PV Überschussladung mit Raspberry Pi und Go-e Charger

Inhalt

1. Voraussetzungen für den Raspberry Pi

Als Betriebssystem verwende ich das Raspbian mit Desktop, dann können später die Leistungsdaten per GUI angezeigt werden. Da der Raspberry im Home-Netzwerk hängt können dann jederzeit vom PC oder jedem mobilen Endgerät per Remote-Desktop die Daten angesehen werden.

Am Besten den Link folgen.

1.1 24/7 Dauerbetrieb

Der Raspberry Pi ist im Prinzip für den Dauerbetrieb ausgelegt. Die einzige Schwachstelle sind die begrenzten Schreibzugriffe auf die Speicherkarte. Um einen 24/7 Betrieb zu realisieren müssen diese möglichst vermieden werden. Die ermittelten Daten sollen auf keinen Fall auf die Speicherkarte geschrieben werden!

Vom Raspbian-System aus gibt es schon zahlreiche Systemzugriffe auf die SD-Karte, insbesondere wenn Log-Dateien in /var/log geschrieben werden.
Diese Schreibzugriffe kann man von der SD-Karte wegnehmen und in eine RAM-Disk in den Arbeitsspeicher auslagern – natürlich mit der Einschränkung, dass die Daten verloren gehen, wenn der Raspberry Pi vom Strom entfernt wird.

Sehen Sie dazu die Anleitung: Link

Diese Optionen habe ich auf meinem Raspberry ausprobiert aber leider ohne Erfolg, bzw. der Raspberry hat nicht mehr gebootet! Darum habe ich es dabei belassen, und finde mich damit ab evtl. alle paar Jahre die SD-Karte auszutuschen.

