Necessary information:

ODrive setup will go more smoothly if you locate this information before you begin.

Motor:

- Nominal voltage [V]
- Pole pairs [integer]
- Speed constant Kv [rpm/V]
- Max speed [rpm]

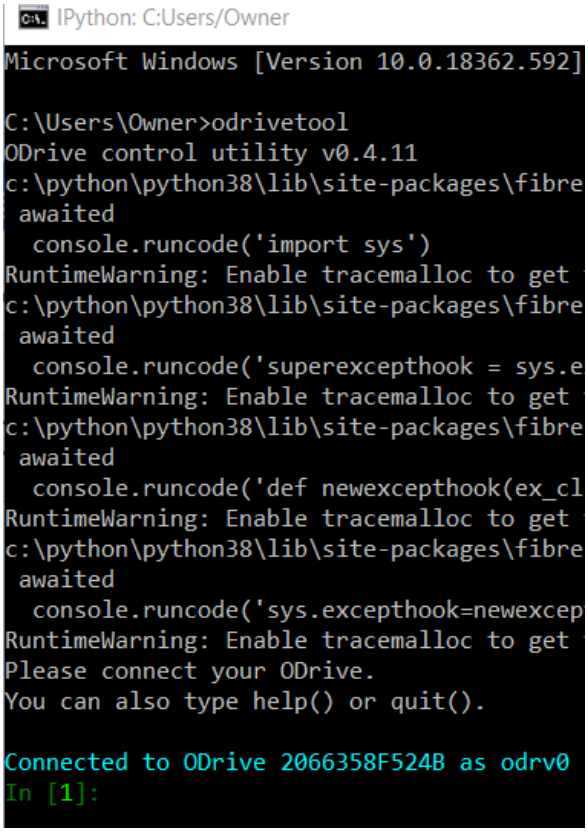
Encoder:

- Counts per rotation [integer] (or counts per turn, or whatever else the datasheet says)
- Max speed [rpm]

See the “Component records” section below to see if anyone has already compiled the information for your motor and/or encoder.

Running and controlling a motor and encoder over USB (Windows)

1. First, download:
    a. ODrivetool: use command prompt to run “pip install odrivetool”.
        i. In Windows 10, this will default to putting it in Users/[username]/AppData/Local/Programs/Python/Python[versionNumber]/Scripts
    b. Zadig utility: download .exe from https://zadig.akeo.ie/. No installation required.
2. Disconnect any motors and encoders from the ODrive. Plug in ODrive USB to PC, then power it up. Note: order matters. Connect first, then power. The ODrive runs from ~10 V to 24 V; choose a voltage appropriate for your motor.
3. The ODrive should be detected on the PC as “ODrive Native Interface 2”. If it doesn’t, you may need to flash its firmware; see here for instructions.
4. Run zadig.exe and select Options>List All Devices. From the dropdown menu, select “Odrive Native Interface 2” (make sure it is “2” and not “0”). Set the target driver to libusb32, and replace/install driver. You only need to do this once, not every time.
    a. Note: you may need to do this separately for different connection options, e.g. once for the USB cable and once for the ST Link.
5. Launch odrivetool in command prompt. If everything is on the path properly, you should be able to simply open the command prompt, type `odrivetool`, and hit enter. After a few seconds, you should see this sequence of text (possibly without the Python warnings) and the blue “Connected to ODrive” notification.



