Monitoring and alerting | Filament sensing with KY-040 Rotary Encoder

> The encoder is not mounted at the moment because hardware failed (but software works well)

Targets

The rotary encoder can be used to monitor the direction and speed of filament feeding or retracting (mm/s), or at least the amount of moved filament (mm). If it does not give rotational feedback while Trikarus is printing that will be useful information. The

information can be used to trigger events like pausing the print job or similar:

- the hotend is not warm enough to extrude
- the filament is slipping or grinding at the gears
- filament is just empty
- filament diameter is too thick → clogged
- something else happened that prevents rotating the axis of the encoder. It might also be an encoder problem itself. The movement of the encoder axis is done by a small rubber cylinder. That rubber part might get rubbish after some kilometers of movement due to wear. It seems that the used rubber gets more softly when moving all the time. Thus leads to raw filament string slipping.

The filament sensor can also be used to analyse how much filament per second the extruder can push through the nozzle at a defined temperature and steps/mm. If you try to punch the filament through the nozzle with exceeding speeds, the encoder will not rotate like theoretically exspected. The deviation between real and desired value can be easily measured with this sensing tool. So it was done. It figured out that Super Volcano cannot push filament with 60 mm/s through nozzle at heating temperature of 240 °C. A good upper limit is around 30 mm/s. You can read about here too: Printer profiles, slicing and filaments

Encoders are imperfect, there will be a position error that we just have to expect and accept. With a constant Δt approach, this translates into a constant worst-case speed error, independent of speed. The choice of Δt:

- If we choose a large Δt, we get a low bandwidth and large time lag, but we get very low speed error
- If we choose a small Δt, we get very high bandwidth and small time lag, but we get high speed estimation error

> The rotary encoder is also used to switch the Smart Stepper modes. It's button is wired up and monitored by a python detection script which will change the modes accordingly. See Smart Stepper - Overview for more information on how to do this.

Setting up

Based on https://pypi.org/project/pigpio-encoder

>

> The switch (knob function) of the KY040 Encoder is wired up too, but it is not used. IT could be implemented to perform some gimmick like triggering an action if some specific trigger pattern or behaviour was recognized - for example (e.g. pressing 3 times could quickly put Smart Stepper into torque mode or back to normal operation).

