Indoor Cycling: Leistungsmessung an einem Spinning-Bike nachrüsten

Einleitung

Die in den 80er Jahren aufgekommenen Spinning-Bikes erfreuen sich als Trainingsgerät immer noch großer Beliebtheit. Der Sport wird zwar meistens in Form von Gruppenkursen von Fitness-Centern angeboten. Ein immer noch lebhafter Gebrauchtmarkt deutet jedoch darauf hin, dass die Bikes auch zuhause als Trainingsgerät genutzt werden und somit auch für eigene Bastelprojekte zugänglich sind.

Alle Spinning-Bikes haben eine Einstellung für den Fahrwiderstand. Zumindest bei älteren Modellen ist das Schwungrad mit Bremsbacken versehen, die mit einem Handrad verstellt werden.

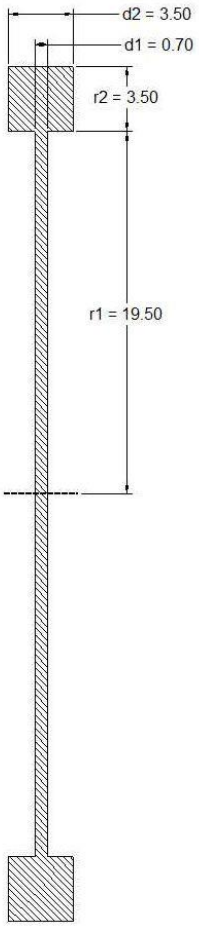
Mit dem Handrad wird der Widerstand über mehrere Umdrehungen nach subjektivem Empfinden eingestellt. Es gibt jedoch oft weder eine Skala noch eine Anzeige für den eingestellten Widerstand. Die momentane Leistung in Watt kann meist nicht ermittelt werden. Somit ist es kaum möglich, die Trainingsintensität oder die Entwicklung des persönlichen Leistungsvermögens zu beurteilen.

Im Folgenden wird beschrieben, wie eine Leistungsmessung in Watt ähnlich einem Ergometer an einem Spinning-Bike nachgerüstet werden kann. Weitere Möglichkeiten sind die Messung der Herzfrequenz über einen Brustgurt und Anzeige von Kalorienverbrauch und Trittfrequenz.

Der beschriebene Umbau wurde an einem Schwinn Jonny G Elite Spinner aus den späten 1990er Jahren durchgeführt.

Theoretischer Hintergrund

Das Schwungrad



Die Abbildung zeigt einen schematischen Querschnitt durch ein typisches Schwungrad und die Abmessungen des Beispiels. Zur Vereinfachung wurde die Ausführung der Nabe nicht in die Rechnung mit einbezogen, da sie aufgrund des geringen Achsabstands nur wenig zum Trägheitsmoment beiträgt.

Zur Überprüfung wurden zunächst Volumen und Masse des Schwungrads berechnet:

$$V = \pi(d_1 \, {r_1}^2 + \, d_2((r_1 + r_2)^2 - {r_1}^2)) = \, 2.471 \, cm^3$$

Mit der Dichte (ρ) von Gusseisen von etwa 7,2 g/cm^3 ergibt sich die Masse des Schwungrads zu 17,8 kg. Dies stimmt gut mit der Werksangabe von 17,5 kg überein.

Für die weiteren Rechnungen wird das Trägheitsmoment des Schwungrades benötigt. Die allgemeine Formel ist:

$$I = 2\pi\rho \int_0^R r^3 \, h(r) \, \mathrm{d}r.$$

Für das betrachtete Schwungrad ergibt sich:

$$I = 2\pi\rho \left(d_1 \frac{{r_1}^4}{4} + d_2 \frac{(r_1 + r_2)^4 - {r_1}^4}{4} \right) = 0{,}6498 \, kg \, m^2$$

Mit

$$E_{rot} = \frac{1}{2} \, I \, \omega^2 \qquad \text{und} \qquad \omega = 2\pi RPS$$

erhält man für die Rotationsenergie bei einer Umdrehungszahl pro Sekunde RPS die Berechnungsformel und den Wert

$$E_{rot} = \, 2\pi^2 I \, RPS^2 = 12{,}83 \, kg \, m^2 RPS^2 \quad (1)$$

Beispiel: Die Übersetzung von Pedal zu Schwungrad beträgt 1 : 3,25. Bei einer Pedaldrehzahl (Kadenz) von 60 pro Minute beträgt die Schwungraddrehzahl 3,25 pro Sekunde. Damit ergibt sich eine Rotationsenergie von 135,5 J.