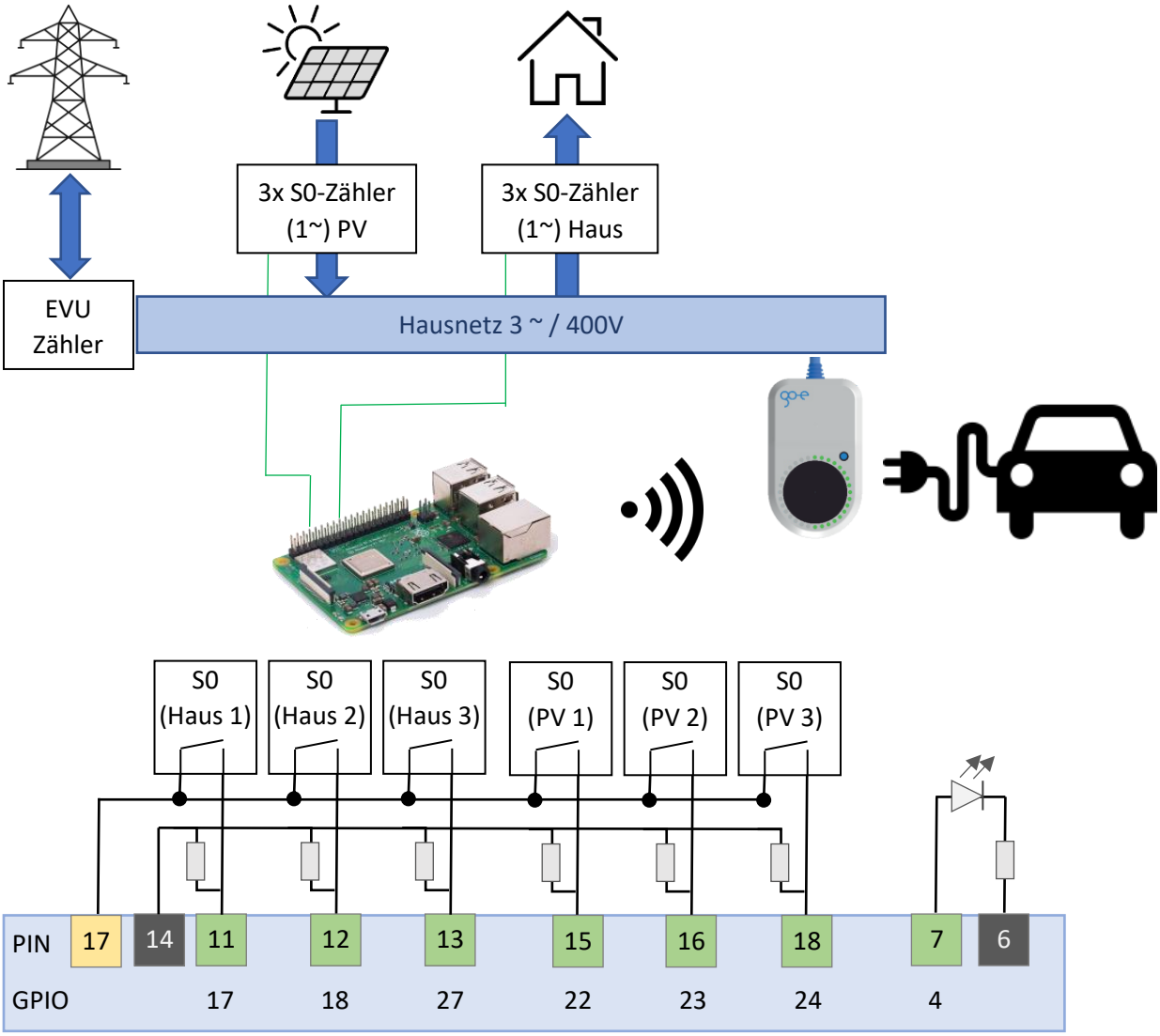
1.2 Python als Programmiersoftware

Die umgesetzte Anwendung ist an sich eine klassische Aufgabe für eine Hardware-SPS. Mit ein paar Tricks kann aber auch Python so verbogen werden, dass das Programm wie in einer SPS zyklisch abgearbeitet wird.
Python hat den weiteren Vorteil, dass es im Betriebssystem bereits enthalten ist, und damit alle Funktionen wie SPS, GUI zur Visualisierung der Werte und API zur Anbindung des Go-eChargers via Ethernet realisiert werden können.

1.3 Ermittlung der aktuellen PV- und Verbrachs Leistungsdaten über die On Board GPIO‘s

Die GPOI’s des Raspberry können als EIN- und AUSGÄNGE genutzt werden. Wir benötigen nur die Eingänge. Die Treiberspannung von 3.3V ist dazu völlig ausreichend. Bei offenem Ausgang des S0 Zählers wird der Eingang am GPIO durch einen ca. 40k Widerstand sofort wieder auf 0V gezogen. Zur Messung der aktuellen Leistung können herkömmliche S0-Zähler mit Impulsausgang verwendet werden. Je kleiner dabei die Wertigkeit des Impulses ist, desto genauer und schneller ist der aktuell ermittelte Leistungswert. Ich habe für meine Anwendung die S0-Zähler **B+G E-Tech DRS155DC-V3** verwendet mit 2000 Impulsen/kWh, also 0,5W pro Impuls. Im Raspberry wird dann die Zeit zwischen 2 Impulsen gemessen und daraus die wieder die aktuelle Leistung mit folgender Formel berechnet:
P (in W) = (3600s/aktuelle Laufzeit (in s)) * 0,5W.
Die aktuelle Leistung wird für jede Phase des Solargenerators und jede Phase des aktuellen Verbrauches der Hausinstallation (***ohne den Strom für die Wallbox!!***) parallel ermittelt. Aus diesen Werten kann dann der Verfügbare Strom, der zum Laden des E-Autos verfügbar ist berechnet werden, bei einphasigem Laden sogar unter Berücksichtigung der Schieflastverordnung.
Der zusätzliche GPIO 4 dient nur zur Anzeige des Lifebit (1Hz), um ohne einen Bildschirm zu erkennen ob das Python Programm läuft.

1.4 Prinzipschaltbild der Installation



2. Das PYTHON Programm

Das Programm ist in verschiedene Abschnitte und mehrere Funktionen unterteilt. Messung der aktuellen Leistungen, Steuerung des Go-e Chargers und die Anzeige der Daten im GUI. Diese Funktionen laufen unabhängig voneinander in einzelnen Thread's. Die Kommunikation zwischen den einzelnen Thread's wird über globale Variablen realisiert.

