates” at a rate that is an integer multiple of the rotor rate of rotation, this is why this type of motor is termed “synchronous”.

This is also why, the motor controlled needs to be able to estimate the angular position of the rotor, in order to derive the stator magnetic field that produces torque. Many controllers estimate the rotor using magnetic encoder readings, which is then converted to electrical angle in software. This is termed *sensored estimation*, and this is the method employed in Tinymovr. In contrast, ESCs such as those used in drones use voltage feedback from one of the motor phases to estimate rotor position. This is termed *sensorless estimation*.

13.2 Field Oriented Control (FOC)

Field Oriented Control (FOC) is a control scheme employed in brushless motor applications, offering superior performance compared to other algorithms. Although FOC requires more computational power due to its complexity, advancements in technology have made it increasingly accessible. Modern 32-bit microcontrollers can now execute FOC algorithms at high rates, contributing to its widespread adoption. By directly controlling the current in the motor phases, FOC enables precise torque management, resulting in enhanced performance and efficiency for various applications.

The FOC algorithm comprises three main elements:

1. Precise 3-phase current estimation
2. Transformation of 3-phase measurements to rotating frame.
3. Regulation of the rotational frame variables.

13.2.1 Measuring Phase Currents

In order for FOC to work, the currents in the three phases (U, V, W) of the motor must be precisely estimated. This is done with the help of shunt resistors. Shunt resistors are high-power elements with low, precisely known resistance values, that are placed in series to the low-side mosfet sources, and enable current measurement by measuring the voltage differential across their terminals. Hardware and software filtering techniques improve current measurement accuracy.

As the motor phases are connected, current measurement in each of the three phases is redundant, and measurement in two phases would theoretically suffice. Nonetheless, having all three measurements allows for better accuracy through averaging of measurement errors. This is why Tinymovr uses three phase current measurement.

13.2.2 Reference Frames

Next, the 3-phase measurements are transformed to the rotating frame of the rotor which is termed dq. The relevant transformation is known as the dq0 transform. The resulting quantities are the direct (d) current and quadrature (q) current. Motor torque is attributed to the quadrature component, while the direct component is minimized (except in the case of *Flux Braking*).

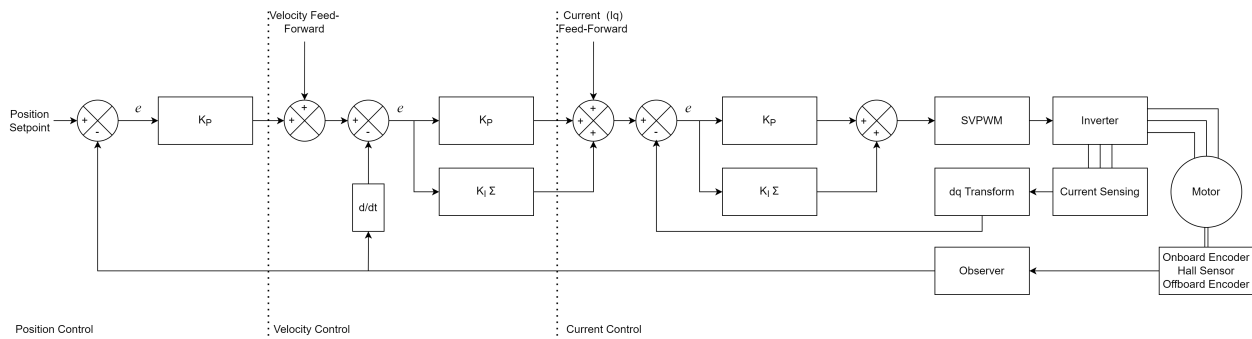
13.2.3 Current Regulation and Motor Parameter Identification

Because the d and q quantities represent DC signals in the rotational frame, it is possible to apply PI regulation to control current. For optimal regulation, the PI current regulation requires correct identification of proportional and integral gains. Tinymovr uses the method proposed in [1] to calculate the gains from the resistance and inductance motor parameters. The parameters are measured automatically by the firmware during the calibration procedure.

Thus the whole process is automated, and you don't need to worry about it.

13.3 Control loop Overview

On top of the FOC loop, Tinymovr implements an embedded control loop. This control loop runs at a 20kHz rate.



13.3.1 Position mode

This is the most versatile mode, it accepts a position setpoint, and additional velocity and current feedforward terms.

You can tune separately each gain of the loop.

P_p: The gain of the position proportional term

P_v: The gain of the velocity proportional term

I_v: The gain of the velocity integral term

The integral term is especially useful for tracking positions at low velocities. You can set it to zero for greater position control bandwidth.

Velocity Integrator Deadband

The integrator deadband is a setting that is useful in minimizing the “hunting” phenomenon, where the rotor oscillates around the setpoint at standstill. This phenomenon is due to interaction of integrator windup and the non-linearities of cogging torque. The integrator deadband feature is only active in position control mode and disables the integrator term update within a configurable window around the position setpoint (the “deadband”).

Take a look at the [*controller.velocity.deadband*](#) endpoint for specifics.

Example applications

Robot joint control, CNC axis.

13.3.2 Velocity mode

In this mode the controller accepts a velocity setpoint, and an optional current feedforward term.

Example applications

Drone and aircraft propeller

Large airframes where constant angular velocity is desired, without exceptionally high RPM (i.e. 5000rpm or less).

Industrial Automation

Where constant RPM is required, regardless of load. Pumps, ventilators, cutters, drills, etc.

Wheel propulsion

For brushless wheel based projects like differential drives or rovers.

13.3.3 Current mode

This is the most direct mode, where you can specify current setpoints, that are direct inputs to the FOC algorithm.

Example applications

Force based control

Robots controlled in admittance or impedance.

E-scooter

Where the throttle input is mapped to the current target. It translates to the acceleration of the vehicle, which feels more natural than a velocity input.

13.4 Further Reading

Additional documentation regarding specific high-level control concepts in Tinymovr is available in the [Features](#) section.

