Electric Wheelbarrow

Max-Felix Müller

2020
July

Contents

List of Figures

1 Motivation

In the summertime I have to get my stuff to the public swimming pool. Riding my bicycle is ok, but when the weather is really hot, I do not want to be sweaty on the way home already. One solution would of course be an E-bike but that is too mainstream. Instead I decided to convert a wheelbarrow with an electric hub motor I got for free. The motor is driven by an O-Drive and powered using an old Li-Ion battery from a grass trimmer.

2 Mechanical Assembly

I found a really cheap wheelbarrow in a local hardware store. It only required re-drilling some holes to make room for the thicker motor instead of the wheel. Some 3d printed adapters are used to fit the motor on he original wheel supports. I mounted the O-Drive and the battery holder at the back of the wheelbarrow close to the handles.

3 Getting Started With The O-Drive

The O-Drive is connected to a PC with Linux installed (because that is easier than Windows in such cases).

First I installed the odrivetool which is a Python program used to communicate with and configure the O-Drive from a PC.

```
pip3 install odrive
```

To check and update to the newest firmware, simply call the dfu (device firmware update). Note that the device must be powered, not just via USB but also from the battery to connect with the PC. My O-Drive is rated for up to 56V but that can vary. The battery I use is rated at only 36V but charges up to 42V. Keep that in mind when using Lithium based batteries.

```
odrivetool dfu
```

Next I backed up the previous configuration (it is not overwritten with the firmware update). I think I did not change much but it is the better way in case I break something.

```
# backup:
odrivetool backup-config config.json

# to restore use:
odrivetool restore-config config.json
```

3.1 Hard-to-find Errors

After some searching around I found the solution to some of the errors I encountered. I think it might be helpful to have them collected here:

ERROR_CURRENT_UNSTABLE
This is some tolerance problem when running sensorless it seems.

```
I managed to get around this error message by increasing the tolerance.
It might also be, that the problem is the stall current at motor startup.
Changing this value might introduce uneven motor speeds,
but I did not check in the firmware.
odrv0.axis0.motor.config.current_lim_tolerance = 5.0
```

```
ERROR_PHASE_RESISTANCE_OUT_OF_RANGE
An error resulting from a bad calibration.
After I increased the calibration current I also had to increase the
maximum calibration voltage.
odrv0.axis0.motor.config.calibration_current = 20.0
odrv0.axis0.motor.config.resistance_calib_max_voltage = 5.0
```

```
ERROR_DC_BUS_UNDER_VOLTAGE
This is triggered when the input voltage falls under a given threshold.
For me the limit is set to 8V. With a 40V battery I thought it could not
get that low but the
diagram showed some kind of spike that would, for a very brief moment,
drop the battery voltage
under that limit.
I added a huge capacitor to fix the problem.
The voltage can be plotted using this.
start_liveplotter(lambda: [odrv0.vbus_voltage])
```

Whenever an error occurs it can be read and cleared using the following command. By leaving "True" out, the error is shown but not cleared. While there is an un-cleared error the motor will not start.

```
dump_errors(odrv0, True)
```

3.2 Basic Setup For Sensorless Mode

Sensorless mode means, that there is no sensor giving feedback to the motor driver about the motor speed or direction. It is not exactly an open loop controller either. Instead the back-EMF is used to estimate the motors state. Back-EMF is the votage created by the magnets in the motor changing the magnetic field through the motor coils. This works fine at higher speeds but can result in inaccurate control at low speeds where the signal strength is very low.

Since there is no accurate motor feedback I do strongly recommend AGAINST using this setup for CNC or 3D printing. Instead of distance control I only use velocity control.

I just found the documentation from my previous attempt so I will combine both the old findings and the new ones to get the best result.

```
odrv0.axis0.controller.config.control_mode = 2
odrv0.axis0.requested_state = AXIS_STATE_MOTOR_CALIBRATION
odrv0.axis0.motor.config.direction=1

# start the motor
odrv0.axis0.requested_state = AXIS_STATE_SENSORLESS_CONTROL
```

```
# change motor speed (e.g. 500 RPM)
odrv0.axis0.controller.vel_setpoint = 500

# stop the motor, either 0 velocity or idle state
odrv0.axis0.controller.vel_setpoint = 0
odrv0.axis0.requested_state = AXIS_STATE_IDLE
```

Do not forget to save your configuration after any changes were made...

```
odrv0.save_configuration()
```

This saves the settings to the EEPROM so that they are not lost when the battery is disconnected.

3.3 Starting The Motor At Power Up

Next step is to have the motor running (or at least armed but the velocity set to 0) right after power is applied. In the examples below I set the velocity at power up to 400 rpm. If that does not suit your application, feel free to change the values.

The O-Drive offers the ability to start up as soon as power is applied. A control mode has to be set and the motor needs to be pre-calibrated to do so.

```
# start into sensorless mode
odrv0.axis0.config.startup_sensorless_control = True

# set the acceleration to start up with
odrv0.axis0.config.sensorless_ramp.accel = 200

# set the targeted velocity
odrv0.axis0.config.sensorless_ramp.vel = 400

# set finish when reaching the target velocity
# disale the other finishing conditions
odrv0.axis0.config.sensorless_ramp.finish_on_vel = True
odrv0.axis0.config.sensorless_ramp.finish_on_distance = False
odrv0.axis0.config.sensorless_ramp.finish_on_enc_idx = False
```

4 Arduino Code To Set The Velocity

I use a rotary encoder that's read by an Arduino to set the target velocity. The limits can be set in the O-Drive. I use the code form my previous post about reading an encoder to get clean input signals.