Bremse und Leistungsberechnung

Beim Treten ist neben der Bremswirkung auch die Reibung in Kette/Riemen und Lager zu überwinden. Hier wurde ermittelt, dass die dafür erforderliche Leistung weniger als 1 W beträgt. Aus diesem Grund wird dieser Beitrag im Folgenden nicht gesondert betrachtet.

Beim verwendeten Spinning-Bike erfolgt das Anpressen der Bremsbacken über eine Feder, die über eine Stellschraube vorgespannt wird. Die Federkraft und damit die Anpresskraft der Bremse

ist somit proportional zur Stellung der Schraube (pos). Da die (Gleit-)Reibungskraft unter Annahme eines konstanten Reibwerts ebenfalls proportional mit der Anpresskraft wächst, ergibt sich die Bremskraft F_B zu

$$F_B = k_1\, pos\ [N]$$

Zu beachten ist allerdings, dass am Anfang des Stellbereichs zunächst ein Anlegen der Bremsbacken erfolgt, so dass die Proportionalität erst dann gilt, wenn die Bremsbacken vollständig am Schwungrad anliegen.

Nun soll folgender Fall betrachtet werden: Das Schwungrad wird auf eine bestimmte Umdrehungszahl gebracht und dann im freien Lauf durch die Bremswirkung abgebremst.

Es lässt sich zeigen, dass die Umdrehungszahl RPS dann linear mit der Zeit t abfällt. Trägt man die z.B. mit Hilfe der unten beschriebenen Messeinrichtung gemessene Umdrehungszahl RPS über die Zeit t auf, so erhält man eine Regressionsgerade mit folgenden Parametern:

$$RPS(t) = m_{RPS}\, t + RPS_0$$

Die Rotationsenergie zu Beginn der Messung lässt sich gemäß Gleichung (1) errechnen. Aus dieser erhält man das Bremsmoment M_B mit der Gesamtzahl der Umdrehungen bis zum Stillstand, da F_B nicht bekannt ist:

$$M_B = \frac{E_{rot}}{\varphi_{ges}}$$

Aus der Regressionsgeraden erhält man

$$\varphi_{ges} = \pi \frac{{RPS_0}^2}{m_{RPS}}$$

und damit das von der Drehzahl unabhängige Bremsmoment:

$$M_B = 2\pi\, I\, m_{RPS}$$

Um das Schwungrad bei konstanter Drehzahl zu halten, muss das Antriebsmoment M_W das Bremsmoment kompensieren, also

$$M_W = -M_B = -2\pi\, I\, m_{RPS}$$

Die hierfür notwendige Leistung P_W bei einer Drehzahl RPS beträgt

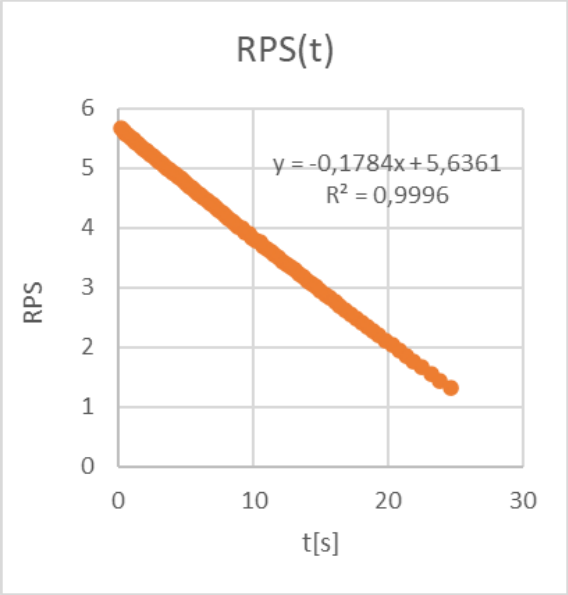
$$P_W = M_W\, 2\pi\, RPS = I\, 4\, \pi^2\, m_{RPS}\, RPS$$