2.1 Globale Vorbedingungen

Aufruf der im Programm benötigten Python Funktionen

```
#!/usr/bin/python3
import os
import threading
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
import time
from datetime import datetime
import tkinter
from tkinter import *
```

Definition und Vorbelegung der globalen Variablen

```
#Global tags:
global exitFlag
global vorOrt, conect
#Im weiteren Programm steht N für Netzbedarf und S für Solarstrom.
global N1, N2, N3
global S1, S2, S3
global Strom, Lsoll, Frg, Send_ok, Send_fail
exitFlag, vorOrt, conect = 0, 0, 0
S1, S2, S3 = 0, 0, 0
N1, N2, N3 = 0, 0, 0
Strom, Lsoll, Frg, Send_ok, Send_fail = 0, 0, 0, 0, 0
```

Definition der GPIO's

```
GPIO.setwarnings(False)
GPIO.setup(4, GPIO.OUT)  #Livebit LED zur Anzeige dass das Programm noch läuft
GPIO.setup(17, GPIO.IN)
GPIO.setup(18, GPIO.IN)
GPIO.setup(22, GPIO.IN)
GPIO.setup(23, GPIO.IN)
GPIO.setup(24, GPIO.IN)
GPIO.setup(27, GPIO.IN)
```

2.2 Messung aktueller Leistungsdaten

Vorbereitung und Variablendefinition des Thread „messen“

```
#Threads
class messen(threading.Thread):
   def __init__(self, threadID, name):
      threading.Thread.__init__(self)
      self.threadID = threadID
      self.name = name
   def run(self):
      #print("\nStarting " + self.name)
      msec_act = datetime.now().microsecond
      msec_old = 0
      global N1, N2, N3
      global S1, S2, S3
      global Sendung, Send_ok, Send_fail
      Run1N, Run2N, Run3N = 0.0, 0.0, 0.0
      Run1S, Run2S, Run3S = 0.0, 0.0, 0.0
      Bit1N, Bit2N, Bit3N = 0, 0, 0
      Bit1S, Bit2S, Bit3S = 0, 0, 0
      E_N, E_S, E_Ntag, E_Stag = 0.0, 0.0, 0.0, 0.0  # Energiezähler in kWh
      Stunde = datetime.now().hour
```

Definition der Funktion getWatt. Darin wird die aktuelle Leistung „watt“ und der Energiezähler „wattH“ berechnet.

```
def getWatt(watt, wattH, runtime, flag, io, cycle):
    Leistung, Energie, Laufzeit, Bit = watt, wattH, runtime, flag
    if GPIO.input(io) == 0 and Laufzeit < 361000:
        Laufzeit = Laufzeit + cycle
        Bit = 0
    else:
        if Bit == 0:
            if Laufzeit >= 0.1: #in Millisekunden
                Laufzeit = Laufzeit / 1000 #in Sekunden

                #Ist die Laufzeit > 360 Sekunden ergibt sich eine Leistung von < 5.
                #Für die Messung ist diese Geringe Leistung irrelevant
                #und wird somit auf 0 gesetzt.
                if Laufzeit > 360:
                    Leistung = 0
                else:
                    Leistung = round(1800 / Laufzeit)
                    Energie = Energie + 0.5
                Laufzeit = 0
                Bit = 1
        else:
            Laufzeit = Laufzeit + cycle
    return Leistung, Energie, Laufzeit, Bit
```