[Vector Control \(Wikipedia\)](#)

[Sensorless Field Oriented Control of Brushless PMSMs](#)

13.5 References

[1] [High Performance Brushless DC Motor Control](#)

TROUBLESHOOTING

Please take a look at [Github Discussions](#) and [Github Issues](#) , and ask any questions there.

You are more than welcome to join the [Tinymovr Discord Server](#) and ask any questions there, or have a chat with us!

15.1 Overview

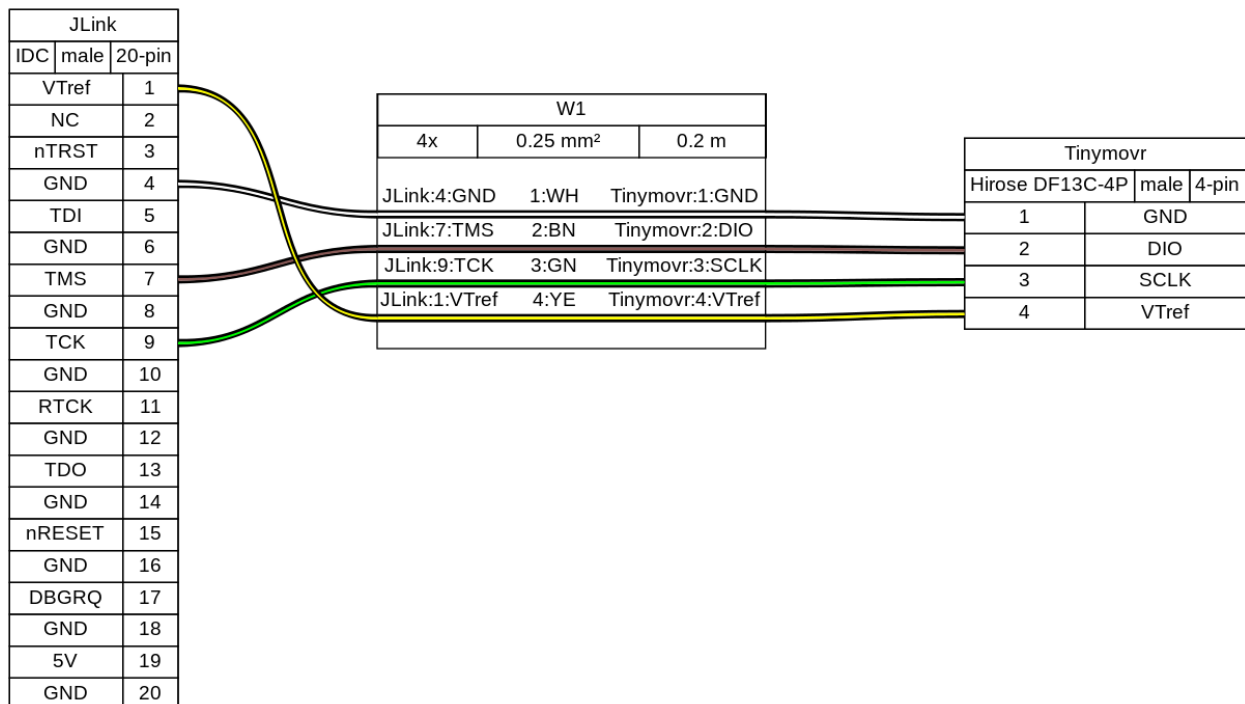
This document provides a guide for setting up a development environment for developing for Tinymovr, including firmware and client software (Studio).

15.2 Hardware Connections

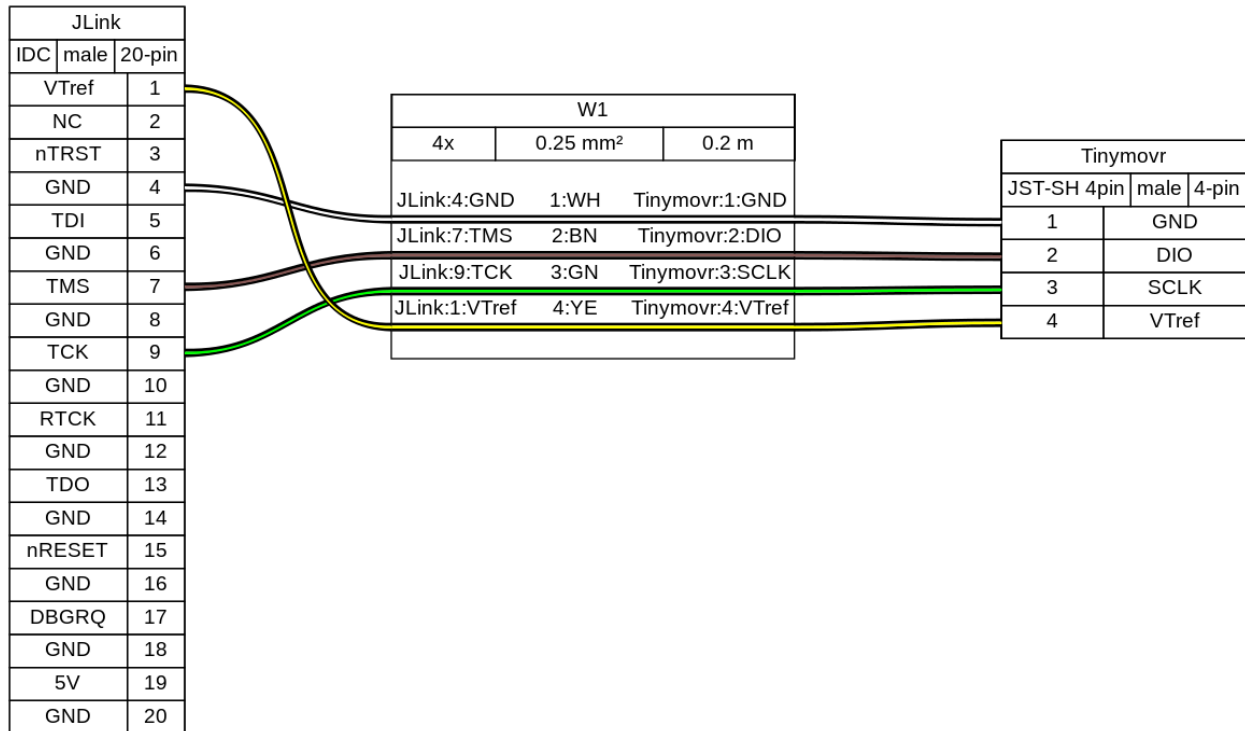
For proper debugging capabilities you will need a Segger J-Link adapter. Unfortunately original J-Link adapters are quite expensive. A more affordable option is the J-Link EDU adapter at around \$60 or the J-Link EDU mini adapter at around \$20. In addition, there are J-Link clones that can be purchased for very low prices on ebay or Aliexpress. However, reliability of these clones is not guaranteed, and we do not provide support when using these clones.