Zur Berechnung der Leistung bei einem eingestellten Widerstand pos (Zahl der Umdrehungen des Einstellrades) wird der Leistungsfaktor K_P(pos) eingeführt werden. Dieser ergibt mit der Drehzahl des Schwungrads multipliziert die eingesetzte Leistung:

$$\frac{P_W}{RPS}(pos) = K_P(pos) = -I\, 4\, \pi^2\, m_{RPS}(pos)$$

Da von einem konstanten Reibwert zwischen Bremsbacken und Schwungrad ausgegangen wird, ist es empfehlenswert, diese Kalibrierung zeitweilig zu überprüfen. Bei Änderungen wie Austausch oder Reinigung/Schmierung der Bremsbacken sollte eine Neubestimmung von m(pos) vorgenommen werden.

Beispiel



Die Abbildung zeigt die Umdrehungszahl des Schwungrads (RPS) in Abhängigkeit von der Zeit beim gebremsten Auslaufen.

m_{RPS} beträgt -0,1784 s^{-2}

Mit dem weiter oben berechneten Trägheitsmoment I erhält man

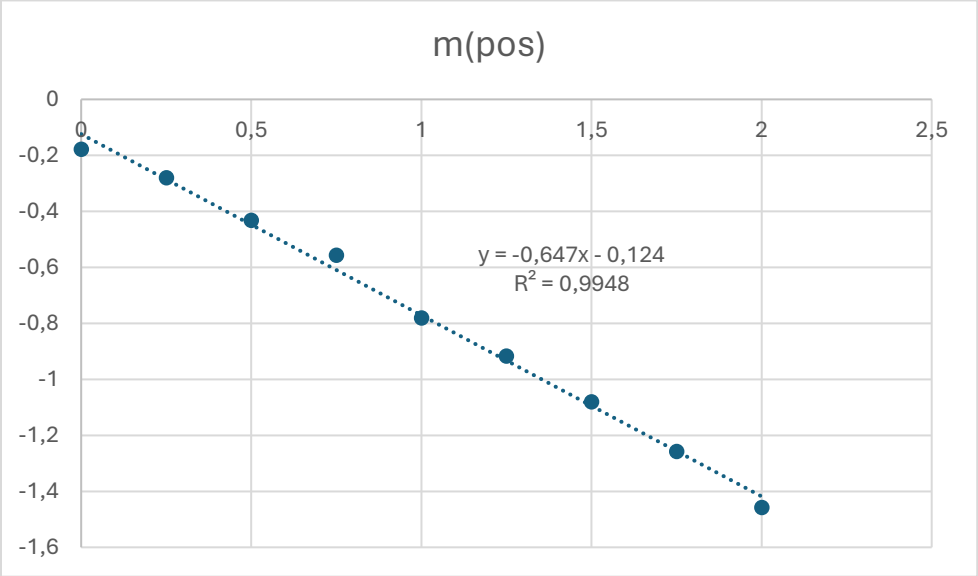
K_P = 4,58 kg m^2 s^{-2}

Bei einer Trittfrequenz von 60 min^{-1} beträgt die Schwungraddrehzahl aufgrund der Übersetzung 3,25 s^{-1}.

Die aufgebrachte Leistung ist dann

P_W = 14,9 kg m^2 s^{-3} = 14,9 W

Führt man die Bestimmung von m(pos) für einen Bereich von Widerstandseinstellungen pos durch, kann durch lineare Regression schließlich eine Formel zur Ermittlung von K_P aus pos erhalten werden, die für die Leistungsberechnung eingesetzt werden kann:



$$K_P(pos) = -(-0{,}124 - 0{,}647 pos)\, 4\,\pi^2\, I = 3{,}18 + 16{,}6\, pos$$

Da im Programm nicht die RPS des Schwungrads, sondern die minütliche Trittfrequenz der Pedale cad vorliegt, muss noch einmal mit dem Faktor 3,25/60 umgerechnet werden und man erhält:

$$P_W(pos, cad) = (0{,}172 + 0{,}899\, pos)\, cad$$

Diese Formel wird im Programm verwendet.