To monitor slow filament feeding, the debouncing has to be really low. It was tested with default values which did not trigger properly. I ended up using 10 ms as a good woring value for feed speeds from 1 mm/s up to to 30 mm/s. The monitoring happens in different parts. At first a Python service script is created and running all the time to monitor filament movement. It will calculate data like rounds per minute, speed and more. The data will be written to journald (systemd unit). A second python script will grab that data by hooking up the journald output stream

```
apt install python3-pip
pip3.7 install pigpio
pip3.7 install influxdb
pip3.7 install systemd
```

```
vim /opt/gpio/rotaryEncoder.py
```

```
import pigpio
from datetime import *
from time import sleep
import time, math
import logging
from systemd.journal import JournaldLogHandler
from influxdb import InfluxDBClient
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

logger = logging.getLogger('rotaryEncoder')
logger.addHandler(JournaldLogHandler())
logger.setLevel(logging.DEBUG)

sequence = []
sequence_up = ['dt1', 'clk1', 'dt0', 'clk0']
sequence_down = ['clk1', 'dt1', 'clk0', 'dt0']
```

```
debounce = 10
clk = 12
dt = 6
clk_input = pigpio.pi()
dt_input = pigpio.pi()
clk_input.set_glitch_filter(clk, debounce)
dt_input.set_glitch_filter(dt, debounce)

# process params
dist_meas_mm = 0.00
olddist_meas_mm = 0.00  # last loop
mm_per_s = 0
rpm = 0
elapse = 0
pulse = 0
gust = 0  # last gust
avg = 0  # last average

sleeptime = 0.1  # secs between reporting loop
gustint = 3  # secs to calc gust (>sleeptime)
avgint = 5  # secs to trigger average calc (>gustint)
secsnoread = 6  # number of seconds rotor is stationary before a 'no read' is declared and set
result to zero
loopcount = 0  # a'nothing is happening' counter

d_eff = 11.5  # mm effective diameter of the rubber wheel where filament strives over
lmove = math.pi * d_eff / 12  # mm circumference which are rotated at the encoder by one tick

start_timer = time.time()  # for interrupt function
gust_timer = time.time()  # start of this gust timing
gustm_start = dist_meas_mm  # start of this gust distance
avg_timer = time.time()  # start of this average timing
avgm_start = dist_meas_mm  # start of average distance

# build some database connection
client = InfluxDBClient(host='localhost', port=8086, username='username', password='password',
ssl=False, verify_ssl=False)
#client.ping()
client.switch_database('trikarus')
```

```
#print(client.query('SHOW DATABASES'))
#print(client.query('SHOW MEASUREMENTS'))

def clk_fall(gpio, level, tick):
    if len(sequence) > 2:
        sequence.clear()
    sequence.append('clk1')


# interrupt
def clk_rise(gpio, level, tick):
    sequence.append('clk0')
    if sequence == sequence_up:
        sequence.clear()
        calculate_elapse(1)


def dt_fall(gpio, level, tick):
    if len(sequence) > 2:
        sequence.clear()
    sequence.append('dt1')


# interrupt
def dt_rise(gpio, level, tick):
    sequence.append('dt0')
    if sequence == sequence_down:
        sequence.clear()
        calculate_elapse(-1)


# callback function
def calculate_elapse(increment):
    global pulse, start_timer, elapse
    pulse += increment  # increase/decrease pulse whenever interrupt occurred
    elapse = time.time() - start_timer  # elapsed time
    start_timer = time.time()  # let current time equals to start_timer
```

```
def calculate_speed():
    global rpm, olddist_meas_mm, dist_meas_mm, mm_per_s
    try:
        rpm = 1 / elapse * 60
        mm_per_s = lmove / elapse
        dist_meas_mm = lmove * pulse  # measure distance traverse
        if dist_meas_mm == olddist_meas_mm:
            mm_per_s = 0
            rpm = 0
        if dist_meas_mm < olddist_meas_mm: # this indicates the switch between cw and ccw
rotation
           mm_per_s = -1 * mm_per_s
        return mm_per_s
    except ZeroDivisionError:
        pass

# gust is synonym for onrush
def calcgust():
    global gust_timer, gustm_start, avg_timer, avgm_start, gust, avg

    gustime = time.time() - gust_timer  # how long since start of gust check?
    if gustime >= gustint:  # then calc average speed over gust time
        gustmm = (dist_meas_mm - gustm_start)  # how far since start of gust check
        thisgust = gustmm / gustime
        # print('gust', gustime, gustmm, thisgust, gust)
        if thisgust > gust:
            gust = thisgust
        gust_timer = time.time()  # reset
        gustm_start = dist_meas_mm

    avgtime = time.time() - avg_timer  # how long since start of avg check?
    if avgtime >= avgint:  # then calc average speed over avg time
        avgmm = (dist_meas_mm - avgm_start)  # how far since start of avgcheck
        thisavg = avgmm / avgtime
        # print('avg', avgtime, avgint, avgmm, thisavg)
        avg = thisavg
        avg_timer = time.time()  # reset
        avgm_start = dist_meas_mm
        gust_timer = time.time()
```

```
        gustm_start = dist_meas_mm
        if avg != 0:
            report('average')
        gust = 0  # reset max gust over average duration as well
        avg = 0  # and reset avg in case of calm


def report(mode):
    if mode == 'realtime':
        json_body = [
                 {
                    "measurement": "rotary_encoder",
                    "tags": {
                         "host": "hangdevice.fablabchemnitz.de"
                     },
                    "time": datetime.utcnow().strftime('%Y-%m-%dT%H:%M:%SZ'),
                    "fields": {
                        "rpm": float(rpm),
                        "mm_per_s": float(mm_per_s),
                        "dist_meas_mm": float(dist_meas_mm),
                        "pulse": float(pulse),
                        "elapse": float(elapse * 1000.0) #report them as milliseconds to the
InfluxDB
                    }
                },
            ]
        client.write_points(json_body)
        logger.info("rpm:{0:.0f} mm_per_s:{1:.1f} dist_meas_mm:{2:.2f} pulse:{3}
elapse:{4}".format(rpm, mm_per_s, dist_meas_mm, pulse, elapse))
    elif mode == 'average':
        json_body = [
                        {
                        "measurement": "rotary_encoder",
                        "tags": {
                            "host": "hangdevice.fablabchemnitz.de"
                        },
                        "time": datetime.utcnow().strftime('%Y-%m-%dT%H:%M:%SZ'),
                        "fields": {
                            "gust": float(gust),
```

```
                            "avg": float(avg)
                        }
                    },
                ]
        client.write_points(json_body)
        logger.info("{0:.1f} Gust (mm/s), {1:.1f} Average (mm/s)".format(gust, avg))
    elif mode == 'error':
        logger.info("dead calm or connection fault")
    else:
        logger.warning('bad report mode')


if __name__ == '__main__':
    clk_falling = clk_input.callback(clk, pigpio.FALLING_EDGE, clk_fall)
    clk_rising = clk_input.callback(clk, pigpio.RISING_EDGE, clk_rise)
    dt_falling = dt_input.callback(dt, pigpio.FALLING_EDGE, dt_fall)
    dt_rising = dt_input.callback(dt, pigpio.RISING_EDGE, dt_rise)

    while True:
        olddist_meas_mm = dist_meas_mm
        calculate_speed()
        calcgust()
        if olddist_meas_mm != dist_meas_mm:
            loopcount = 0
            report('realtime')
        else:
            loopcount += 1
            if loopcount == secsnoread / sleeptime:  # its stopped
                report('realtime')
            if loopcount == 20 / sleeptime:  # if secsnoread is reached report error
                loopcount = secsnoread / sleeptime + 1  # reset loopcount
                #report('error')
                report('realtime') #always report realtime instead of error to have better
output for InfluxDB
                json_body = [
                                {
                                "measurement": "rotary_encoder",
                                "tags": {
                                    "host": "hangdevice.fablabchemnitz.de"
```

```
                                },
                                "time": datetime.utcnow().strftime('%Y-%m-%dT%H:%M:%SZ'),
                                "fields": {
                                    "gust": float(0),
                                    "avg": float(0)
                                }
                            },
                        ]
                client.write_points(json_body)

        sleep(sleeptime)
```