Die im Thread „messen“ enthaltene Endlosschleife läuft im 10ms Intervall zur kontinuierlichen Messung der Energiedaten. Das Intervall ist so kurz gewählt da der Impuls des S0-Zählers nur eine Dauer von 30ms aufweist. Die Funktion getWatt wird darin 6x mit verschiedenen Parametern aufgerufen. Einmal am Tag um 8:00 werden per Printausgabe die Energiesummen die den EVU Zähler passiert haben ausgegeben. Die Auswertung ist ganz nett um z.B. den Verbrauch über Nacht zu ermitteln.

```
    while True:
      time.sleep(0.01)
      #Ermittlung der Cykluszeit im Millisekunden
      msec_old = msec_act
      msec_act = datetime.now().microsecond
      cycle = (msec_act - msec_old) / 1000
      if msec_act < msec_old: #Überlauf bei den Mikrosekunden
        cycle= (msec_act - msec_old + 1000000) / 1000

      N1, E_N, Run1N, Bit1N = getWatt(N1, E_N, Run1N, Bit1N, 17, cycle)
      N2, E_N, Run2N, Bit2N = getWatt(N2, E_N, Run2N, Bit2N, 18, cycle)
      N3, E_N, Run3N, Bit3N = getWatt(N3, E_N, Run3N, Bit3N, 27, cycle)
      S1, E_S, Run1S, Bit1S = getWatt(S1, E_S, Run1S, Bit1S, 22, cycle)
      S2, E_S, Run2S, Bit2S = getWatt(S2, E_S, Run2S, Bit2S, 23, cycle)
      S3, E_S, Run3S, Bit3S = getWatt(S3, E_S, Run3S, Bit3S, 24, cycle)
      #Ermittlung des Überhanges pro Stunde
      if Stunde != datetime.now().hour:
        E_S = round(E_S/100) / 10 #Erzeugte Solarenergie
        E_N = round(E_N/100) / 10 #Verbrauchte Netzleistung
        if E_N > E_S:
          E_Ntag = E_Ntag + E_N - E_S #Tagessumme Netzleistung
        else:
          E_Stag = E_Stag + E_S - E_N #Tagessumme PV Einspeisung
        Stunde = datetime.now().hour
        E_N, E_S = 0.0, 0.0

        if Stunde == 8: #Printausgabe um 8:00 Uhr
          print (str(datetime.now()) + "   Überschuss Solar: " + str(E_Stag) +"  Netzbedarf: " +
str(E_Ntag) +"kWh")
          E_Ntag, E_Stag = 0.0, 0.0 #Rücksetzen Energiezähler
          #Rücksetzen Telegrammzähler gute bzw. fehlerhafte Telegramme
          Send_ok, Send_fail = 0, 0
          print ("Fehlerspeicher: "+ str(Send_ok) +" / "+ str(Send_fail))

      if msec_act < 500000:          #Life Bit 1Hz
        GPIO.output(4, GPIO.HIGH)    #LED am GPIO angeschlossen als "RUN" Anzeige
      else:
        GPIO.output(4, GPIO.LOW)
      if exitFlag:
        break
```

2.3 Fernsteuern des Go-e Chargers

Vorbereitung und Variablendefinition des Thread „Goe“

```
#Go-eCharger steuern
class Goe(threading.Thread):
    def __init__(self, threadID, name):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
    def run(self):
        from threading import Thread
        global N1, N2, N3
        global S1, S2, S3
        global conect, vorOrt, Send_ok, Send_fail
        global Strom, Lsoll, Frg
        Frg, Frg_last, Strom_last = 0, 9, 99
        #Vorbelegung mit 9 bzw. 99 um den ersten Sendebefehl anzustoßen
        Frg_send, Strom_send = 0, 0 #Zähler für fehlerhafte Sendeaufträge
        Lslv = 0   #Leistung Schieflastverordnung
```

Definition der Funktion „Send“ mit der später der Sendeauftrag zum Go-e Charger abgesetzt wird.
Hinweis: Gibt es Probleme beim Verbindungsaufbau über WLAN wird der aktive Thread u.U. gestoppt. Deshalb wird später jeder Sendeauftrag in einem eigenen temporären Thread bearbeitet. Über die globale Variable „conect“ wird dann ausgewertet ob der Sendeauftrag erfolgreich war.

```
    def Send(api,value):
        import requests
        from requests.exceptions import Timeout
        global conect
        hostname = "http://go-eCharger.fritz.box/mqtt?payload"
        url = hostname + api + str(value)
        try:
            r = requests.get(url, timeout=3)
        except Timeout:
            conect = 0 #keine Verbindung
        else:
            conect = 1 #Verbindung OK
        time.sleep(3.5)
```