Ermittlung des Kalorienverbrauchs

Prinzipiell kann der Energieverbrauch zwar aus der Herzfrequenz errechnet werden, diese unterliegt jedoch bei gleicher Belastung großen inter- und intra-individuellen Schwankungen. Weiterhin gibt es hierfür verschiedene Berechnungsmethoden, die teilweise auch den Grundumsatz des Körpers mit beinhalten.

Da die Leistung über die Zeit des Trainings und damit die abgegebene Energie vorliegt, kann der Energieverbrauch des Trainings (ohne Grundumsatz) besser durch Division mit dem Wirkungsgrad η des Körpers errechnet werden.

Für Fahrradergometrie ist in (1) die Abhängigkeit der Wirkungsgrad von Leistung und Trittfrequenz (Kadenz) beschrieben. Diese Daten liegen in Form einer per Regression ermittelten Formel der Berechnung zugrunde:

$$\eta\ [\%] = 26{,}5 - 0{,}0037 * (80 - \frac{2000}{P} - cad)^2$$

$$P_{verbr}\ [kCal/\min] = 1{,}44 * \frac{P}{\eta}$$

Aufgrund der nicht sinnvollen Extrapolation für kleine Wert von P und cad ist der Minimalwert von η im Programm auf 10% begrenzt.

Hardware

Computer

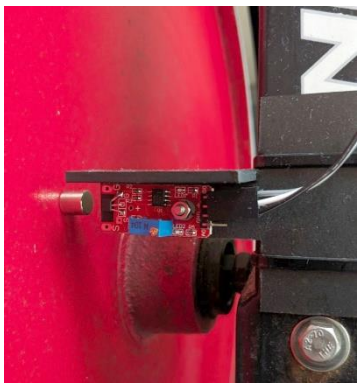
Für die Leistungsmessung wird ein Einplatinencomputer vom Typ Raspberry Pi (Raspi) 4 mit 1 GB Arbeitsspeicher und einem 7“ IPS-Touchscreen-Display (Auflösung 800x480 pixel) eingesetzt. Es wurde nicht geprüft, ob Raspberry Pi Typen mit geringerer CPU-Leistung (z.B. Zero 2, 3B+) ebenfalls geeignet sind.

Messung der Bremseinstellung

Da die Einstellung des Schraubrads über mehrere Umdrehungen erfolgt, wird ein Rotary Encoder vom Typ KY-40 mit 20 Impulsen pro Umdrehung verwendet. Dieser wird in einer geeigneten Anordnung mit der Achse des Einstellrads verbunden (siehe Abbildung).

Anschluss:

Raspi pin	**Encoder**
17 (3,3 V)	+
9 (GND)	GND
15 (GPIO 22)	CLK
13 (GPIO 27)	DT



Drehzahlmessung

Für die Messung der Schwungraddrehzahl kommt ein Hall-Sensor vom Typ KY-024 mit dem Sensor LM393 zum Einsatz. Dieser wird in passender Position im Bereich des Schwungrads befestigt, so dass ein am Schwungrad haftender Neodym-Magnet (Ø 8mm, Höhe 8mm) bei jeder Umdrehung einen Impuls des Sensors auslöst (siehe Abbildung). Anschluss:

Raspi pin	**Sensor**
1 (3,3 V)	+
6 (GND)	GND
11 (GPIO 17)	DO

Herzfrequenz-Messung

Zur Messung der Herzfrequenz wird ein Polar® H7 HR Sensor eingesetzt, der seine Messwerte über Bluetooth LE an den Computer sendet.

Software

Einrichtung des Raspberry Pi

Auf einer 64 GByte SD-Karte wird mit Hilfe der Software „Raspberry Pi Imager“ eine aktuelle 64-bit Version des Betriebssystems Debian 12 (Bookworm) installiert (2). Als weitere Optionen werden ein Netzwerkzugang über WLAN sowie der Fernzugriff über SSH und VNC eingerichtet.

Da die Programmierung auf einem Windows-PC erfolgen soll, wird Samba auf dem Raspi installiert (3) und das Home-Verzeichnis auf dem PC eingebunden. Zur Programmentwicklung in Python 3 kann das auf dem PC installierte IDE-Tool Geany eingesetzt werden.

Weiterhin werden folgende Python-Bibliotheken auf dem Raspi installiert: bitstruct, bleak, und python3-evdev.

