Python & MicroPython: Programmieren lernen für Einsteiger

Interaktiv mit Tkinter!

Teil 10

In den letzten Beiträgen dieser Serie ging es überwiegend um Datenausgaben, wie allgemeine Zahlenwerte, Sensordaten oder auch Lichtsignale von LEDs. Eine der wichtigsten Vorteile von Python ist jedoch, dass auch interaktive Steuerungen sehr leicht programmiert werden können. Hierfür bietet sich die Tkinter-Bibliothek an. Tkinter wird als standardmäßige Python-Bibliothek zur Erstellung grafischer Benutzeroberflächen (GUIs – „Grafical User Interface") verwendet. Mit Tkinter können Fenster, Buttons, Textfelder, Menüs und andere GUI-Elemente erstellt werden. Steuerungsaufgaben, vom einfachen Ein- und Ausschalten einer LED bis hin zu kompletten Heimautomatisierungen wie Steuerung von Motoren für Jalousien, Heizungen oder Klimaanlagen können damit problemlos umgesetzt werden.

Flexible Steuerung über Fenster

Die Tkinter-Bibliothek ist nach der Installation von Python üblicherweise vollständig einsatzbereit. Im Bedarfsfall kann die Nachinstallation mit:

```
sudo apt-get install python3-tk
```

erfolgen.

Die ordnungsgemäße Funktion wird mit einem kurzen Testprogramm überprüft:

```
import tkinter
tkinter._test()
```

Die _test-Methode sorgt dafür, dass ein eigenständiges Fenster erzeugt wird (Bild 1).

Innerhalb dieses Fensters findet sich der Text „This is Tcl/Tk version x.y“ und darunter folgt eine weitere Zeile mit Text. Dann kommt ein sogenannter „Button“ mit der Aufschrift „Click me!“. Darauf kann man mit der Maus oder bei Touch-Bildschirmen auch mit dem Finger klicken. Bei jedem Klick erscheint eine weitere eckige Klammerebene um den Text „Click me!“

Darunter findet sich der QUIT-Button, über den die Applikation geschlossen und wodurch das Programm beendet werden kann.

Man erkennt, dass durch die _test-Methode bereits verschiedene Aufgaben im Hintergrund umgesetzt werden. Anhand dieses einfachen Beispiels erklären sich die wichtigsten Grundbegriffe sowie der Aufbau einer Python-Tkinter-GUI-Anwendung nahezu von selbst.

Bei jeder GUI wird zunächst dafür gesorgt, dass sich ein oder mehrere Fenster öffnen. Dabei handelt es sich um die eigentliche Programmoberfläche, die wie die Fenster anderer gängiger Computeranwendungen vergrößert, minimiert und über das Kreuzsymbol geschlossen werden können. Sobald das Hauptfenster selbst erstellt worden ist, wird dieses noch mit sogenannten „Widgets“ befüllt.

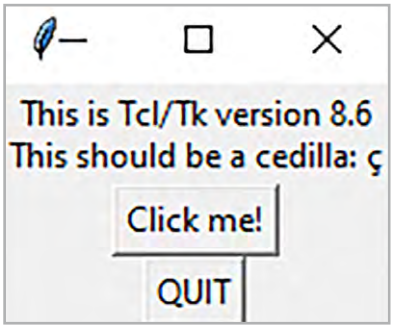

Bild 1: Testfenster

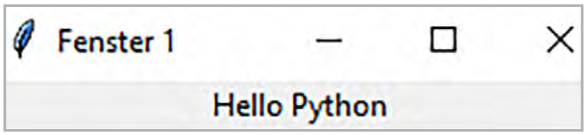

Bild 2: Tkinter-Fenster mit Ausgabe

Die Python-Tkinter-Widgets

Control- bzw. Steuerelemente werden auch als „Widgets“ bezeichnet. Der Text in einem Widget ist ein sogenannter „Label“. Dabei handelt es sich um einen ersten wichtigen Baustein der GUI. Auch der „Click me!“-Button ist ein solches Widget.

Neben Labels und Buttons stehen noch zahlreiche weitere Widgets zur Verfügung. Einige werden später noch genauer betrachtet. Zunächst soll die Erzeugung eines GUI-Fensters näher erläutert werden. Mit tkinter.Tk() wird ein sogenanntes Tk-Objekt erzeugt, welches das Hauptfenster der GUI darstellt. Die Referenz auf dieses Objekt kann in einer Variablen „root“ gespeichert werden:

```
root = tkinter.Tk()
```

Damit kann im weiteren Verlauf des Programms über diese Variable auf das gerade erzeugte Tk-Objekt zugegriffen werden. Wird das Programm ausgeführt, stellt man jedoch fest, dass noch kein GUI-Fenster erscheint. Das liegt daran, dass die sogenannte Tkinter-Event-Loop noch gestartet werden muss. Dies erfolgt über das Objekt, auf welches mit der Variable root referenziert wird:

```
root.mainloop()
```

Nun erscheint das zugehörige GUI-Fenster, zunächst noch ohne Text und Titel (Bild 2). Dieses Fenster ist bereits voll funktionsfähig und kann vergrößert, minimiert oder geschlossen werden. Um in diesem Fenster den Text „Hallo Welt“ auszugeben, wird ein sogenanntes Label-Widget benötigt:

```
import tkinter

root = tkinter.Tk()
label = tkinter.Label(root, text="Hello Python")
label.pack()

root.mainloop()
```

Für dieses Beispiel wurde der einfachste Layout-Manager verwendet, der mit der Methode „pack“ aufgerufen werden kann. Das Fenster wird nur so groß, wie der Platz für den Text es erfordert, sodass genau das angezeigt wird, was sich in der GUI befindet (Bild 2). Es wird allerdings die Möglichkeit geboten, das Fenster zu vergrößern, indem in eine Ecke geklickt wird, woraufhin ein Pfeil erscheint und das Fenster mit der Maus größer gezogen werden kann. So wie es bereits von anderen Anwendungen bekannt ist, in denen ebenfalls die Größe mithilfe der Maus beliebig angepasst werden kann.