Create and enable pigpiod daemon (required to run pigpio things)

```
apt install pigpiod
```

```
vim /etc/systemd/system/pigpiod.service
```

```
[Unit]
Description=Pigpio daemon

[Service]
Type=forking
#disallow port 8888 from outside
ExecStart=/usr/bin/pigpiod -l

[Install]
WantedBy=multi-user.target
```

```
systemctl enable pigpiod.service
systemctl start pigpiod.service
service pigpiod restart
```

Run the encoder script for testing

```
python3.7 /opt/gpio/rotaryEncoder.py #do not use Python2 because it will fail
```

Install encoder script as service

> The service will automatically restart after 10 seconds in cause of failure, e.g. if InfluxDB is down or host not reachable. This will trigger things like refused connection. The data flows into separately created InfluxDB database.

```
vim /opt/gpio/rotaryEncoder.service
```

```
[Unit]
After=network.target
Description=Rotary Encoder Service

[Service]
Type=simple
ExecStart=/usr/bin/python3.7 /opt/gpio/rotaryEncoder.py
KillMode=process
Restart=on-failure
RestartSec=10
RemainAfterExit=no
User=root
Group=root

[Install]
WantedBy= multi-user.target
```

```
systemctl enable /opt/gpio/rotaryEncoder.service
systemctl daemon-reload && service rotaryEncoder restart && journalctl -f -u
rotaryEncoder.service
```

Physical Measurements

To write the monitoring script and to check against it's good practice to do a lot of measurements.

Notes:

- KY-040: one full rotation is expressed by 12 pulses (low resolution encoder). So $n_f = 12$. See Encoder KY-040 by Keyes - filament monitor

- the effective diameter d_{eff} of the rubber cylinder (filament feed) is ~11.5 mm
- values are sorted by count increments

$l_{move} = \pi \cdot \ d_{eff} \cdot \frac{c_{inc}}{n_{f}}$

	test results for 50 mm extrusion with 15 mm/s @ 240°C				test results for 100 mm extrusion with 15 mm/s @ 240°C				test results for 500 mm extrusion with 30 mm/s @ 240°C		
											
	15	1,25	45,16 mm		27	2,25	81,29 mm		14	1,17	42,15 mm
	15	1,25	45,16 mm		28	2,33	84,30 mm		14	1,17	42,15 mm
	16	1,33	48,17 mm		28	2,33	84,30 mm		15	1,25	45,16 mm
	16	1,33	48,17 mm		29	2,42	87,31 mm		16	1,33	48,17 mm
	16	1,33	48,17 mm		31	2,58	93,33 mm		16	1,33	48,17 mm
	17	1,42	51,18 mm		32	2,67	96,34 mm		17	1,42	51,18 mm
	17	1,42	51,18 mm		32	2,67	96,34 mm		18	1,50	54,19 mm