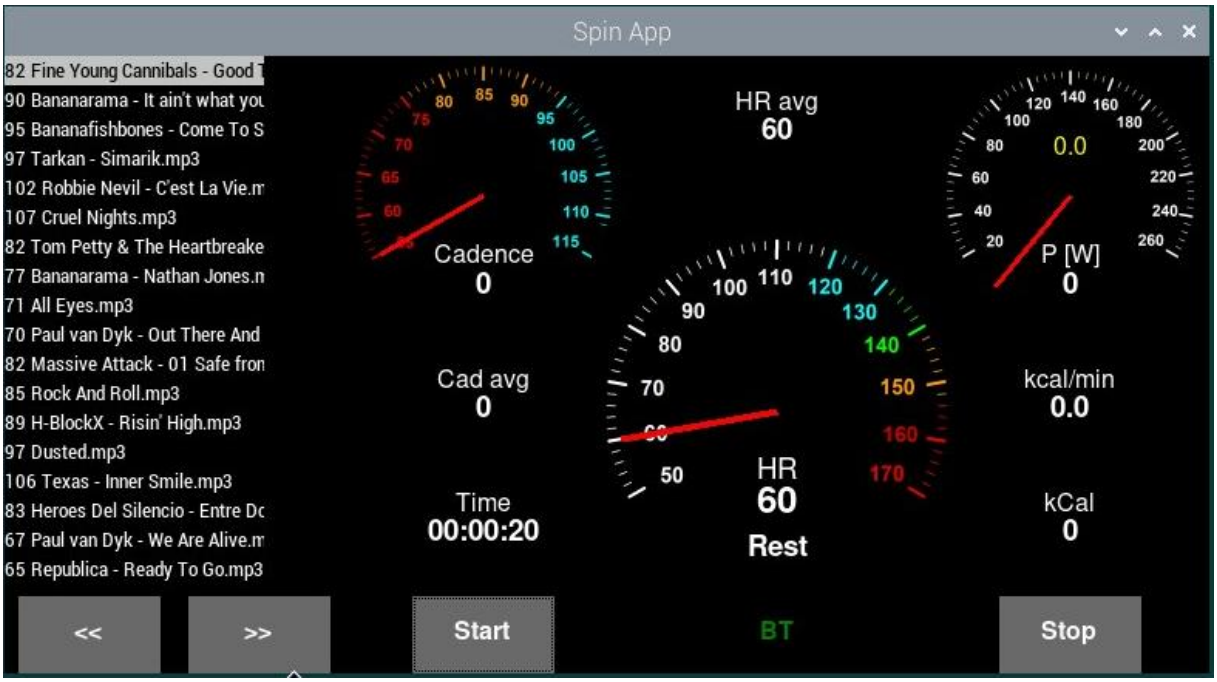
Zur Einbindung des Rotary Encoders über das Betriebssystem wird in der Datei /boot/config.txt die Zeile

```
dtoverlay=rotary-encoder,pin_a=22,pin_b=27,relative_axis=1
```

eingefügt.

Bedienoberfläche

Die Oberfläche zur Bedienung des Programms (siehe Abbildung) ist mit Hilfe der Tkinter-Bibliothek in Python realisiert. Die Realisierung sowie die Programmteile zur Wiedergabe der auf der linken Seite sichtbaren MP3-Playlist werden im Folgenden nicht weiter beschrieben. Zur MP3-Wiedergabe im Programm ist jedoch anzumerken, dass diese mit pygame einfacher und zuverlässiger umsetzbar ist als mit python-vlc.



Im Folgenden werden einige für die Messung von Herzfrequenz, Trittfrequenz und Leistung benötigte Programmteile beschrieben.