Wenn das Fenster vergrößert wird, wird das Label durchgehend ganz oben zentriert angezeigt. Das liegt daran, dass der Layout-Manager „pack“ standardmäßig dafür sorgt, dass alle Informationen möglichst dicht in das Fenster ge-“packt“ werden. Die Ausrichtung des Labels und die Fenstergröße können allerdings modifiziert werden, indem dem Layout-Manager entsprechende Attribute hinzugefügt werden. Hierzu kann die geometry-Methode verwendet werden, z. B.:

```
root.geometry("breite x höhe" +x +y)
```

Diese gibt dem Fenster eine festgelegte Breite und Höhe. Der Parameter x bezeichnet zusätzlich die horizontale Position des Fensters. Zum Beispiel bedeutet ein Wert von +50, dass der linke Rand des Fensters 50 Pixel vom linken Rand des Bildschirms entfernt positioniert werden soll. Der Parameter y bezeichnet entsprechend die vertikale Position des Fensters. Über die title-Methode kann ein eigener Titel festgelegt werden. Das Fenster kann mit der Maus natürlich weiterhin beliebig verschoben und in der Größe verändert werden. Widgets werden nacheinander in einer bestimmten Reihenfolge (horizontal oder vertikal) „gestapelt“. Das ist ideal für einfache Layouts, bei denen Widgets in einer Richtung angeordnet werden sollen. Zusätzlich stehen einige wichtige Optionen zur Verfügung:

- **side (z. B. tk.TOP, tk.BOTTOM, tk.LEFT, tk.RIGHT)** für die Ausrichtung
- **fill (z. B. tk.X, tk.Y, tk.BOTH)**: bestimmt, ob das Widget den verfügbaren Raum ausfüllt.
- **expand (True/False)**: gibt an, ob das Widget zusätzlichen Raum nutzen soll

Das folgende Programm (stuctured_window.py) erklärt die Anwendung und liefert als Ausgabe Bild 3.

```
import tkinter as tk

root = tk.Tk()
root.title("Pack Beispiel mit Textfeldern")
root.geometry("400x300")

entry1 = tk.Entry(root, bg="lightblue")
entry1.insert(0, "Oben, keine Füllung")
entry1.pack(side=tk.TOP)

entry2 = tk.Entry(root, bg="lightgreen")
entry2.insert(0, "Unten, horizontal gefüllt")
entry2.pack(side=tk.BOTTOM, fill=tk.X)

entry3 = tk.Entry(root, bg="lightyellow")
entry3.insert(0, "Links, vertikal gefüllt")
entry3.pack(side=tk.LEFT, fill=tk.Y)

entry4 = tk.Entry(root, bg="lightpink")
entry4.insert(0, "Rechts, komplett gefüllt + expandiert")
entry4.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

root.mainloop()
```

Bild 3: Strukturiertes Fenster

Dabei ist zu beachten, dass das genaue Aussehen des Fensters vom Betriebssystem (Windows, Linux, oder PiOS) abhängen kann.

Im Beispiel oben wurden zusätzlich verschiedene Hintergrundfarben (bg für background) der Textfelder (tk.Entry) mit dem Parameter bg (background) festgelegt, um sie visuell voneinander zu unterscheiden. Die Vordergrundfarbe (Textfarbe oder Farbe der Inhalte eines Widgets) kann mit dem Parameter fg (foreground) festgelegt werden. Er bestimmt, in welcher Farbe der Text innerhalb eines Widgets, wie z. B. eines tk.Entry-Textfelds, angezeigt wird. Ähnlich wie bei der Hintergrundfarbe (bg) können benannte Farben (z. B. „black“, „white“) oder hexadezimale Farbcodes (z. B. „#000000“ für schwarz oder „#FFFFFF“ für weiß) verwendet werden. Das Beispiel mit Textfeldern, ergänzt um fg-Angaben zur Definition der Vordergrundfarbe, findet sich auch im Download-Paket (Colours.py).

Interaktive Steuerung: Spielwürfel

Eine weit verbreitete Anwendung der Tkinter-Bibliothek sind Spielprogramme. Diese können hier nicht in aller Tiefe vorgestellt werden. Als Anregung für eigene Entwicklungen soll an dieser Stelle lediglich ein software-basierter Spielwürfel dienen, der sein echtes Gegenstück grafisch simuliert. Es wird wieder ein Canvas-Widget verwendet, um die Würfelpunkte (Augen) darzustellen, und ein Button, um zufällig eine Zahl zwischen 1 und 6 zu „würfeln“. Die Punkte werden entsprechend der gewürfelten Zahl angezeigt (Dice.py):

```
import tkinter as tk import random

# Hauptfenster erstellen
root = tk.Tk() root.title("Würfelsimulation")
root.geometry("300x400")

# Canvas für die Würfelanzeige
canvas = tk.Canvas(root, width=200, height=200, bg="white")
canvas.pack(pady=50)

# Funktion zum Zeichnen der Würfelpunkte
def draw_dice(number): canvas.delete("all")
# Vorherige Punkte löschen
canvas.create_rectangle(10, 10, 190, 190, outline="black", width=2)
# Würfelrahmen
# Positionen der Punkte (Mitte = 100, 100)
positions = {
1: [(100, 100)], # Mitte
2: [(50, 50), (150, 150)], # Diagonale
3: [(50, 50), (100, 100), (150, 150)], # Diagonale + Mitte
4: [(50, 50), (50, 150), (150, 50), (150, 150)], # Ecken
5: [(50, 50), (50, 150), (150, 50), (150, 150), (100, 100)], # Ecken + Mitte
6: [(50, 50), (50, 100), (50, 150), (150, 50), (150, 100), (150, 150)] # 3x3 ohne Mitte
}
```