With the board and J-Link adapter powered off, connect the J-Link to Tinymovr as shown below:

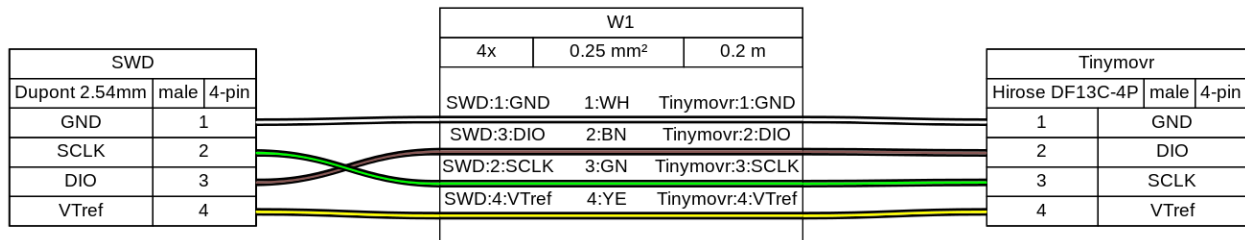
Connection directly to J-Link adapter for R3.x:



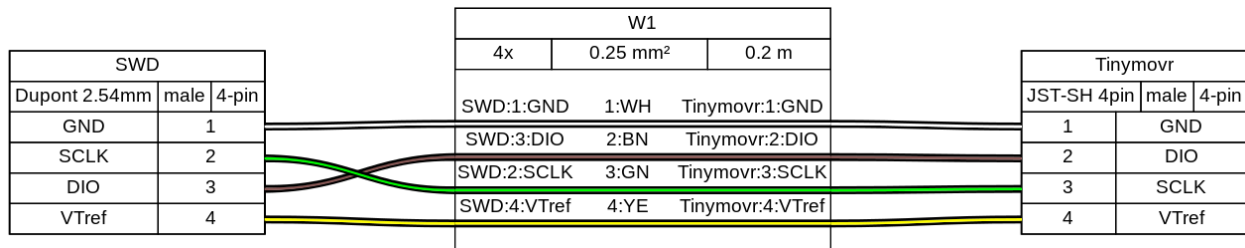
Connection directly to J-Link adapter for R5:



Connection with SWD adapter (e.g. isolator) for R3.x:



Connection with SWD adapter (e.g. isolator) for R5:



(diagrams made with [Wireviz](#))

As of Tinymovr 1.0.1, it is no longer necessary to patch the `JLinkDevices.xml` file with the PAC additions, in order for JLinkGDBServer to work properly. Relevant files are included in the repo.

15.3 Preparation

First, clone the Tinymovr repo:

```
git clone https://github.com/yconst/Tinymovr
```

The Tinymovr repo includes the firmware source code and supporting files, however the PAC55xx SDK is not included due to licensing restrictions imposed by Qorvo. Thus, you will need to [download it from the Qorvo website](#), where you will need to supply your email.

The file comes in a zipped installer exe (!), which all it does is extract the contents to a directory. Navigate to the extracted files directory and copy the 'pac55xx_sdk' directory inside the Tinymovr repo:

```
cp -r <pac55xx_sdk_location> <tinymovr_location>/firmware/pac55xx_sdk/
```

In any case, the target directory should be named pac55xx_sdk. The above copy command ensures this.

Now you have the required PAC SDK almost ready. There is a small patch that you will need to apply in the pac55xx_sdk directory. It is suggested to use the [Python patch package](#), which is cross-platform. If you do not have the package, install using pip:

```
pip3 install patch
```

Then:

```
cd <tinymovr_location>/firmware  
python3 -m patch sdk_patch.patch
```

As a final step, install the [GNU Arm Embedded Toolchain](#).

If you are in Windows you will also need to install GNU make. This is rather easy in Windows 10 or later:

```
choco install make
```

15.4 Using VSCode

Our development environment for Tinymovr utilizes VSCode and is designed with cross-platform compatibility in mind. We support all major operating systems: Linux, MacOS, and Windows. This method allows for a unified approach to building, flashing, and debugging firmware.

We take great pride in creating this in-house, cross-platform development solution, as an alternative to the official Windows-only solution, providing enhanced flexibility and simplifying the process of developing for Tinymovr across multiple platforms.

15.4.1 Configuring

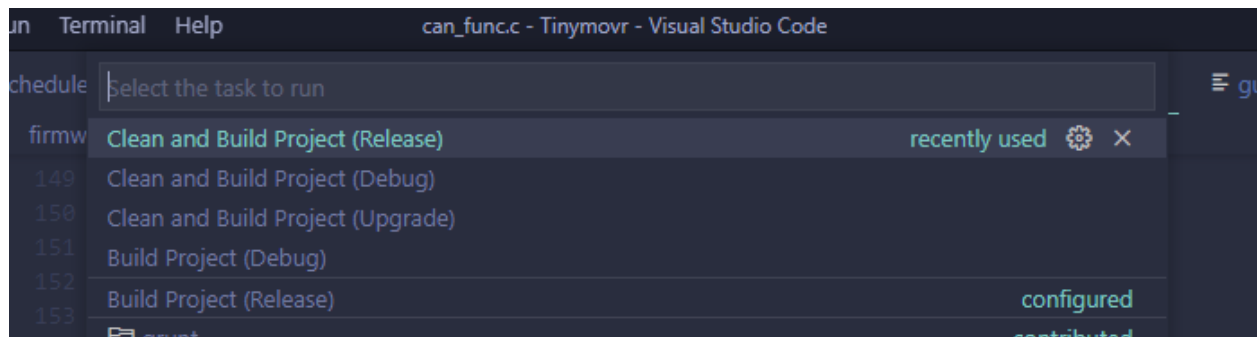
You'll need to configure the path to the JLink debug server in `.vscode/launch.json`. The configuration blocks look like this:

```
"windows": {
  // Update the path below as necessary
  "serverpath": "C:\\Program Files\\SEGGER\\JLink\\JLinkGDBServerCL.exe"
},
"linux": {
  // Update the path below as necessary
  "serverpath": "/opt/SEGGER/JLink/JLinkGDBServer"
}
```