In der folgenden Endlosschleife wird im Zyklus von 100ms der aktuell verfügbare Überschuss der PV-Anlage ermittelt. DieVariable „Frg“ steht für die generelle Freigabe zum Laden. Es muss dabei mindestens ein Überschuss von 6A vorhanden sein. In der Variable „Strom“ ist dann der aktuelle Sollwert für den Ladestrom hinterlegt.

```
while True:
    # Ermittlen Stromstärke für Go-eCahrger - angeschlossen auf Phase L3
    time.sleep(0.1)
    if N1 >  N2:                    #4650W Schieflastverordnung
        Lslv = round(4650 + N2 - N3)
    else:
        Lslv = round(4650 + N1 - N3)
    if Lslv <= 0:
        Lslv = 0
    #maximle verfügbare Ladeleistung, abzüglich 200W Hausgrundversorgung
    Lmax = round((S1 + S2 + S3) - (N1 + N2 + N3) - 200)
    if Lmax <= 0:
        Lmax = 0

    if Lslv > Lmax:
        Lsoll = Lmax
    else:
        Lsoll = Lslv

    Strom_soll = round((Lsoll + 10) / 230)
    #Einstellung maximaler Ladestrom abhängig von der Sicherung
    if Strom_soll > 20:
        Strom_soll = 20

    if Strom_soll < 6:
        Strom_soll = 6
        Frg_soll = 0
    else:
        Frg_soll = 1
    if vorOrt == 1:
        Frg_soll = 1
    Frg = Frg_soll
    Strom = Strom_soll
```

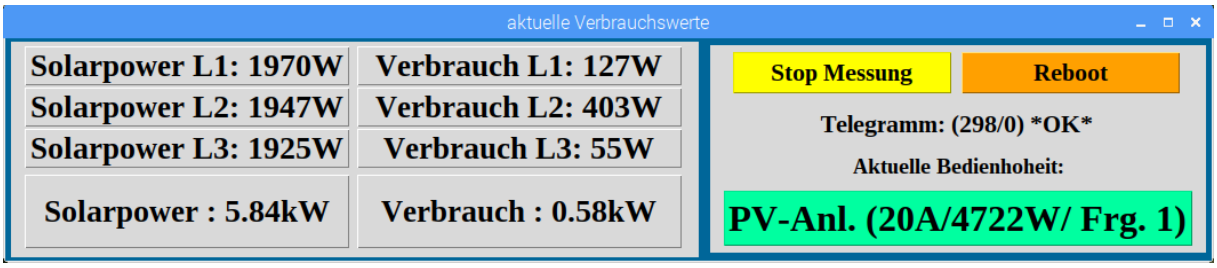
In den folgenden temporären Thread's wird je ein Sendeauftrag gestartet wenn die entsprechende Vorbedingung erfüllt ist. Die Variablen „**Frg_send**“ und „ **Strom_send**“ zählen die fehlerhaften Sendeaufträge und veranlassen bis zu 5 Wiederholungen. Mit der Variable „**vorOrt**“ kann über das GUI, s.weiter unten in der Beschreibung, auch eine generelle Freigabe zur Ladung erteilt werden. Wenn z.B. das Auto über Nacht geladen werden soll. Der Ladestrom kann dann direkt am Go-e Charger mit der Taste eingestellt werden.

```
#write to go-e
if Frg != Frg_last or Frg_send > 0:
   Frg_last = Frg
   t = Thread(target=Send, args=("=alw=",Frg))
   t.start()
   time.sleep(4)
   if conect:
      Send_ok = Send_ok + 1
      Frg_send = 0
   else:
      Send_fail = Send_fail + 1
      Frg_send = Frg_send + 1
      print(str(datetime.now()) +" Send Freigabe Fehler("+str(Frg)+"):"+str(Frg_send))
   if Frg_send > 4:
      Frg_send = 0

if vorOrt == 0 and Strom != Strom_last or Strom_send > 0:
   Strom_last = Strom
   t = Thread(target=Send, args=("=amp=",Strom))
   t.start()
   time.sleep(4)
   if conect:
      Send_ok = Send_ok + 1
      Strom_send = 0
   else:
      Send_fail = Send_fail + 1
      Strom_send = Strom_send + 1
      print(str(datetime.now()) +" Send Strom Fehler("+str(Strom)+"A):"+str(Strom_send))
   if Strom_send > 4:
      Strom_send = 0

if exitFlag:
   break
```

2.4 Erstellen eines GUI zur Visualisierung und Steuerung des Go-e Chargers

Im folgenden Thread wird das GUI erstellt, mit dem z.B. via Remote Control vom Handy aus aktuelle Istwerte angezeigt bzw. bestimmte Funktionen ausgeführt werden .