6. Set configuration:
    a. General settings (may differ by motor, but can probably stay the same):
        i. `odrv0.config.brake_resistance = 0.7`
        ii. `odrv0.axis0.motor.config.calibration_current = 2`
            1. If you're getting a "DC bus under voltage" error, increase to 10 A
        iii. `odrv0.axis0.motor.config.current_lim = 60`
        iv. `odrv0.axis0.motor.config.resistance_calib_max_voltage = 2`
            1. We've used both 1 V and 2 V here, and I'm unclear why
    b. Encoder-specific settings: adjust values according to your encoder's datasheet.
        i. `odrv0.axis0.encoder.config.cpr = 32768`
            1. Note: the ODrive uses CPR to mean "counts per revolution", but some manufacturers may use "cycles per revolution", which will be off by a factor of four. Maxon uses "counts per turn", which I would expect to be equivalent to CPR, but (at least in one case) I still had to multiply the "CPT" by four to get a working CPR value.
            2. If you have any problems, you can just try the nominal cpr divided by four, multiplied by four, maybe even divided/multiplied by two...
    c. Motor-specific settings: adjust values according to your motor's datasheet.
        i. `odrv0.axis0.motor.config.motor_type = MOTOR_TYPE_HIGH_CURRENT`
            1. This is the setting for any motor that isn't a gimbal motor
            2. Note that to run MRD lab motor characterization, you *will* need to set this to MOTOR_TYPE_GIMBAL (since that's what allows the firmware to send straight voltage inputs) - just make sure you remember to switch back to MOTOR_TYPE_HIGH_CURRENT once you're done with characterization
        ii. `odrv0.axis0.motor.config.pole_pairs = 2`
            1. Should be on your motor's datasheet, but you can also try finding it yourself using the method described here; search "pole pairs".
        iii. `odrv0.axis0.controller.config.vel_limit = 1638000`
            1. Velocity limit should be chosen per the guidelines here
                a. Choose a value no higher than the smallest of these:
                    i. 0.75*bus_voltage*motorKv
                    ii. 0.8*encoderMaxSpeed
                    iii. 35,000 electrical RPM
                b. A suggested starting limit is something like 3000 RPM
            2. ODrive takes the limit in counts per second, so calculate it using RPM*CPR/60 for whatever limit you want to set
            3. E.g., for this motor with CPR = 32768, vel_limit = 546*RPM; at 3000 RPM, that's vel_limit = 1638000
    d. Save configuration using odrv0.save_configuration()
    e. Reboot using odrv0.reboot()

    i. This is because there's a known bug that causes problems if you save configurations multiple times without power cycling the ODrive. Rebooting immediately after any configuration save ensures you avoid this.
7. Turn off ODrive, connect motor and encoder per the instructions here (search "wiring up the ODrive"), then power ODrive again. This tutorial assumes you are attaching your motor to "axis 0", the set of screw terminals farthest from the USB port; if you want to use the other set of terminals, replace all instances of "axis0" with "axis1".
8. Calibrate motor and encoder. If motor does not calibrate automatically on startup, run:
    a. `odrv0.axis0.requested_state = AXIS_STATE_FULL_CALIBRATION_SEQUENCE`
    b. This includes a few functions that are partially observable
        i. Beep sounds, no movement; this is AXIS_STATE_MOTOR_CALIBRATION, when the ODrive is measuring the motor's phase resistance and phase inductance. If successful, sets `<axis>.motor.is_calibrated` to True.
        ii. Motor makes one rapid, stuttering step; this is AXIS_STATE_ENCODER_INDEX_SEARCH (only does this if `<axis>.encoder.config.use_index` is set to True).
        iii. Motor turns steadily in one direction for a few seconds, then reverses direction and turns for an equal length of time; this is AXIS_STATE_ENCODER_OFFSET_CALIBRATION. If successful, sets `<axis>.encoder.is_ready` to True.
        iv. If calibration isn't working, you can also run these subprocesses individually in order to troubleshoot.
    c. Only when `<axis>.motor.is_calibrated` and `<axis>.encoder.is_ready` are both True can you move on to feedback control.
        i. If you are encountering problems, run dump_errors(odrv0) to attempt to diagnose what's wrong. When you're ready to try again, run dump_errors(odrv0, True) to clear the error state, then run the calibration sequence again.
        ii. You can also try running the motor in sensorless control to see if it works without the encoder; see instructions below.
9. Running motor characterization (only works if you've flashed the MRD Lab version)
    a. Set motor type to gimbal and control mode to current
        i. `odrv0.axis0.motor.config.motor_type = MOTOR_TYPE_GIMBAL`
        ii. `odrv0.axis0.controller.config.control_mode = CTRL_MODE_CURRENT_CONTROL`
    b. Set current limit to the appropriate value for *voltage* limit
        i. When in gimbal motor mode, the ODrive interprets current limit as voltage limit, so choose appropriately
        ii. `odrv0.axis0.motor.config.current_lim = 10`
    c. Set desired input configuration

i. Set input type and parameters (e.g. step voltage, chirp frequency, test input duration) by setting the values of `odrv0.axis0.input_config`
ii. You can see all options by calling `odrv0.axis0.input_config`
iii. E.g. `odrv0.axis0.input_config.step_voltage = 2`
iv. Note that if you save_configuration() and reboot, this will be saved as well

d. Run data collection
i. `run_motor_characterize_input(odrv0, 0, dir)`
ii. The first argument is the ODrive object (odrv0 by default)
iii. The second argument is the axis number, an integer 0 or 1
iv. dir is a string literal specifying the directory where you would like data to be saved
v. The motor will be still for a duration test_delay, then run the test input for a duration test_duration. If running a chirp input (or any high-voltage input), make sure the motor is secured or gripped tightly.
vi. When the input is complete, the data will be saved to the target directory under the name motorCharacterizeData_[date and time].