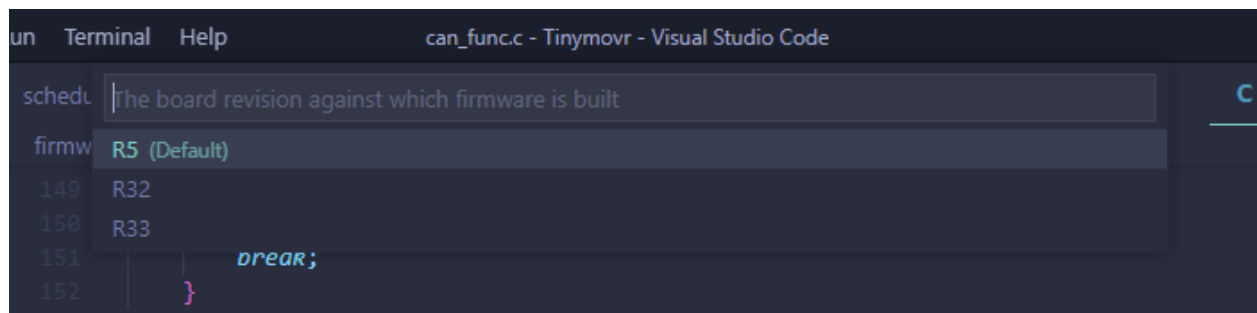
There are several instances of such blocks in the file that need to be updated.

15.4.2 Building

To try out a test build from within VSCode select **Terminal -> Run Task...** from the menu bar, and select **Clean and Build Project (Debug)**.



Tinymovr implements board revision selection using an input menu with a few predefined choices.



Select the board revision against which you are compiling from the list. The build process will start, it usually takes less than half a minute. After it finishes, you should end up with a `build/` directory inside `firmware/` and there you should see the files `tinymovr_fw.elf`, `tinymovr_fw.hex` and `tinymovr_fw.bin`.

More about Hardware Revisions

Tinymovr passes the `BOARD_REV_XX` argument regarding the board revision to the compiler. This is used in the firmware to configure the hardware accordingly. Invoking a make command with the `REV` argument is as follows:

```
make debug REV=R5
```

Currently the following hardware revision values are supported:

- R32
- R33
- R50
- R51
- R52
- M5

15.4.3 Flashing and Debugging

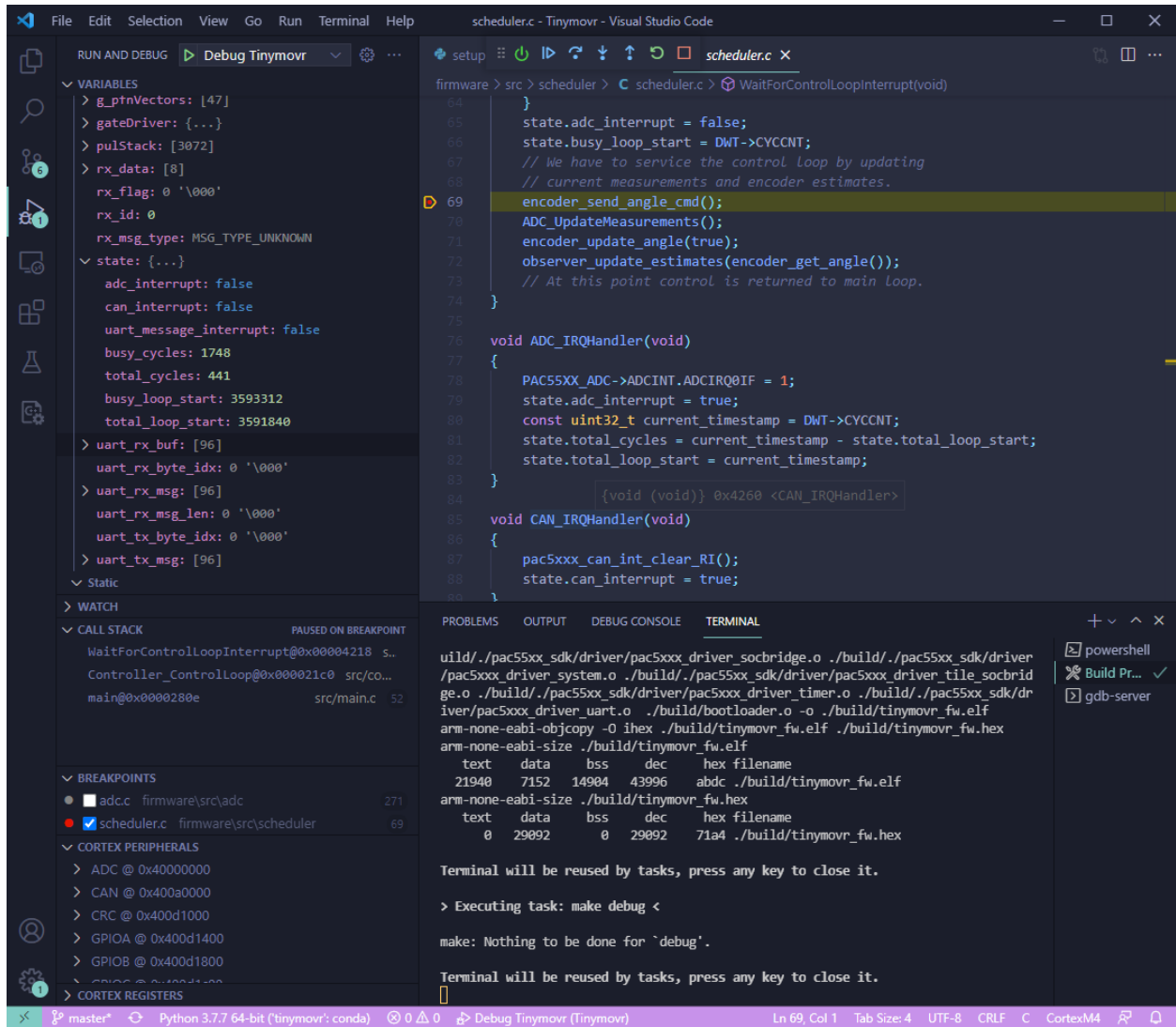
Before debugging, make sure the J-Link drivers and software is installed. The drivers and software, together with instructions, can be found in the [Qorvo website](#), under the download ‘Segger J-Link Support’. This download includes a necessary patch to enable J-Link to work with Qorvo devices. Instructions on how to apply the patch are included in the download.