Initialisierung und Programmstart

```
from time import *
from time import time
import math
import asyncio
import bitstruct
import struct
import sys
from bleak import BleakClient
import RPi.GPIO as GPIO
import evdev

Cont = True                # Schleife läuft
Rec_on = False             # Aufzeichnung und Mittelung laufen
HR_Sum = 0
HR_Ct = 0
Cad_Sum = 0
Cad_Ct = 0
Encpos = 0
Pow_Sum = 0
Pow_Ct = 0
HR_max = 55
HRt = 75
kCm = 0
kCal = 0
Cad = 65
```

```
Pow = 55
StartTime = time()
enctask = None
channel = None
event = None

# Encoder
encd = evdev.InputDevice('/dev/input/by-path/platform-rotary@16-event')
Encpos = 0
pospround = 20            # Impulse pro Umdrehung des Encoders

# Pulsmessung BLE
address = ("00:22:D0:8A:B7:4B")  # Change to address of device
HR_MEAS = "00002A37-0000-1000-8000-00805F9B34FB"

# Hall Sensor
gpio_pin_number = 17      # GPIO Pin des Sensors
elapsed_time = 0          # Zeit der Umdrehungsmessung
rounds = 0                # Anzahl Umdrehungen
cad_timer = time()        # Initialisierung timer
gear_rat = 3.25           # Übersetzung zur Umrechnung in Trittfreq.
cad_val=0

# Main
…
Encpos = 0
if __name__ == "__main__":       # Start der asnychronen Schleife
    asyncio.run(run(address))
root.mainloop()
# Main Ende
```

Beim Programmstart muss der Widerstand am Spinning Bike in die 0-Position, die für die Kalibrierung Kp(pos), s.o. verwendet wurde, gestellt sein. Das Programm startet die asynchrone Schleife.

Asynchrone Schleife

```
async def run(address):
  global HRt, Cont, encd, enctask, channel, event
  loop = asyncio.get_running_loop()
  event_queue = asyncio.Queue()
  enctask = asyncio.ensure_future(enc_events(encd))

  async with BleakClient(address, timeout=35.0) as client:
    connected = BleakClient.is_connected
    if connected:

      def hr_val_handler(sender, data):
```

```
            global HRt
            (unused, rr_int, nrg_expnd, snsr_cntct_spprtd, snsr_detect, hr_fmt) \
              = bitstruct.unpack("b3b1b1b1b1b1", data)
            if hr_fmt:
              hr_val, = struct.unpack_from("<H", data, 1)
            else:
              hr_val, = struct.unpack_from("<B", data, 1)
            HRt = hr_val

          await init_GPIO()
          while Cont:
            await client.start_notify(HR_MEAS, hr_val_handler)
            await hr_update(HRt)
            await cadpow_update()

            if not BleakClient.is_connected:
              print("BLE disconnected")
            await asyncio.sleep(1)

        enctask.cancel()
        await BleakClient.disconnect(client)
        loop.stop()
```

Der Aufbau dieses Programmteils ist einer Anleitung für die Anbindung eines Brustgurt über BLE (4) entnommen und wurde entsprechend erweitert. Die Schleife wird in diesem Fall nur dann gestartet, wenn die BLE-Verbindung mit dem Brustgurt zustande gekommen ist.

Mit asyncio.ensure_future wird die asynchrone Verarbeitung der externen Events gesichert.
Die asynchronen Routinen zu Verarbeitung und Darstellung von Herzfrequenz (hr_update(HRt) sowie Trittfrequenz und Leistung (cad_pow_update() werden in der Schleife abgerufen, die einmal pro Sekunde durchlaufen wird.
Der Ausstieg aus der Schleife erfolgt über einen Button auf der Benutzeroberfläche, der die Variable Cont auf False setzt.

Drehzahlmessung

Initialisierung

```
async def init_GPIO(): # Initialisierung GPIO
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(gpio_pin_number,GPIO.IN,GPIO.PUD_UP)
    GPIO.add_event_detect(gpio_pin_number, GPIO.RISING, \
      callback = calculate_elapsed_time, bouncetime = 80)
```

Jedes Mal, wenn bei einer Umdrehung der Magnet den Hall-Sensor auslöst, wird ein Event getriggert und die Funktion calculate_elapsed_time aufgerufen.

Event-Behandlung

```
def calculate_elapsed_time(channel):
```

```
    global cad_timer, elapsed_time, rounds
    if rounds==0:               # Alter Wert wurde verarbeitet
      cad_timer = time()
      elapsed_time = 0
    else:                       # Aufsummieren Runden und Zeit
      elapsed_time = time() - cad_timer
    rounds+=1
```

Beim ersten Event (rounds = 0) wird der cad_timer initialisiert. Bei jedem nächsten Event wird die Zeit nach dem Timer-Start berechnet und die Umdrehungszahl inkrementiert. Zeit und Undrehungszahl stehen als globale Variablen zur Verfügung.