The code translates the velocity set by the encoder to the ASCII format understood by the O-Drive. Since I did not include the library but coded the translation myself that is the only thing that it currently does.

The Arduino is then simply connected to the GPIO pins 1 and 2 of the O-Drive, which is TX and RX for a serial UART port.

5 Tests

Testing went fine. One wheel broke since it stood too long and the rubber was too old. Another wheel broke due to overloading.

Motor-speed and -control was working as intended.

6 E-Bike Handle To Control Speed

I bought a cheap E-Bike handle with an integrated hall sensor. The Arduino is able to read the analog voltage coming from that sensor and convert it into a speed value to send to the O-Drive.

There were some difficulties since the diameter of the handle did not fit the original handles of the wheelbarrow. I used a round piece of wood that was screwed into the original and the new handle.

The hall sensor does not read zero when it is in neutral position. I modified the Arduino code to take the value of the very first reading and use it as a baseline for all further readings. I also added a dead zone in case the value drifts slightly.

7 Pictures

Figure 1: Handle from an E-Bike to control the speed

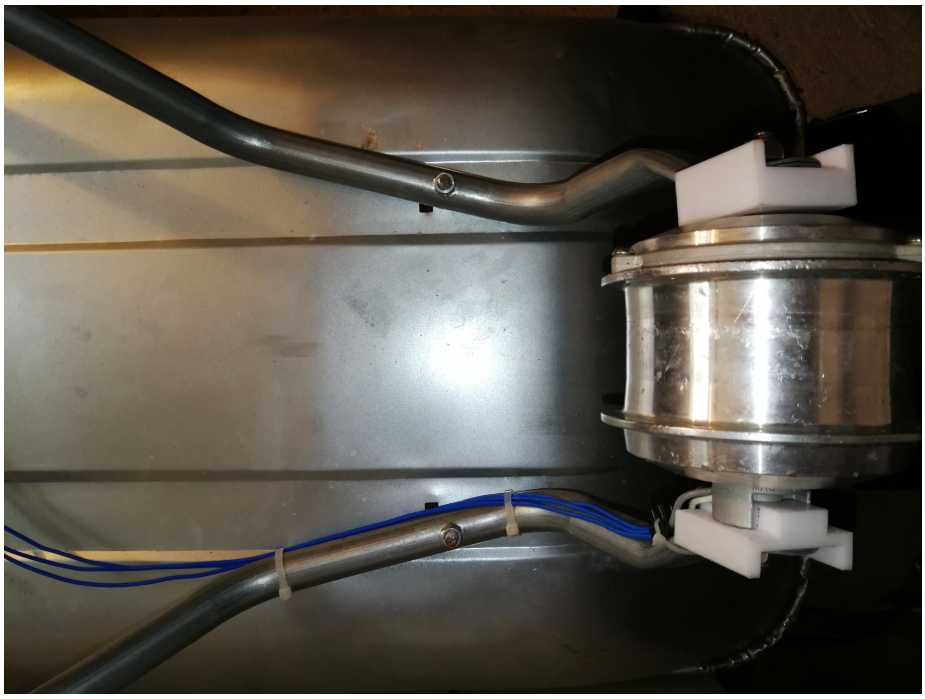

Figure 2: The hub motor and some cable management

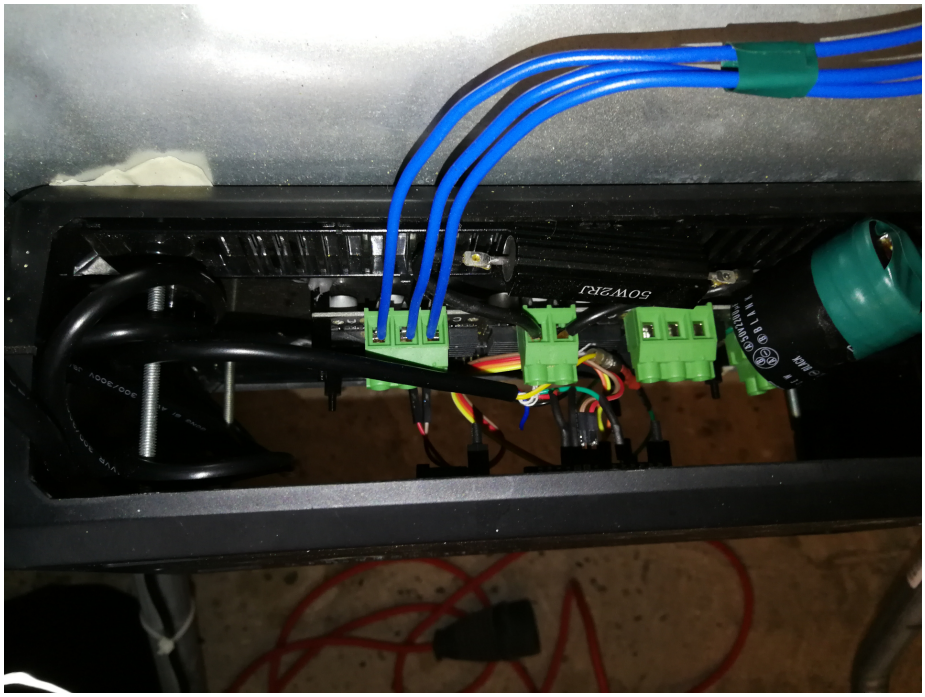

Figure 3: The main electronics, hidden inside an enclosure

8 Links

Code on GitHub