The Tinymovr repo includes all VSCode settings configured, except for the `JLink serverpath` variable in `launch.json`, which you’ll need to update to reflect your system. Note that there are multiple instances in the file, you’ll need to update all of them.

We offer various VSCode launch configurations to suit different development and debugging tasks, including remote Tinymovr flashing debugging using a remote JLink server. These are are briefly outlined below.

Rebuild Debug and Start Session

This is a convenience configuration that does a clean debug build, flashes it and launches a debug session. After a while the debug session should start and you should see a screen like below:



Rebuild Release and Flash

This is a convenience configuration that does a clean release build and flashes it to Tinymovr.

Flash Built Binary and Start Session

This will flash an already built binary and start a debug session. It assumes a binary already exists in the `firmware/build/` directory.

Flash Built Binary

This will flash an already built binary. It assumes a binary already exists in the `firmware/build/` directory.

Attach to Target

This will attach to an already running target.

Flash Built Binary and Start Remote Session

This will connect to a remote JLink server, upload the firmware to the remote Tinymovr device, and start a remote debug session. You will need to have a JLink Server configured on a network attached device, such as a Raspberry Pi ([here is a good guide on how to do that](#)). You will also need to input the remote device's IP address to the `ipAddress` field of the `launch.json` file.

Note that the launch configurations can be selected and initiated from the VSCode “Run and Debug” pane. You can also hit F5 to launch the currently selected configuration.

Congrats! You are now fully set to start with Tinymovr development!

15.5 Using Eclipse

Eclipse is no longer supported. Consider *Using VSCode* instead.

15.6 Setup Studio for Development

Tinymovr Studio is a Python application and as such can be easily set up to facilitate development. The approach is to use `pip` to install Tinymovr in develop mode, from a local copy of the (git repo `<https://github.com/tinymovr/Tinymovr>``_`). This allows any changes you make to the local code to be immediately available when you run the executable (```tinymovr, tinymovr_cli, or tinymovr_dfu`).

Note: We recommend installing Tinymovr in a virtual environment. [Here is a quick tutorial on how to setup a virtual environment using Conda](#).

Clone the Tinymovr repo as outlined in the first step of *Preparation*. Then:

```
cd <tinymovr_location>/studio/Python/  
pip3 install -e .
```

or to enable GUI support:

```
pip3 install -e .[GUI]
```

This will install the Tinymovr Studio in develop mode. Now, `tinymovr`, `tinymovr_cli` and `tinymovr_dfu` will use the local Tinymovr Studio code.

Happy coding!

15.7 Custom Device Definitions

Tinymovr uses YAML files for device specification (the spec). Device spec files reside in `{tinymovr_package_dir}/studio/Python/tinymovr/specs/`. This is where files covering Tinymovr firmware 1.3.x to 1.6.x are situated by default.

You can add custom device spec files in this directory to tell Tinymovr Studio GUI, CLI and library how to communicate with custom devices. For instance, assuming that you have installed Tinymovr Studio using `pip3`, you will need to find the directory of the tinymovr package first using the command below:

Windows

```
python -c "import os, tinymovr; print(os.path.dirname(tinymovr.__file__))"
```

Macos and Linux

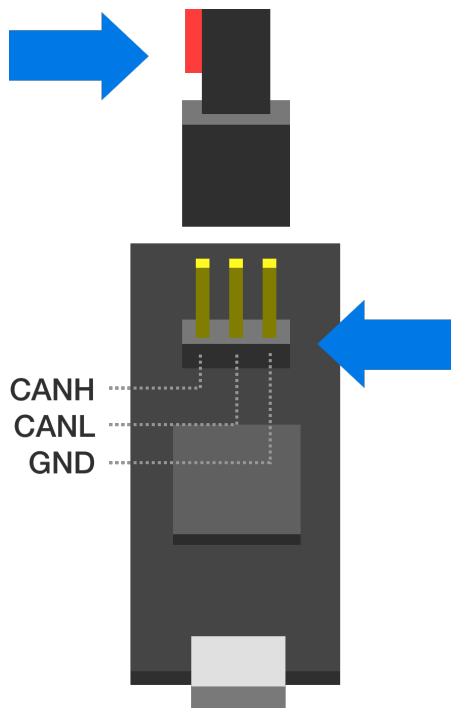
```
python3 -c "import os, tinymovr; print(os.path.dirname(tinymovr.__file__))"
```

Then, paste your custom spec file to `{tinymovr_package_dir}/studio/Python/tinymovr/specs/`. Tinymovr should correctly discover your custom device.