	17	1,42	51,18 mm		33	2,75	99,35 mm		18	1,50	54,19 mm
	18	1,50	54,19 mm		34	2,83	102,36 mm		19	1,58	57,20 mm
					34	2,83	102,36 mm		21	1,75	63,22 mm
					35	2,92	105,37 mm		21	1,75	63,22 mm
					35	2,92	105,37 mm		22	1,83	66,24 mm
					36	3,00	108,38 mm				
					39	3,25	117,42 mm				

Stress test macro on Duet Web Control / Repetier Server

You can use the following macro to test the wear of the rubber cylinder of the filament feed sensor. The macro will quickly move the filament up and down. Measure the filament at some defined point (e.g. 2 cm above entry of feed sensor) and check if the same measurement can be taken after running it. In parallel you can check the encoder values. If everything runs fine you start with zero and end with zero (or near zero).

```
;heat up the nozzle to 205 °C and wait for it
G10 P0 R140 S205
T0
M116

;50 times extrude and retract 5 mm with 60 mm/s
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
```

```
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600

G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
```

```
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600

G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
```

```
G1 E-5 F3600
G1 E+5 F3600

G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
G1 E-5 F3600
G1 E+5 F3600
```

Validation of single extrusion command against output of rotary encoder script

```
G1 E+25 F900 ; = 15mm/s
```

```
python3 /opt/gpio/rotaryEncoder_recent.py
Mon Apr 20 12:33:22 2020 rpm:0 mm_per_s:0.0 dist_meas_mm:0.00 pulse:0 elapse:0
Mon Apr 20 12:33:26 2020 rpm:6 mm_per_s:0.3 dist_meas_mm:3.01 pulse:1
elapse:10.136061668395996
Mon Apr 20 12:33:26 2020 rpm:300 mm_per_s:15.1 dist_meas_mm:6.02 pulse:2
elapse:0.19987750053405762
Mon Apr 20 12:33:26 2020 rpm:347 mm_per_s:17.4 dist_meas_mm:9.03 pulse:3
elapse:0.17283368110656738
Mon Apr 20 12:33:27 2020 rpm:315 mm_per_s:15.8 dist_meas_mm:12.04 pulse:4
```

```
elapse:0.19038128852844238
Mon Apr 20 12:33:27 2020 rpm:117 mm_per_s:5.9 dist_meas_mm:15.05 pulse:5
elapse:0.511589527130127
Mon Apr 20 12:33:27 2020 rpm:300 mm_per_s:15.1 dist_meas_mm:18.06 pulse:6
elapse:0.19991707801818848
Mon Apr 20 12:33:28 2020 rpm:308 mm_per_s:15.5 dist_meas_mm:21.07 pulse:7
elapse:0.19481205940246582
Mon Apr 20 12:33:31 2020 7.0 Gust (mm/s), 4.2 Average (mm/s)
Mon Apr 20 12:33:34 2020 rpm:0 mm_per_s:0.0 dist_meas_mm:21.07 pulse:7
elapse:0.19481205940246582
```

Dropping old values

```
use trikarus
drop series from rotary_encoder
show series
show measurements
```

Advanced script

The rotaryEncoder.py could be developed with some advanced code to store and restore values like pulse (position) to do accumulative or continuous reading which continues after restarting the service or whole Raspberry Pi. But due the fact the the feed sensor is not that exact, storing these values won't be that important to do so. However, saving previous values into json object seems to be straight forward and easy:

https://stackabuse.com/reading-and-writing-json-to-a-file-in-python. Additionally the script can be used to trigger actions by using Repetier Server API. It easily be used to perform pause or stop commands or to just print some information to print console or display. To keep the rotaryEncoder.py speedy just let the python script write it's values to journald like before but connect another script to read out the values again from journald. Or you can add some asynchronous procedures (e.h. nohup) to the rotaryEncoder.py script to make it work and keep it quick.

Version #1
Erstellt: 2026-06-08 15:33:15 CEST von Mario Voigt
Zuletzt aktualisiert: 2026-06-08 15:36:55 CEST von Mario Voigt