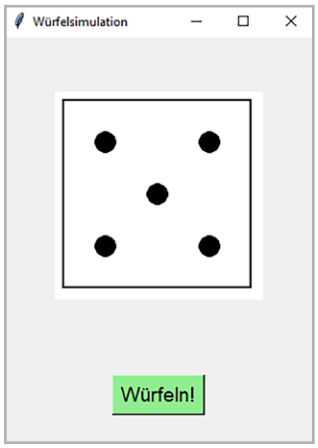

Bild 4: Spielwürfel mit Tkinter

```
# Punkte zeichnen
for pos in positions[number]:
  x, y = pos canvas.create_oval(x-10, y-10, x+10, y+10, fill="black")

# Funktion zum Würfeln
  def roll_dice():
    number = random.randint(1, 6) # Zufällige Zahl von 1 bis 6
    draw_dice(number) # Würfel neu zeichnen
    result_label.config(text=f"Gewürfelt: {number}") # Ergebnis anzeigen

# Button zum Würfeln
roll_button=tk.Button(root,text="Würfeln!",command=roll_dice,font=("Arial",14),bg="lightgreen") roll_button.pack(pady=20)

# Label für das Ergebnis
result_label = tk.Label(root, text="Gewürfelt: -", font=("Arial", 12)) result_label.pack(pady=10)

# Initialen Würfel zeichnen (z. B. 1)
draw_dice(1)

# Hauptschleife starten
root.mainloop()
```

Das Programm liefert die Ausgabe wie in Bild 4 zu sehen. Durch Klicken auf das Feld „Würfeln!“ erscheint eine neue zufällige Augenzahl.

Im Programm wird zunächst ein Fenster mit 300 x 400 Pixel erstellt. Ein Canvas (200 x 200 Pixel) dient als Zeichenfläche für die Würfelaugen.

In der draw_dice-Funktion wird ein quadratischer Rahmen für den Würfel erstellt. Die Positionen der Punkte sind in einem Array (positions) für jede Zahl (1–6) definiert:

- 1: ein Punkt in der Mitte
- 2: zwei Punkte diagonal gegenüber
- 3: drei Punkte diagonal inklusive Mitte
- 4: vier Punkte in den Ecken
- 5: vier Ecken plus Mitte
- 6: sechs Punkte in zwei Reihen (ohne Mitte)

Für jede Position wird ein schwarzer Kreis (create_oval) gezeichnet.

Die roll_dice-Funktion erzeugt eine Zufallszahl zwischen 1 und 6 über die random.randint-Routine.

Anschließend wird die draw_dice-Funktion mit dieser Zahl aufgerufen, und das Ergebnis im Label angezeigt.

Das Programm verwendet die folgenden GUI-Elemente:

- Ein Button (Würfeln!) löst das Würfeln aus.
- Der Button ist grün (bg="lightgreen"), um ihn hervorzuheben.
- Ein Label zeigt die gewürfelte Zahl an.

Für den Initialzustand wird beim Start ein Würfel mit der Zahl 1 gezeichnet, damit das Canvas nicht einfach nur leer ist, sondern wie bei einem echten Würfel aussieht.

Raspberry Pi als Taschenrechner

Dass man mit Tkinter auch ernsthaftere Anwendungen leicht programmieren kann, zeigt das folgende Beispiel: Es liefert einen vollständig funktionsfähigen Taschenrechner als grafische Emulation (Calculator.py):

```
import tkinter as tk

def calculate():
    try:
        expression = entry.get()
        result = eval(expression)
        entry.delete(0, tk.END)
        entry.insert(tk.END, str(result))
    except Exception as e:
        entry.delete(0, tk.END)
        entry.insert(tk.END, "Error")

def clear():
    entry.delete(0, tk.END)

root = tk.Tk()
root.title("Einfacher Taschenrechner")

entry = tk.Entry(root, width=30, borderwidth=5)
entry.grid(row=0, column=0, columnspan=4, padx=10, pady=10)

buttons = [
    "7", "8", "9", "/",
    "4", "5", "6", "*",
    "1", "2", "3", "-",
    "0", ".", "=", "+"
]

row_val = 1
col_val = 0
```

```
for button in buttons:
    tk.Button(root, text=button, padx=20, pady=20,
              command=lambda b=button: calculate() if b == "=" else entry.insert(tk.END, b) if b != "C" else clear()).
grid(row=row_val, column=col_val)
    col_val += 1
    if col_val > 3:
        col_val = 0
        row_val += 1

tk.Button(root, text="C", padx=20, pady=20, command=clear).grid(row=row_val, column=col_val)

root.mainloop()
```

Nach dem Import von Tkinter wird die Funktion „calculate" erstellt. Diese wird aufgerufen, wenn der Benutzer auf das „="-Zeichen klickt. Sie liest den Ausdruck aus dem Eingabefeld, wertet ihn mit eval() aus und zeigt das Ergebnis an. Falls ein Fehler auftritt, wird „Error" angezeigt. Zum Erstellen des Eingabefelds wird eine Liste von Tasten definiert, über die der Benutzer mathematische Ausdrücke eingeben kann:

```
buttons = [
    "7", "8", "9", "/",
    "4", "5", "6", "*",
    "1", "2", "3", "-",
    "0", ".", "=", "+"
]
```

Über die Funktion „for button in buttons" wird jede Taste in einem Raster im Fenster platziert. Die command-Option der Tasten ruft entweder die calculate()-Funktion auf (wenn „=" gedrückt wird), fügt den Tastenwert in das Eingabefeld ein oder löscht das Eingabefeld (wenn „C" gedrückt wird).