Im linken Fenster werden die Aktualwerte angezeigt.
Im rechten Fenster kann die Messung gestoppt, bzw. ein Reboot des Raspberry angefordert werden. In der zweiten Zeile wird der Zähler der Sendeaufträge (gute/fehlerhafte) und der Status des Letzten Sendeauftrages (*OK*/*Fehler*) angezeigt. Im Unteren Schalter kann die Bedienhoheit gewechselt werden, und es werden der Sollstrom, die Überschussleistung und Status der Ladefreigabe angegeben.



Vorbereitung Hauptfenster des GUI

```
#GUI Anzeige im Display
class myGUI (threading.Thread):
   def __init__(self, threadID, name):
      threading.Thread.__init__(self)
      self.threadID = threadID
      self.name = name

   def run(self):

      root = Tk() #Fenseter erstellen
      root.wm_title("aktuelle Verbrauchswerte") #Fenster Titel
      root.config(background = "#006699") #Hintergrundfarbe
```

Festlegung verschiedener interner Funktionen. Mit „**refresch**“ werden die Aktualwerte im GUI upgedatet. Mit „**killmess**“ wird die Messusng unterbrochen. Kann nur mit einem Neustart des Python Script rückgängig gemacht werden. Mit „**newstart**“ wird der Raspberry neu gebootet. Mit „**location**“ wird die Bedienhoheit des Go-e geändert.

```
#Funktionen festlegen
def refresh():
    global vorOrt, Frg, conect
    AnzS1.set("Solarpower L1: " + str(S1) + "W")
    AnzS2.set("Solarpower L2: " + str(S2) + "W")
    AnzS3.set("Solarpower L3: " + str(S3) + "W")
    AnzSges.set("Solarpower : " + str(round((S1+S2+S3)/10)/100) + "kW")

    AnzN1.set("Verbrauch L1: " + str(N1) + "W")
    AnzN2.set("Verbrauch L2: " + str(N2) + "W")
    AnzN3.set("Verbrauch L3: " + str(N3) + "W")
    AnzNges.set("Verbrauch : " + str(round((N1+N2+N3)/10)/100) + "kW")
    if vorOrt == 0:
        SelectControl.set("PV-Anl. (" + str(Strom) +"A/" + str(Lsoll) +"W/ Frg. "+str(Frg)+")")
    else:
        SelectControl.set("Wallb. (" + str(Strom) +"A/" + str(Lsoll) +"W/ Frg. "+str(Frg)+")")

    if conect == 0:
        ConText.set("Telegramm: ("+str(Send_ok)+"/"+str(Send_fail)+") *Fehler*")
    else:
        ConText.set("Telegramm: ("+str(Send_ok)+"/"+str(Send_fail)+") *OK*")

    root.after(1000, refresh) #GUI wird einmal pro Sekunde upgedatet

def killmess():
    global exitFlag
    exitFlag = 1
    #print("Manueller Abbruch " + str(exitFlag))

def newstart():
    global exitFlag
    exitFlag = 1
    #print("Manueller Abbruch und Reboot " + str(exitFlag))
    time.sleep(10)
    os.system("sudo reboot")

def location():
    global vorOrt
    if vorOrt == 0:
        vorOrt = 1
    else:
        vorOrt = 0
```

Einteilung des Haupt GUI in zwei Fenster.

```
leftFrame = Frame(root, width=400, height=240)
leftFrame.grid(row=0, column=0, padx=5, pady=3)

rightFrame = Frame(root, width=400, height=240)
rightFrame.grid(row=0, column=1, padx=5, pady=3)
```

Die Anzeigevariablen werden als „StringVar()“ definiert. Diese aktualisieren sich somit selbstständig im GUI.

```
AnzS1 = StringVar()
AnzS2 = StringVar()
AnzS3 = StringVar()
AnzSges = StringVar()
AnzN1 = StringVar()
AnzN2 = StringVar()
AnzN3 = StringVar()
AnzNges = StringVar()
SelectControl = StringVar()
ConText = StringVar()
```