Abfrage der Encoderposition

```
async def enc_events(dev):
  global Encpos
  async for ev in dev.async_read_loop():
    if ev.type == evdev.ecodes.EV_REL:
      Encpos += ev.value
```

Bei jeder Änderung wird die Encoderposition in der globalen Variablen Encpos gespeichert. Jede Umdrehung des Einstellrads entsprich posprounde, also 20 Schritten.

Bearbeitung der Herzfrequenz

```
async def hr_update(HR):
    global root, HR_Sum, HR_Ct, HR_max, StartTime, kCal, kCm
    if Rec_on:
        # Hier anfügen: Anzeigen/Speichern der Herzfrequenz HR
        HR_Sum = HR_Sum + HR
        HR_Ct+=1
        HR_Avg = round(HR_Sum/HR_Ct)
        if HR > HR_max:
          HR_max = HR
        # Hier anfügen: Anzeigen max./durchschn. Herzfrequenz HR_max, HR_avg
        TimeDiff = time() - StartTime
        hrtimevar = strftime("%H:%M:%S", gmtime(TimeDiff))
        # Option: Anzeigen/Speichern der Trainingszeit hrtimevar
        kCal = kCal + kCm/60                                # Einmal pro s
        # Hier anfügen: Anzeigen/Speichern der verbrauchten Kalorien kCal
```

Bearbeitung von Trittfrequenz, Leistung und Kalorienverbrauch

```
async def cadpow_update():
  global rounds, updateflag, gear_rat, cad_val
```

```
def cad_update(cadv):
  global Cad_Sum, Cad_Ct
  if Rec_on:
    Cad_Sum = Cad_Sum + cadv
    Cad_Ct +=1
    Cad_Avg = round (Cad_Sum/Cad_Ct)
   # Hier anfügen: Anzeigen/Speichern der aktuellen/durchschn.
   # Trittfrequenz cadv, Cad_avg

def pow_update(cadv):
  global Pow_Sum, Pow_Ct, pospround, Encpos, kCm
  p_fact = 0.172 + 0.899 * (Encpos / pospround)          # Aus Kp(pos)
  if p_fact<0.01:
    p_fact=0.01
  powv = p_fact * cadv
  if Rec_on:
    Pow_Sum = Pow_Sum + powv
    Pow_Ct +=1
    Pow_Avg = round (Pow_Sum/Pow_Ct)
    if powv>0:
      etanet = 26.5-0.0037*math.pow((80-2000/powv-cadv), 2) # Berechnung
                                                            # Wirkungsgrad
      if etanet<10.0:
        etanet = 10.0
      kCm = round(14.4*powv/etanet)/10                      # Berechnung
                                                            # kCal/min
     # Hier anfügen: Anzeigen/Speichern der aktuellen/durchschn.
     # Leistung powv, Pow_Avg und des Kalorienverbrauchs pro
     # Minute kCm

if ((rounds-6) * elapsed_time) > 0:
  cad_val = 60*(rounds-1)/elapsed_time/gear_rat
  rounds = 0
cad_update(cad_val)
pow_update(cad_val)
```

Die Berechnung der Trittfrequenz wird nach jeweils mindestens sieben Umdrehungen des Schwungrads (ca. zwei Pedalumdrehungen) durchgeführt. Hierdurch ergibt sich ein Mittelungseffekt. Durch das Setzen von rounds auf 0 wird die Event-Behandlung neu initialisiert.

Literatur

(1) https://www.germanjournalsportsmedicine.com/archive/archive-2017/issue-9/the-efficiency-of-muscular-exercise/, abgerufen am 20.03.2024

(2) https://www.raspberrypi.com/software/, abgerufen am 23.03.2024

(3) https://www.raspberry-buy.de/Raspberry_Pi_Tutorial_Windows_Dateifreigabe_Samba_SMB_Server_Installation.html, abgerufen am 23.03.2024

(4) https://stackoverflow.com/questions/72539419/unable-to-read-heart-rate-service-over-bluetooth/72541361#72541361, abgerufen am 24.03.2024