HARDWARE ERRATA

16.1 Tinymovr Alpha CAN Bus Connector Erratum

The CANine v1 Adapter that came with Tinymovr alpha had the DF-13 pins reversed and as such is not compatible with regular DF-13 cables. Alpha users are advised to use the alternative pin header on the board and the included DF-13 to 2.54mm pin converter cable to communicate with Tinymovr, as shown in the diagram below:



Note above that the red wire should stay disconnected and to the left side of the board as viewed from the USB port side.

16.2 Tinymovr Alpha USB Micro Connector Erratum

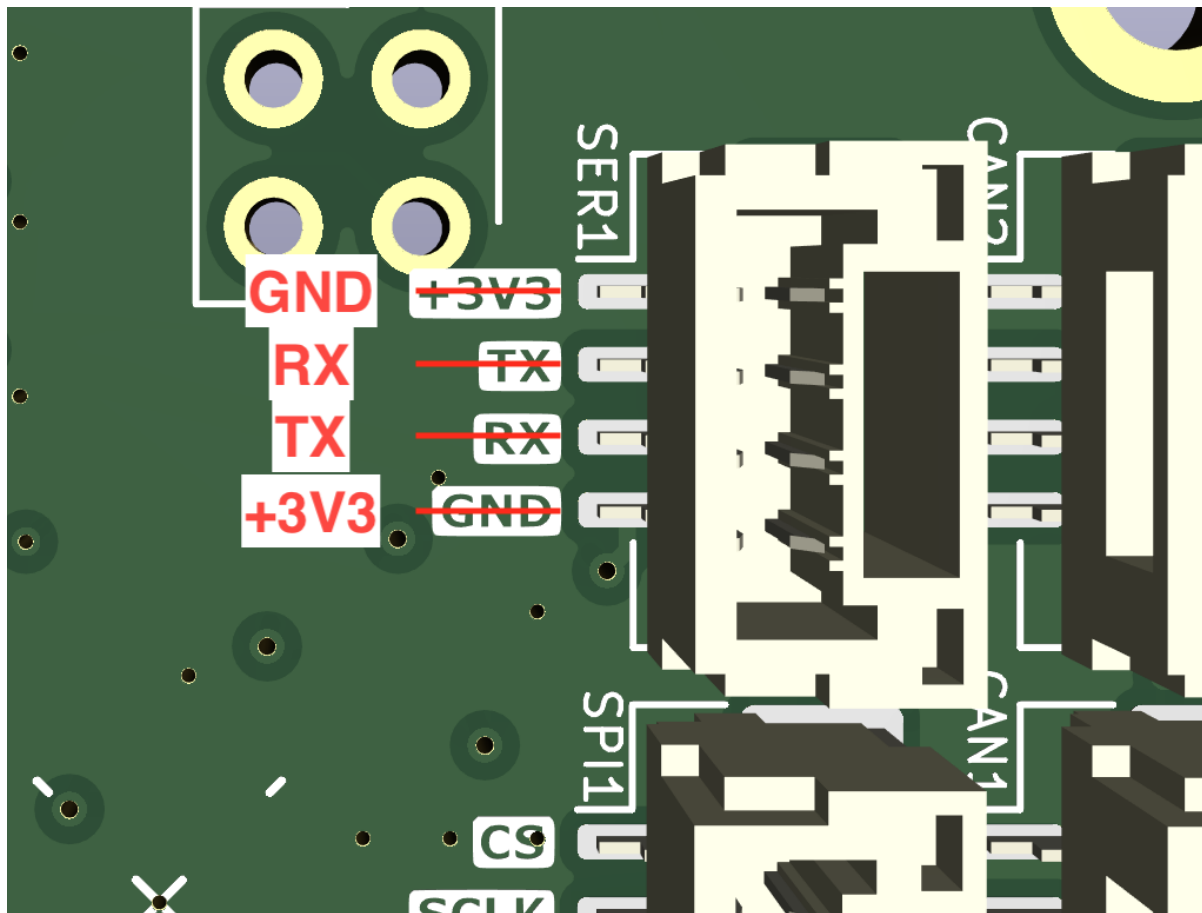
The USB Micro connector used in the v1 adapter was unfortunately not very robust. In order to ensure that there is a good contact between the board and the USB cable, please ensure the male connector of the cable is firmly seated in the female connector of the board.

In addition, avoid exerting lateral forces to the connector (upwards or downwards) as they place stress on the soldered retaining flaps.

Later adapter revisions (aka CANine) use a USB Type C connector and do not have this issue.

16.3 Tinymovr R5 UART Silkscreen Reversed

The silkscreen next to the UART port on the rear of Tinymovr R5 has the order of pins reversed. The correct pins are provided in the figure below.



API REFERENCE

17.1 protocol_hash

ID: 0

Type: uint32

The Avlos protocol hash.

17.2 uid

ID: 1

Type: uint32

The unique device ID, unique to each PAC55xx chip produced.

17.3 fw_version

ID: 2

Type: str

The firmware version.

17.4 hw_revision

ID: 3

Type: uint32

The hardware revision.

17.5 Vbus

ID: 4

Type: float

Units: volt

The measured bus voltage.

17.6 Ibus

ID: 5

Type: float

Units: ampere

The estimated bus current. Only estimates current drawn by motor.

17.7 power

ID: 6

Type: float

Units: watt

The estimated power. Only estimates power drawn by motor.

17.8 temp

ID: 7

Type: float

Units: degree_Celsius

The internal temperature of the PAC55xx MCU.

17.9 calibrated

ID: 8

Type: bool

Whether the system has been calibrated.

17.10 errors

ID: 9

Type: uint8

Any system errors, as a bitmask

Flags:

- UNDERVOLTAGE

17.11 warnings

ID: 10

Type: uint8

Any system warnings, as a bitmask

Flags:

- DRIVER_FAULT
- CHARGE_PUMP_FAULT_STAT
- CHARGE_PUMP_FAULT
- DRV10_DISABLE
- DRV32_DISABLE
- DRV54_DISABLE

17.12 save_config() -> void

ID: 11

Return Type: void

Save configuration to non-volatile memory.

17.13 erase_config() -> void

ID: 12

Return Type: void

Erase the config stored in non-volatile memory and reset the device.

17.14 reset() -> void

ID: 13

Return Type: void

Reset the device.