e. Open data file in MATLAB and analyze
i. A DC motor is assumed to have a transfer function of the form G(s) = b/(s*(s+a)). By plotting the data in MATLAB, you can identify a and b.
ii. You can do this by hand with a step input by identifying the settling time Ts (the time at which the step reaches 98% of its steady-state value SSV) and taking a = 4/Ts and b = a*SSV
iii. Other inputs must be analyzed by finding a vector C = pinv(X)*V where X is composed of motor velocity and acceleration data, for V voltage inputs, where b=1/C(1) and a = C(2)/b, but that's beyond the scope of this project.
iv. Example MATLAB code can be found in the docs/references folder of the lab github

10. Tuning the motor manually
a. Note - generally only stable for small perturbations about the initial setpoint
b. Set initial gains
i. `odrv0.axis0.controller.config.pos_gain = 20.0`
[(counts/s) / counts]
ii. `odrv0.axis0.controller.config.vel_gain = 5.0 / 100000.0`
[A/(counts/s)]
iii. `odrv0.axis0.controller.config.vel_integrator_gain = 0`
[A/((counts/s) * s)]

c. Use Liveplotter to monitor system response to position input:
i. `start_liveplotter(lambda:[odrv0.axis0.encoder.pos_estimate, odrv0.axis0.controller.pos_setpoint])`
ii. `odrv0.axis0.controller.pos_setpoint = 0`
iii. Update: as of v0.5.0, you instead use `odrv0.axis0.controller.input_pos = 0`

d. Put in position control mode and implement closed loop control
    i. `odrv0.axis0.controller.config.control_mode = CTRL_MODE_POSITION_CONTROL`
    ii. `odrv0.axis0.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL`
    iii. It should now be actively maintaining its position, pushing back if you try to change it. It will probably be vibrating, though, which means you need to tune the gains.
    iv. Common error: if it takes off too fast and triggers a controller overspeed error, reduce the velocity gain.
e. To stop motor while you adjust gains,
    i. `odrv0.axis0.requested_state = AXIS_STATE_IDLE`
f. Follow tuning methodology here (scroll to bottom of page)
    i. For the Maxon motor(s), start with very low gains: position <15, velocity <5/100000, vel_integrator=0

11. Optional: running in velocity control mode
    a. This can be helpful for troubleshooting. Velocity control mode is still closed loop control, but you command a velocity setpoint instead of a position setpoint.
    b. `odrv0.axis0.controller.config.control_mode = CTRL_MODE_VELOCITY_CONTROL`
    c. `odrv0.axis0.controller.vel_setpoint = 5000 #in counts/s`
        i. Make sure the velocity limit is at least 10% higher than the setpoint to allow for some error.
        ii. Note that this is in [counts/s]; if your encoder has 8192 CPR, 5000 counts/s is less than 1 RPM. Adjust accordingly for a reasonable speed; 10-200 rpm is an easy, low-stakes range.
    d. `odrv0.axis0.requested_state=AXIS_STATE_CLOSED_LOOP_CONTROL`
        i. Common error: if it takes off too fast and triggers a controller overspeed error, reduce the velocity gain.
    e. To stop motor while you adjust gains,
        i. `odrv0.axis0.controller.config.control_mode = AXIS_STATE_IDLE`

Useful diagnostics and visualizations

- Liveplotter - produces a regularly-updating plot of the specified value over time
  - `start_liveplotter(lambda:odrv0.axis0.encoder.pos_estimate)`
  - Replace `odrv0.axis0.encoder.pos_estimate` with desired variable; you can also plot multiple values by enclosing them in brackets, e.g. `start_liveplotter(lambda:[odrv0.axis0.encoder.pos_estimate, odrv0.axis0.controller.pos_setpoint])`
  - Commonly useful options include
    - Encoder position estimate: `odrv0.axis0.encoder.pos_estimate`
    - Position setpoint: `odrv0.axis0.controller.pos_setpoint`
    - Current draw: `odrv0.axis0.motor.current_control.Iq_measured`
- Dump_errors - returns a list of the error status of each axis and its components
  - `dump_errors(odrv0)`
  - The ODrive will generally disable the motor if there are any outstanding errors. If you've changed something and want to try again, you'll have to clear the error state in order to re-enable the motor: `dump_errors(odrv0, True)`