Mit root.mainloop() startet die Hauptschleife der Anwendung, welche dann auf Benutzerinteraktionen wartet. Bild 5 zeigt, wie der fertige Taschenrechner auf dem Bildschirm dargestellt wird.

Bild 5: Der Raspberry Pi wird mit Tkinter zum voll funktionsfähigen Taschenrechner.

Dynamische Grafiken und Uhren

Aber nicht nur statische oder interaktive Grafiken lassen sich mit Tkinter leicht umsetzen, auch dynamische, sich selbstständig aktualisierende Grafiken, sind problemlos gestaltbar. Ein schönes Beispiel hierfür sind programmierte Uhren. Der einfachste Fall einer Digitaluhr soll hier nicht betrachtet werden. Stattdessen soll eine wesentlich interessantere quasi-analoge Uhr auf dem Bildschirm erscheinen. Das folgende Programm erledigt diese Aufgabe:

```
import tkinter as tk
from math import sin, cos, radians
import time

class AnalogClock:
    def __init__(self, root):
        self.root = root
        self.root.title("Analog Clock")
        self.canvas = tk.Canvas(root, width=400, height=400, bg='white')
        self.canvas.pack()
        self.MX = 200
        self.MY = 200
        self.R = 190
        self.draw_clock_face()
        self.s1 = -1
        self.update_clock()
```

```
    def draw_clock_face(self):
        for i in range(60):
            x, y = self.get_point(self.R, i)
            size = 4 if i % 5 == 0 else 2
            self.canvas.create_oval(x-size, y-size, x+size, y+size, fill='black')

    def get_point(self, length, angle):
        w1 = radians(angle * 6 - 90)
        x1 = self.MX + length * cos(w1)
        y1 = self.MY + length * sin(w1)
        return (x1, y1)

    def update_clock(self):
        zeit = time.localtime()
        s = zeit.tm_sec
        m = zeit.tm_min
        h = zeit.tm_hour

        if h > 12:
            h = h - 12
        hm = (h + m / 60.0) * 5

        if s != self.s1:
            self.canvas.delete("hands")
            x, y = self.get_point(120, hm)
            self.canvas.create_line(self.MX, self.MY, x, y, width=6, fill='black', tags="hands")
            x, y = self.get_point(170, m)
            self.canvas.create_line(self.MX, self.MY, x, y, width=4, fill='black', tags="hands")

            x, y = self.get_point(180, s)
            self.canvas.create_line(self.MX, self.MY, x, y, width=2, fill='red', tags="hands")

            self.s1 = s
            self.root.title("Aktuelle Zeit: " + time.asctime())
        self.root.after(1000, self.update_clock)

root = tk.Tk()
clock = AnalogClock(root)
root.bind('<Escape>', lambda e: root.quit())
root.mainloop()
```

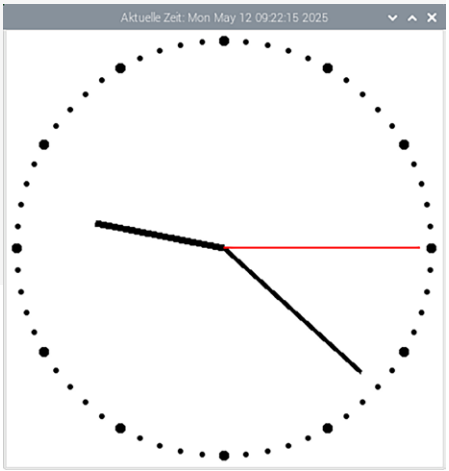

Bild 6: Tkinter zaubert eine ansprechende Analoguhr auf den Bildschirm.

Die Analoguhr zeigt die aktuelle Zeit als klassische „Bahnhofsuhr“ an. Es wird ein Tkinter-Fenster mit einem 400×400 Pixel großen Canvas verwendet, auf dem das Zifferblatt gezeichnet wird. Der Mittelpunkt der Uhr liegt auf den Koordinaten (200, 200 – self.MX = 200 / self.MY = 200), der Radius des Zifferblatts beträgt 190 Pixel (self.R = 190), um einen kleinen Rand zu lassen.

Zunächst wird das Zifferblatt erstellt, indem kleine Kreise für die Minuten und größere Kreise für die Stunden (alle 5 Minuten) gezeichnet werden (draw_clock_face). Diese Kreise werden entlang des Umfangs der Uhr platziert, wobei ihre Positionen mit trigonometrischen Funktionen berechnet werden. Konkret wird für jeden Punkt ein Winkel in Grad berechnet (6 Grad pro Minute, da 360° ÷ 60 = 6°).
Für die Zeiger der Uhr wird die aktuelle Zeit mit time.localtime() abgerufen, um Sekunden, Minuten und Stunden zu erhalten. Die Stunden werden auf ein 12-Stunden-Format reduziert (d. h. 13 Uhr wird zu 1 Uhr usw.).