Aktualwerte im linken Fenster des GUI.

```
    varFrame = Frame(leftFrame)
    varFrame.grid(row=0, column=0, padx=10, pady=3)
    lS1 = Label(varFrame, width=20, height=1, relief = RAISED, font=("Times","20","bold"),
textvariable = AnzS1)
    lS1.grid(row=0, column=0, padx=2, pady=1)
    lS2 = Label(varFrame, width=20, height=1, relief = RAISED, font=("Times","20","bold"),
textvariable = AnzS2)
    lS2.grid(row=1, column=0, padx=2, pady=1)
    lS3 = Label(varFrame, width=20, height=1, relief = RAISED, font=("Times","20","bold"),
textvariable = AnzS3)
    lS3.grid(row=2, column=0, padx=2, pady=1)
    lSges = Label(varFrame, width=20, height=2, relief= RAISED, font=("Times","20","bold"),
textvariable= AnzSges)
    lSges.grid(row=3, column=0, padx=2, pady=5)

    lN1 = Label(varFrame, width=20, height=1, relief = RAISED, font=("Times","20","bold"),
textvariable = AnzN1)
    lN1.grid(row=0, column=1, padx=5, pady=1)
    lN2 = Label(varFrame, width=20, height=1, relief = RAISED, font=("Times","20","bold"),
textvariable = AnzN2)
    lN2.grid(row=1, column=1, padx=5, pady=1)
    lN3 = Label(varFrame, width=20, height=1, relief = RAISED, font=("Times","20","bold"),
textvariable = AnzN3)
    lN3.grid(row=2, column=1, padx=5, pady=1)
    lNges = Label(varFrame, width=20, height=2, relief= RAISED, font=("Times","20","bold"),
textvariable= AnzNges)
    lNges.grid(row=3, column=1, padx=5, pady=5)
```

Anzeigen und Bedientasten im rechten Fenster des GUI.

```
    buttonFrame1 = Frame(rightFrame)
    buttonFrame1.grid(row=1, column=1, padx=1, pady=3)

    B1 = Button(buttonFrame1, text="Stop Messung", bg="#FFFF00", width=15,
font=("Times","16","bold"), command=killmess)
    B1.grid(row=1, column=0, padx=2, pady=2)
    B2 = Button(buttonFrame1, text="Reboot", bg="#FFA000", width=15,
font=("Times","16","bold"), command=newstart)
    B2.grid(row=1, column=1, padx=5, pady=2)

    buttonFrame2 = Frame(rightFrame)
    buttonFrame2.grid(row=2, column=1, padx=1, pady=2)

    Tx1 = Label(buttonFrame2, width=25, height=1, font=("Times","16","bold"), textvariable=
ConText)
    Tx1.grid(row=2, column=0, padx=5, pady=5)
    Tx2 = Label(buttonFrame2, width=25, height=1, font=("Times","14","bold"), text= ("Aktuelle
Bedienhoheit:"))
    Tx2.grid(row=3, column=0, padx=5, pady=5)
    BSelect = Button(buttonFrame2, textvariable= SelectControl, bg= "#00FFA0", width=24,
font=("Times","24","bold"), command=location)
    BSelect.grid(row=4, column=0, padx=10, pady=3)
```

Mit dem Befehl „**root.after**“ wird zur Aktualisierung der Anzeigen zum ersten mal die Funktion „**refresh**“ aufgerufen.

```
    root.after(1000, refresh) #GUI wird das erste mal upgedatet
    root.mainloop()
```

Zum Abschluss werden die drei verschiedenen Thread’s gestartet.

```
thread1 = messen(1, "Datenerfassung") # Ablauf im 10ms Zyklus
thread2 = Goe(2, "Go-e steuern") # Ablauf im 100ms Zyklus
thread3 = myGUI(3, "GUI") # Ablauf auf Anforderung bzw. im 1s Zyklus

# Start new Threads
thread1.start()
thread2.start()
thread3.start()
thread1.join()
thread2.join()
thread3.join()
time.sleep(0.5)
```

Das wars auch schon. 😊