Component records

Motor and sensor: Maxon 605064

Motor: Maxon 323218

EC-4pole 22 mm diameter, brushless, 90 Watt

Nominal voltage: 24 V
Nominal max continuous torque: 45.1 mNm
Nominal max continuous current: 3.34 A
Terminal resistance phase to phase: 0.527 ohms
Terminal inductance phase to phase: 0.0503 mH
Max speed: 25,000 rpm
Pole pairs: 2
Torque constant: 14 mNm/A
Speed constant Kv: 680 rpm/V

Encoder: Maxon 575828

Encoder ENC 16 RIO, 8192 counts per turn, 3-channel, RS 422 line driver

CPT = 8192; I thought this would be equivalent to CPR, but I discovered that I do have to multiply it by four; see here for discussion of the confusing conventions for this
Max speed = 20,000 rpm

Tuning notes:
Starts okay with position gain = 10, velocity gain = 5.0 / 1000000.0, velocity integrator gain = 0, but with slight vibration that worsens over time.
Dropped to pos=5, vel=2.5/1000000.0
Here the vibration is much better, but the system goes unstable after about ten seconds.
Try vel=1.0/1000000.0. Unstable.
Try pos = 1. Works with vibration for a few seconds, then goes unstable.
Try pos = 1.0 / 10.0. Same thing...try all gains zero? Still the same.

Something else to try?
If you are using a high resolution encoder (>4000 counts/rotation) then increasing encoder_pll_bandwidth may help reduce vibration
Good info here https://discourse.odriverobotics.com/t/rotor-encoder-pll-and-velocity/224

For easy reference:

```
odrv0.reboot()
odrv0.axis0.requested_state = AXIS_STATE_FULL_CALIBRATION_SEQUENCE
odrv0.axis0.controller.config.pos_gain = 10.0
odrv0.axis0.controller.config.vel_gain = 5.0 / 1000000.0
odrv0.axis0.controller.config.vel_integrator_gain = 0

odrv0.axis0.controller.pos_setpoint = 0
```

```
odrv0.axis0.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
odrv0.axis0.controller.config.control_mode = AXIS_STATE_IDLE
dump_errors(odrv0,True)
```

Common problems

See here for more common errors, and links to the source code for all errors.

- ERROR_CPR_OUT_OF_RANGE (encoder)
  - This probably means you haven't configured the encoder CPR properly - which is very easy to do given the lack of standardization for CPR/PPR/etc values.
  - Try changing the CPR setting to either ¼ or 4x the current value.
    - `odrv0.axis0.encoder.config.cpr = 2048`
- ERROR_DC_BUS_UNDER_VOLTAGE (axis)
  - If this happened when you tried to calibrate the motor, your power supply may be unable to supply enough current.
  - You can check for voltage spikes by calling liveplotter on odrv0.vbus_voltage to see exactly when they happen.
    - `start_liveplotter(lambda: [odrv0.vbus_voltage])`
  - Check the current limit of your supply; if it seems low, try using a more powerful supply.
- ERROR_OVERSPEED (controller)
  - Option one: your velocity gain is set too high.
    - This is likely the case if the motor seems to be taking off faster than it should.
  - Option two: you need to increase your velocity limit and/or velocity limit tolerance.
    - This is more likely the case if the motor is moving as you expect it to, but then throwing this error when moving between widely-spaced positions.
    - NOTE: since vel_limit is in [counts/s], it depends on encoder CPR, so this is also more likely if you're using a high-CPR encoder.
      - `odrv0.axis0.controller.config.vel_limit = 30000`
    - You can also try changing the velocity limit tolerance; see the ODrive Troubleshooting page for details

Running a motor sensorless

1. Configure as per instructions here:
    a. Standard parameters:
        i. `odrv0.axis0.controller.config.vel_gain = 0.01`
        ii. `odrv0.axis0.controller.config.vel_integrator_gain = 0.05`
        iii. `odrv0.axis0.controller.config.control_mode = 2`
            1. Mode “2” is velocity control mode
        iv. `odrv0.axis0.controller.vel_setpoint = 400`
            1. Chooses motor speed in rad/s
        v. `odrv0.axis0.motor.config.direction = 1`
            1. Can be changed if desired
    b. Motor-specific parameters:
        i. `odrv0.axis0.sensorless_estimator.config.pm_flux_linkage = 7.16e4`
        ii. Note: this value should be set to `5.51328895422 / (<pole pairs> * <motor kv>)`
2. Call sensorless control
    a. `odrv0.axis0.requested_state=AXIS_STATE_SENSORLESS_CONTROL`