Dann werden die Winkel für Minuten und Sekunden berechnet, wobei jede Minute und Sekunde 6 Grad entspricht. Mit diesen Winkeln werden die Endpunkte der Zeiger wieder mittels sinus und cosinus berechnet, um ihre Positionen auf dem Kreis zu bestimmen. Die Zeiger erhalten die folgenden Eigenschaften:

- Der Stundenzeiger ist kurz (120 Pixel), dick (6 Pixel) und schwarz.
- Der Minutenzeiger mittellang (170 Pixel), mitteldick (4 Pixel) und schwarz.
- Der Sekundenzeiger ist lang (180 Pixel), dünn (2 Pixel) und rot.

Dadurch sind alle Zeiger optisch gut zu unterscheiden. Die Aktualisierung der Uhr erfolgt jede Sekunde. Das Programm prüft, ob sich die Sekunden seit der letzten Aktualisierung geändert haben.

Anschließend werden die neuen Positionen der Zeiger berechnet und als Linien vom Mittelpunkt zu den berechneten Endpunkten gezeichnet (Bild 6).

Aktive Hardwaresteuerung

Eine der wichtigsten Anwendungen von Tkinter ist die aktive Steuerung von Hardwarekomponenten. Um beispielsweise eine LED zu schalten, benötigt man lediglich eine Kombination aus einer grafischen Benutzeroberfläche (GUI) mit Tkinter und einer Schnittstelle zur Hardware, z. B. über die GPIO-Pins eines Raspberry Pi.

Die LED kann z. B. an GPIO 17 des Raspberry Pi angeschlossen werden. Bild 7 zeigt einen entsprechenden Aufbau mit einem LED-Modul (alternativ kann auch eine einfache LED mit einem passenden Vorwiderstand verwendet werden).

Mit Tkinter wird dann eine GUI erstellt, die passende Schaltflächen („Buttons“) enthält. Diese Buttons rufen Funktionen auf, die den Zustand der LED über die GPIO-Pins ändern, also ein- oder ausschalten. Ein Programm, das eine LED an GPIO 17 mit zwei Buttons („LED an“ und „LED aus“) steuert, kann so aussehen (LEDcontrol.py):

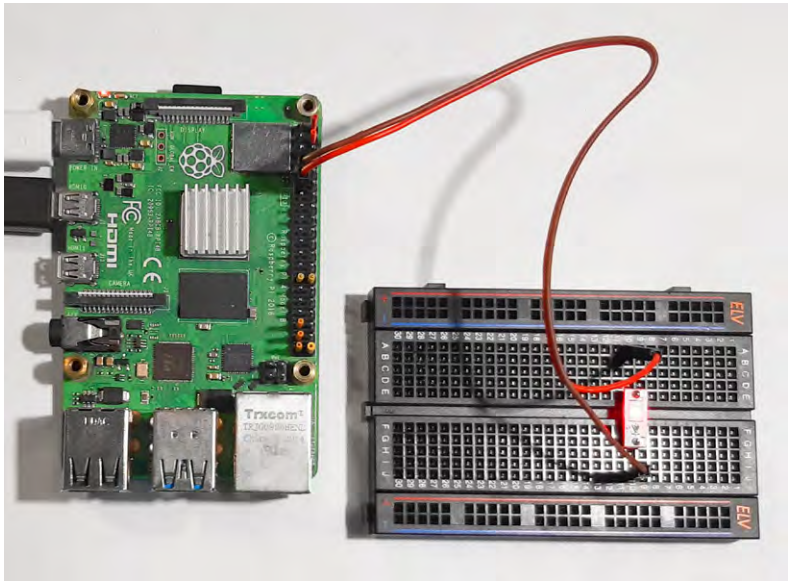

Bild 7: Raspberry Pi mit LED an Pin 17

```
import tkinter as tk
import RPi.GPIO as GPIO
LED_PIN = 17
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED_PIN, GPIO.OUT)
GPIO.output(LED_PIN, GPIO.LOW)

def led_an():
    GPIO.output(LED_PIN, GPIO.HIGH)
    status_label.config(text="LED ist AN")

def led_aus():
    GPIO.output(LED_PIN, GPIO.LOW)
    status_label.config(text="LED ist AUS")

root = tk.Tk()
root.title("LED-Steuerung")
root.geometry("300x200")

status_label = tk.Label(root, text="LED ist AUS", font=("Arial", 14))
status_label.pack(pady=20)

an_button = tk.Button(root, text="LED an", command=led_an, bg="lightgreen", font=("Arial", 12))
an_button.pack(pady=10)

aus_button = tk.Button(root, text="LED aus", command=led_aus, bg="lightcoral", font=("Arial", 12))
aus_button.pack(pady=10)

def cleanup():
    GPIO.cleanup()
    root.destroy()

root.protocol("WM_DELETE_WINDOW", cleanup)
root.mainloop()
```

Im Code erfolgt zunächst die GPIO-Initialisierung. Dabei wird mit

```
GPIO.setmode(GPIO.BCM)
```

die BCM-Nummerierung der Pins festgelegt, sodass GPIO-Nummern statt physischer Pin-Nummern verwendet werden. Anschließend definiert

```
GPIO.setup(LED_PIN, GPIO.OUT)
```

den gewählten Pin als Ausgang, und GPIO.output(LED_PIN, GPIO.LOW) schaltet die LED initial aus.

Danach kommen die Funktionen ins Spiel: Die erste Funktion

```
led_an()
```

schaltet die LED durch GPIO.HIGH ein und aktualisiert gleichzeitig das Label mit dem aktuellen Status, während

```
led_aus()
```

die LED mit GPIO.LOW ausschaltet und ebenfalls das Label anpasst.