17.15 enter_dfu() -> void

ID: 14

Return Type: void

Enter DFU mode.

17.16 config_size

ID: 15

Type: uint32

Size (in bytes) of the configuration object.

17.17 scheduler.load

ID: 16

Type: uint32

Processor load in ticks per PWM cycle.

17.18 scheduler.warnings

ID: 17

Type: uint8

Any scheduler warnings, as a bitmask

Flags:

- CONTROL_BLOCK_REENTERED

17.19 controller.state

ID: 18

Type: uint8

The state of the controller.

Options:

- IDLE
- CALIBRATE
- CL_CONTROL

17.20 controller.mode

ID: 19

Type: uint8

The control mode of the controller.

Options:

- CURRENT
- VELOCITY
- POSITION
- TRAJECTORY
- HOMING

17.21 controller.warnings

ID: 20

Type: uint8

Any controller warnings, as a bitmask

Flags:

- VELOCITY_LIMITED
- CURRENT_LIMITED
- MODULATION_LIMITED

17.22 controller.errors

ID: 21

Type: uint8

Any controller errors, as a bitmask

Flags:

- CURRENT_LIMIT_EXCEEDED

17.23 controller.position.setpoint

ID: 22

Type: float

Units: tick

The position setpoint in the user reference frame.

17.24 controller.position.p_gain

ID: 23

Type: float

The proportional gain of the position controller.

17.25 controller.velocity.setpoint

ID: 24

Type: float

Units: tick / second

The velocity setpoint in the user reference frame.

17.26 controller.velocity.limit

ID: 25

Type: float

Units: tick / second

The velocity limit.

17.27 controller.velocity.p_gain

ID: 26

Type: float

The proportional gain of the velocity controller.

17.28 controller.velocity.i_gain

ID: 27

Type: float

The integral gain of the velocity controller.

17.29 controller.velocity.deadband

ID: 28

Type: float

Units: tick

The deadband of the velocity integrator. A region around the position setpoint where the velocity integrator is not updated.

17.30 controller.velocity.increment

ID: 29

Type: float

Max velocity setpoint increment (ramping) rate. Set to 0 to disable.

17.31 controller.current.lq_setpoint

ID: 30

Type: float

Units: ampere

The Iq setpoint in the user reference frame.

17.32 controller.current.Id_setpoint

ID: 31

Type: float

Units: ampere

The Id setpoint in the user reference frame.

17.33 controller.current.lq_limit

ID: 32

Type: float

Units: ampere

The Iq limit.

17.34 controller.current.lq_estimate

ID: 33

Type: float

Units: ampere

The Iq estimate in the user reference frame.

17.35 controller.current.bandwidth

ID: 34

Type: float

Units: hertz

The current controller bandwidth.

17.36 controller.current.lq_p_gain

ID: 35

Type: float

The current controller proportional gain.

17.37 controller.current.max_lbus_regen

ID: 36

Type: float

Units: ampere

The max current allowed to be fed back to the power source before flux braking activates.

17.38 controller.current.max_lbrake

ID: 37

Type: float

Units: ampere

The max current allowed to be dumped to the motor windings during flux braking. Set to zero to deactivate flux braking.

17.39 controller.voltage.Vq_setpoint

ID: 38

Type: float

Units: volt

The Vq setpoint.

17.40 calibrate() -> void

ID: 39

Return Type: void

Calibrate the device.

17.41 idle() -> void

ID: 40

Return Type: void

Set idle mode, disabling the driver.

17.42 position_mode() -> void

ID: 41

Return Type: void

Set position control mode.

17.43 velocity_mode() -> void

ID: 42

Return Type: void

Set velocity control mode.

17.44 current_mode() -> void

ID: 43

Return Type: void

Set current control mode.

17.45 set_pos_vel_setpoints(float pos_setpoint, float vel_setpoint) -> float

ID: 44

Return Type: float

Set the position and velocity setpoints in the user reference frame in one go, and retrieve the position estimate

17.46 comms.can.rate

ID: 45

Type: uint32

The baud rate of the CAN interface.

17.47 comms.can.id

ID: 46

Type: uint32

The ID of the CAN interface.

17.48 comms.can.heartbeat

ID: 47

Type: bool

Toggle sending of heartbeat messages.

17.49 motor.R

ID: 48

Type: float

Units: ohm

The motor Resistance value.

17.50 motor.L

ID: 49

Type: float

Units: henry

The motor Inductance value.

17.51 motor.pole_pairs

ID: 50

Type: uint8

The motor pole pair count.

17.52 motor.type

ID: 51

Type: uint8

The type of the motor. Either high current or gimbal.

Options:

- HIGH_CURRENT
- GIMBAL

17.53 motor.calibrated

ID: 52

Type: bool

Whether the motor has been calibrated.

17.54 motor.I_cal

ID: 53

Type: float

Units: ampere

The calibration current.

17.55 motor.errors

ID: 54

Type: uint8

Any motor/calibration errors, as a bitmask

Flags:

- PHASE_RESISTANCE_OUT_OF_RANGE
- PHASE_INDUCTANCE_OUT_OF_RANGE
- INVALID_POLE_PAIRS

17.56 sensors.user_frame.position_estimate

ID: 55

Type: float

Units: tick

The filtered position estimate in the user reference frame.

17.57 sensors.user_frame.velocity_estimate

ID: 56

Type: float

Units: tick / second

The filtered velocity estimate in the user reference frame.

17.58 sensors.user_frame.offset

ID: 57

Type: float

Units: tick

The user defined offset.

17.59 sensors.user_frame.multiplier

ID: 58

Type: float

The user defined multiplier.

17.60 sensors.setup.onboard.calibrated

ID: 59

Type: bool

Whether the sensor has been calibrated.

17.61 sensors.setup.onboard.errors

ID: 60

Type: uint8

Any sensor errors, as a bitmask

Flags:

- CALIBRATION_FAILED
- READING_UNSTABLE

17.62 sensors.setup.external_spi.type

ID: 61

Type: uint8

The type of the external sensor.