Die GUI-Elemente werden durch

```
tk.Label und tk.Button
```

realisiert, wobei das Label den aktuellen Status der LED anzeigt und zwei Buttons mit den Funktionen led_an und led_aus ausgestattet sind. Hintergrundfarben (bg) sorgen für visuelle Unterscheidbarkeit – grün für „an" und rosa für „aus".

Abschließend kümmert sich der Clean-up-Prozess darum, dass mit GPIO.cleanup() die GPIO-Pins zurückgesetzt werden, sobald das Fenster geschlossen wird, um unerwünschte Zustände zu vermeiden. Dafür sorgt root.protocol, indem es sicherstellt, dass die cleanup()-Funktion beim Schließen des Fensters aufgerufen wird. Nach dem Starten des Programms sieht das Steuerungsfenster so aus wie in Bild 8.

Bild 8: Kontrollfenster zum Steuern der LED

Die angeschlossene LED kann nun über ein eigenes Fenster gesteuert werden. Im Bedarfsfall sind natürlich auch mehrere Ports nutzbar. Falls passende Relaissysteme zur Verfügung stehen, können über diese auch Verbraucher mit höheren Leistungen via Tkinter gesteuert werden.

Automatisierung im Griff: Motorsteuerung

Um mit Tkinter die Drehzahl eines kleinen DC-Motors zu steuern, kann eine grafische Benutzeroberfläche (GUI) erstellt werden, die es ermöglicht, den Motor interaktiv zu kontrollieren.

Die Hardwaregrundlage bildet ein Raspberry Pi, der über einen Transistor (z. B. NPN-Transistormodul oder einen Einzeltransistor wie den BC547) einen kleinen DC-Motor ansteuert. Die Transistorstufe ist erforderlich, da der Motor mehr Strom benötigt, als die GPIO-Pins direkt liefern können.

Tkinter wird wieder verwendet, um eine GUI mit einem virtuellen Schieberegler („Slider") zu erstellen. Die Drehzahl des Motors wird über PWM gesteuert, wobei der Duty Cycle (Tastverhältnis) die Geschwindigkeit bestimmt: 0 % = aus, 100 % = volle Drehzahl.

Im folgenden Programmbeispiel, wird der Schieberegler verwendet, um die Drehzahl des DC-Motors über GPIO 17 zu steuern (AnalogSliderControl.py):

```
import tkinter as tk
from tkinter import ttk
import RPi.GPIO as GPIO

# GPIO-Setup
MOTOR_PIN = 17  # GPIO-Pin für den Motor
GPIO.setmode(GPIO.BCM)  # BCM-Nummerierung
GPIO.setup(MOTOR_PIN, GPIO.OUT)  # Pin als Ausgang
pwm = GPIO.PWM(MOTOR_PIN, 100)  # PWM mit 100 Hz Frequenz
pwm.start(0)  # Start mit 0 % Duty Cycle (Motor aus)

# Funktion zur Drehzahlsteuerung
def update_drehzahl(value):
    duty_cycle = float(value)  # Wert von 0 bis 100
    pwm.ChangeDutyCycle(duty_cycle)  # PWM anpassen
    status_label.config(text=f"Drehzahl: {int(duty_cycle)}%")

# GUI erstellen
root = tk.Tk()
root.title("DC-Motor Drehzahlsteuerung")
root.geometry("300x200")

status_label = tk.Label(root, text="Drehzahl: 0%", font=("Arial", 14))
status_label.pack(pady=10)
```

```
# Rahmen für Schieberegler und Werte
frame = tk.Frame(root)
frame.pack(pady=10, padx=20, fill="x")

# Label für 0
label_min = tk.Label(frame, text="0", font=("Arial", 12))
label_min.pack(side="left")

# Schieberegler für Drehzahl (0 bis 100)
slider = ttk.Scale(frame, from_=0, to=100, orient="horizontal", command=update_drehzahl)
slider.pack(side="left", expand=True, fill="x")

# Label für 100
label_max = tk.Label(frame, text="100", font=("Arial", 12))
label_max.pack(side="right")

# Cleanup beim Schließen
def cleanup():
    pwm.stop()  # PWM beenden
    GPIO.cleanup()  # GPIO aufräumen
    root.destroy()  # Fenster schließen

root.protocol("WM_DELETE_WINDOW", cleanup)

# Hauptschleife starten
root.mainloop()
```

Zunächst wird im Programm der verwendete GPIO-Pin als Ausgang definiert, und es wird eine PWM-Signalquelle mit einer Frequenz von 100 Hz gestartet. Der Duty Cycle beginnt bei 0 %, sodass der Motor zunächst ausgeschaltet bleibt.

Die Drehzahlsteuerung erfolgt über die Funktion update_drehzahl, die bei jeder Änderung des Schiebereglers aufgerufen wird. Diese Funktion passt den Duty Cycle des PWM-Signals entsprechend dem aktuellen Wert des Sliders im Bereich von 0 bis 100 % an und aktualisiert gleichzeitig die Anzeige der aktuellen Drehzahl in einem Label. Die grafische Benutzeroberfläche besteht aus einem Label, das den prozentualen Wert der Drehzahl anzeigt, sowie aus einem stufenlos verstellbaren Schieberegler zur manuellen Einstellung der Drehzahl.

Nach dem Start des Programms erscheint ein Steuerungsfenster, das die Drehzahl eines Gleichstrommotors regelt (Bild 9).