Options:

- MA7XX
- AS5047
- AMT22

17.63 sensors.setup.external_spi.rate

ID: 62

Type: uint8

The rate of the external sensor.

Options:

- 1_5Mbps
- 3Mbps
- 6Mbps
- 8Mbps
- 12Mbps

17.64 sensors.setup.external_spi.calibrated

ID: 63

Type: bool

Whether the sensor has been calibrated.

17.65 sensors.setup.external_spi.errors

ID: 64

Type: uint8

Any sensor errors, as a bitmask

Flags:

- CALIBRATION_FAILED
- READING_UNSTABLE

17.66 sensors.setup.hall.calibrated

ID: 65

Type: bool

Whether the sensor has been calibrated.

17.67 sensors.setup.hall.errors

ID: 66

Type: uint8

Any sensor errors, as a bitmask

Flags:

- CALIBRATION_FAILED
- READING_UNSTABLE

17.68 sensors.select.position_sensor.connection

ID: 67

Type: uint8

The position sensor connection. Either ONBOARD, EXTERNAL_SPI or HALL.

Options:

- ONBOARD

- EXTERNAL_SPI
- HALL

17.69 sensors.select.position_sensor.bandwidth

ID: 68

Type: float

Units: hertz

The position sensor observer bandwidth.

17.70 sensors.select.position_sensor.raw_angle

ID: 69

Type: int32

The raw position sensor angle.

17.71 sensors.select.position_sensor.position_estimate

ID: 70

Type: float

Units: tick

The filtered position estimate in the position sensor reference frame.

17.72 sensors.select.position_sensor.velocity_estimate

ID: 71

Type: float

Units: tick / second

The filtered velocity estimate in the position sensor reference frame.

17.73 sensors.select.commutation_sensor.connection

ID: 72

Type: uint8

The commutation sensor connection. Either ONBOARD, EXTERNAL_SPI or HALL.

Options:

- ONBOARD

- EXTERNAL_SPI
- HALL

17.74 sensors.select.commutation_sensor.bandwidth

ID: 73

Type: float

Units: hertz

The commutation sensor observer bandwidth.

17.75 sensors.select.commutation_sensor.raw_angle

ID: 74

Type: int32

The raw commutation sensor angle.

17.76 sensors.select.commutation_sensor.position_estimate

ID: 75

Type: float

Units: tick

The filtered position estimate in the commutation sensor reference frame.

17.77 sensors.select.commutation_sensor.velocity_estimate

ID: 76

Type: float

Units: tick / second

The filtered velocity estimate in the commutation sensor reference frame.

17.78 traj_planner.max_accel

ID: 77

Type: float

Units: tick / second

The max allowed acceleration of the generated trajectory.

17.79 traj_planner.max_decel

ID: 78

Type: float

Units: tick / second ** 2

The max allowed deceleration of the generated trajectory.

17.80 traj_planner.max_vel

ID: 79

Type: float

Units: tick / second

The max allowed cruise velocity of the generated trajectory.

17.81 traj_planner.t_accel

ID: 80

Type: float

Units: second

In time mode, the acceleration time of the generated trajectory.

17.82 traj_planner.t_decel

ID: 81

Type: float

Units: second

In time mode, the deceleration time of the generated trajectory.

17.83 traj_planner.t_total

ID: 82

Type: float

Units: second

In time mode, the total time of the generated trajectory.

17.84 move_to(float pos_setpoint) -> void

ID: 83

Return Type: void

Move to target position in the user reference frame respecting velocity and acceleration limits.

17.85 move_to_tlimit(float pos_setpoint) -> void

ID: 84

Return Type: void

Move to target position in the user reference frame respecting time limits for each sector.

17.86 traj_planner.errors

ID: 85

Type: uint8

Any errors in the trajectory planner, as a bitmask

Flags:

- INVALID_INPUT
- VCRUISE_OVER_LIMIT

17.87 homing.velocity

ID: 86

Type: float

Units: tick / second

The velocity at which the motor performs homing.

17.88 homing.max_homing_t

ID: 87

Type: float

Units: second

The maximum time the motor is allowed to travel before homing times out and aborts.

17.89 homing.retract_dist

ID: 88

Type: float

Units: tick

The retraction distance the motor travels after the endstop has been found.

17.90 homing.warnings

ID: 89

Type: uint8

Any homing warnings, as a bitmask

Flags:

- HOMING_TIMEOUT

17.91 homing.stall_detect.velocity

ID: 90

Type: float

Units: tick / second

The velocity below which (and together with *stall_detect.delta_pos*) stall detection mode is triggered.

17.92 homing.stall_detect.delta_pos

ID: 91

Type: float

Units: tick

The velocity below which (and together with *stall_detect.delta_pos*) stall detection mode is triggered.

17.93 homing.stall_detect.t

ID: 92

Type: float

Units: second

The time to remain in stall detection mode before the motor is considered stalled.

17.94 home() -> void

ID: 93

Return Type: void

Perform the homing operation.

17.95 watchdog.enabled

ID: 94

Type: bool

Whether the watchdog is enabled or not.

17.96 watchdog.triggered

ID: 95

Type: bool

Whether the watchdog has been triggered or not.

17.97 watchdog.timeout

ID: 96

Type: float

Units: second

The watchdog timeout period.

Symbols

`__init__()` (*BusRouter method*), 28

A

`add_client()` (*BusRouter method*), 28

B

built-in function

`destroy_router()`, 29

`destroy_tee()`, 29

`get_router()`, 29

`get_tee()`, 29

`init_router()`, 29

`init_tee()`, 29

`BusRouter` (*built-in class*), 28

D

`destroy_router()`

 built-in function, 29

`destroy_tee()`

 built-in function, 29

G

`get_router()`

 built-in function, 29

`get_tee()`

 built-in function, 29

I

`init_router()`

 built-in function, 29

`init_tee()`

 built-in function, 29

R

`run()` (*BusRouter method*), 28

S

`send()` (*BusRouter method*), 28

`shutdown()` (*BusRouter method*), 28

`stop()` (*BusRouter method*), 28