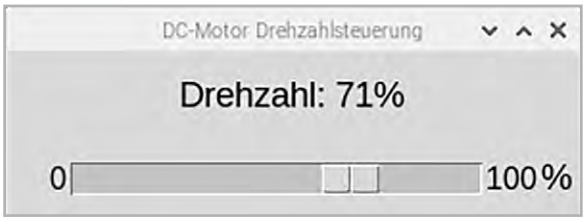

Bild 9: Steuerungsfenster für einen Gleichstrommotor

Beim Schließen des Fensters werden PWM und GPIO wieder durch einen Clean-up-Befehl zurückgesetzt, um die Hardware in einen definierten Zustand zu versetzen.

Für den Hardwareanschluss gibt es zwei Möglichkeiten: Eine einfache Variante nutzt einen NPN-Transistor (z. B. mit dem Kollektor am Minuspol des Motors, dem Emitter an GND und der Basis über einen 1-kΩ-Widerstand an GPIO 17). Der Pluspol des Motors wird dabei direkt an 3,3 V oder 5 V angeschlossen – je nach Spezifikation des Motors. Eine Freilaufdiode (z. B. 1N4007), die antiparallel zum Motor geschaltet ist, schützt vor Spannungsspitzen beim Abschalten (s. Bild 10).

Optional lässt sich das Projekt um verschiedene Funktionen erweitern: Beispielsweise könnten zusätzliche Buttons für „An" (100 % Duty Cycle) und „Aus" (0 % Duty Cycle) integriert werden. Mit einem H-Brücken-Treiber wie dem L298N und einem zweiten GPIO-Pin wäre auch eine Richtungssteuerung des Motors möglich. Darüber hinaus könnte ein Canvas-Element genutzt werden, um die aktuelle Drehzahl grafisch – etwa als Tachometer – darzustellen.

Man kann zunächst testweise die LED wie in Bild 7 anschließen. Die Leuchtdiode kann nun nicht nur ein- und ausgeschaltet, sondern auch stufenlos in ihrer Helligkeit verändert werden. Auf einem Oszilloskop kann der Spannungsverlauf der Pulsweitenmodulation sichtbar gemacht werden. Bild 11 zeigt ein Beispiel für eine mittlere Motorleistung.

Idealerweise kommt für den Motor eine eigene Stromversorgung zum Einsatz. Die direkte Ansteuerung eines kleinen DC-Motors über die 5-V-Pins eines Raspberry Pi ist zwar prinzipiell möglich, aber mit mehreren Nachteilen verbunden:

- **Begrenzte Stromstärke:**
  Die 5-V-Pins des Raspberry Pi liefern nur begrenzten Strom (typisch ~500 mA, abhängig von der Stromversorgung). Ein DC-Motor kann jedoch mehr Strom ziehen, was zu Spannungsabfällen oder Schäden am Raspberry Pi führen kann.
- **Kein Schutzmechanismus:**
  DC-Motoren erzeugen beim Ein-/Ausschalten Spannungsspitzen (Rückstrom), die den Raspberry Pi im Betrieb stören oder sogar beschädigen können.

Für eine sichere und flexible Motorsteuerung ist für die Nutzung eines GPIO-Pins mit PWM daher eine externe Spannungsquelle empfehlenswert, um Probleme oder sogar Schäden sicher zu vermeiden.

Bild 12 zeigt eine Aufbauvorschlag für eine Motorsteuerung mit externer Spannungsquelle und einem Transistormodul.

Mit dieser Schaltung kann die Drehzahl eines DC-Motors interaktiv und präzise über eine Tkinter-GUI gesteuert werden. Dabei ist lediglich zu beachten, dass der Transistor für den Motorstrom ausreichend dimensioniert ist. Der im Modul verwendete BC846 kann beispielsweise für bis zu 100 mA eingesetzt werden. Das Diodenmodul sorgt dafür, dass eventuelle induktive Spannungsspitzen des Motors keine Schäden verursachen.

Ergänzungen und Übungen

Erweiterungsmöglichkeiten zum Spielwürfel

- **Animation:** Wie könnte man eine kurze Animation (z. B. schnelles Wechseln der Augenzahlen) einsetzen um den Würfel vor dem endgültigen Ergebnis „ausrollen" zu lassen?
- **Farben:** Würfelrahmen oder Punkte könnten farbig gestaltet werden (z. B. fill="red" für Punkte).
- **Größe:** Könnte man den Würfel größer oder kleiner skalieren, z. B. indem die Canvas-Größe und Punktkoordinaten angepasst werden?

Erweiterungsmöglichkeiten zum Taschenrechner:

- Wie könnte man wissenschaftliche Funktionen (Sinus, Cosinus, Quadratwurzel etc.) einbauen?
- Wie kann man den einfachen Taschenrechner mit Tkinter um grafische Funktionen erweitern?

Erweiterungsmöglichkeiten zur Motorsteuerung:

- Was ist zu beachten, wenn man leistungsstärkere Motoren betreiben will?
- Wie könnte man die Steuerung auf Vorwärts- und Rückwärtslauf des Motors erweitern?

Zusammenfassung und Ausblick

In diesem Artikel wurden die interaktiven grafischen Darstellungsmöglichkeiten mit Tkinter ausführlich untersucht. Neben rein softwaretechnischen Anwendungen wie Uhren und Taschenrechnern kamen auch Hardwareansteuerungen für LEDs oder DC-Motoren zum Einsatz.

Im nächsten Beitrag wird es um fortgeschrittenere Anwendungen gehen. Insbesondere sollen dann analoge Messwerte anschaulich über virtuelle Messinstrumente dargestellt werden.

Als Anwendungen kommen verschiedene Bereiche wie Wetterstationen, Motorüberwachung im Kfz-Bereich, IoT-Projekte oder auch Hausautomatisierungen infrage. ELV

Material

Raspberry Pi mit Netzteil
Breadboard und Jumperkabel
LED mit Vorwiderstand oder LED-Modul aus PAD-Set
Kleiner DC-Motor
PAD-Transistormodul
Dioden und Widerstand aus einem PAD-Set

Zum Download-Paket

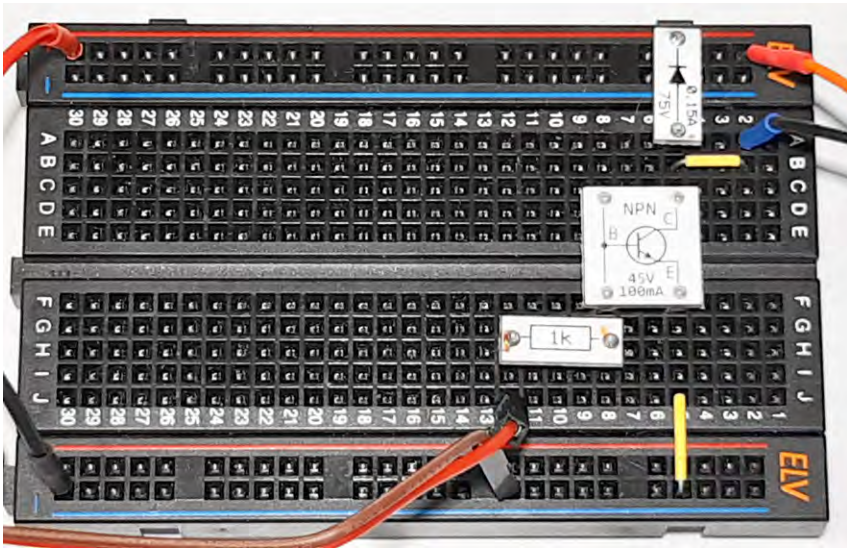

Bild 10: Detailaufbau zur Motorsteuerung

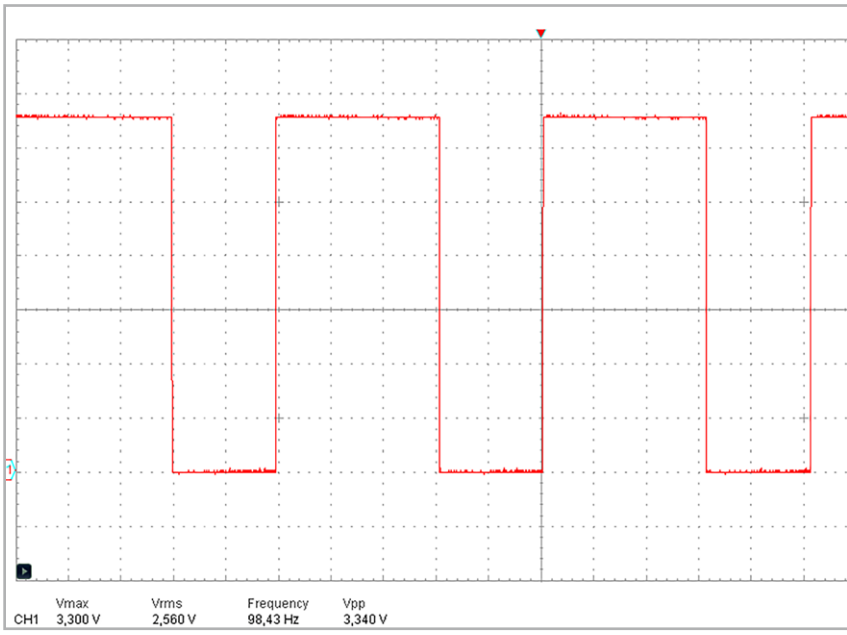

Bild 11: PWM-Signal des Steuerprogramms

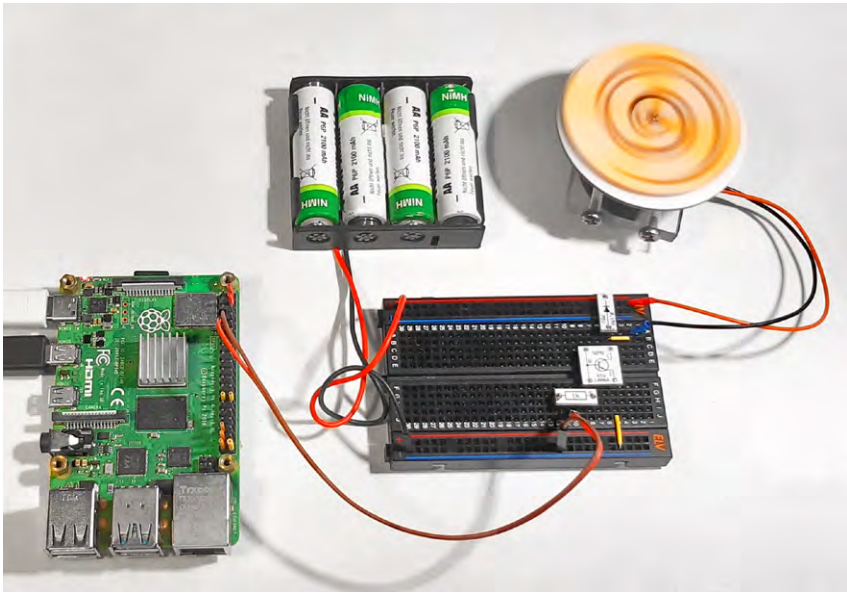

Bild 12: Motorsteuerung für den Raspberry Pi