

Bikelight

Farid Akbar Tabrizi, Jakob Degmair & Ruben Haag



Inhaltsverzeichnis

1 Einleitung	3
2 Problemstellung	3
3 Vorgehensweise, Materialien und Methode	3
3.1 Grundidee	3
3.2 Physikalische Beschreibung	3
3.3 Welche Mindestgeschwindigkeit wird gebraucht?	5
3.4 Ansteuerung des Led Streifens	5
3.5 Programm	6
3.6 Aufbau und Montage	11
3.6.1 Installation des Raspberry Pi's	11
3.6.2 Aufbau und Funktion des ersten Tests mit dem LED-Streifen	12
3.7 Welche Mindestgeschwindigkeit wird gebraucht?	12
4 Ergebnis	13
4.1 Ergebnisdiskussion	13
4.2 Geplante Verbesserungen	14
5 Zusammenfassung	14
6 Anhang	14
6.1 Quellen- und Literaturverzeichnis	14
6.2 Danksagungen	16
6.3 Kostenkalkulation und Materialliste	17

1 Einleitung

In Deutschland gibt es nach Daten des Zweirad-Industrie-Verbands¹ (ZIV) 72 Millionen Fahrräder und 98% der Deutschen können nach eigenen Angaben Fahrrad fahren¹. Trotzdem würden nur 38% der Deutschen mehrmals in der Woche ihr Rad nutzen¹. Dies liegt nach eigener Erfahrung vor allem daran, dass man sich im Stadtverkehr in einem Auto sicherer fühlt, als auf einem Fahrrad. Natürlich kann man sich geeignete Kleidungsstücke, die die eigene Sichtbarkeit erhöhen, kaufen, aber diese sind meistens lästig und sind wenig modisch. Um diese Kleidungsstücke zu vermeiden, haben wir nach einer technisch besseren und interaktiveren Lösungsmöglichkeit gesucht. Wir dachten uns, dass die optimale Lösung ein Leuchtmittel zwischen den Speichen wäre und fanden bei unserer Suche ähnliche Lösungsansätze. Bei dem Besuch der Website der Firma Monkeylectrics² haben wir etwas ähnliches gefunden: ein Speichendisplay, das jedoch mit rund 2000€² nur wenige Fahrradfahrer ansprechen würde. Da haben wir uns gefragt, wie ein solches Speichendisplay kostengünstiger und besser herzustellen ist. Die Grundidee war, dass wir mit einem einfachen Raspberry Pi einen farbigen LED-Streifen ansteuern. Danach haben wir uns einen 1m langen LED-Streifen gekauft, bei dem man jede Diode einzeln mit dem Raspberry Pi ansteuern kann. Der Streifen besitzt dabei eine Dichte von 144 LEDs je Meter und hat somit ausreichend viele LEDs, um ein Bild in ansprechender Auflösung darzustellen. Wir haben uns erhofft, eine interaktive Variante zur Warnweste zu finden, indem zum Beispiel beim Überqueren einer Kreuzung vom Speichendisplay ein Stoppschild angezeigt wird, wodurch ein von der Seite kommender Autofahrer frühzeitig gewarnt wird. Das Speichendisplay und die Software sollen als Selbstbausatz für Fahrradfahrer aller Altersklassen zur Verfügung gestellt werden.

2 Problemstellung

Die Sicherheit von Fahrradfahrern im Straßenverkehr soll durch ein interaktives Speichendisplay für Fahrräder erhöht werden. Dabei sollen Bilder und Informationen während der Fahrt mit der Außenwelt geteilt werden.

3 Vorgehensweise, Materialien und Methode

3.1 Grundidee

Unsere Idee, um das Bild anzeigen zu können, ist, dass wir an einem Speichenrad mehrere LED-Streifen befestigen, die dann jeweils den Teil des Bildes darstellen, an dem sie sich momentan befinden. Die LED-Streifen rotieren dann mit dem Rad, dies führt dazu, dass der Betrachter, aufgrund der Trägheit der Augen, ein vollständiges Bild wahrnimmt. Diesen Sachverhalt haben wir physikalisch folgendermaßen beschrieben:

3.2 Physikalische Beschreibung

Um für jeden Pixel die aktuellen Koordinaten zu berechnen, werden verschiedene physikalische Grundlagenrechnungen verwendet. Als erstes brauchen wir die Umlaufzeit T . Diese berechnen wir, indem die Zeitdifferenz der Umdrehung zwischen der Anfangszeit t_{Anfang} , zu Beginn der Umdrehung, und der Endzeit t_{Ende} , nach einer vollständigen Umdrehung mit einem Magnetschalter, ermittelt wird:

$$T = t_{Ende} - t_{Anfang} \quad (1)$$

Die Winkelgeschwindigkeit ω des Reifens wird nun über die Winkeländerung pro Zeit berechnet. Da die Zeit für eine Umdrehung gemessen wird, entspricht die Winkeländerung in Radian 2π. Die Gleichung lautet demnach:

$$\omega = \frac{2\pi}{T} \quad (2)$$

Für die während der Umdrehung laufenden Berechnungen brauchen wir die Zeit, die seit Beginn der Umdrehung vergangen ist. Diese berechnen wir über die Zeitdifferenz Δt zwischen der Anfangszeit t_{Anfang} der Umdrehung und der momentanen Zeit t_{Momentan} (für die genauere Beschreibung von t_{Momentan} siehe 3.5 Programmabschnitt 6):

$$\Delta t = t_{\text{Momentan}} - t_{\text{Anfang}} \quad (3)$$

Um nun damit den aktuellen Drehwinkel zu berechnen, stellen wir die Gleichung für die Winkelgeschwindigkeit (Gleichung 2), diesmal aber für den Winkel α und die Zeitdifferenz Δt , nach α um.

$$\alpha = \omega \cdot \Delta t \quad (4)$$

Damit jede LED eine Position bei einem bestimmten Winkel hat, haben wir die Bewegung in einem Koordinatensystem dargestellt. Um nun die Koordinaten einer LED zu berechnen, nutzen wir die beiden Grundgleichungen des Einheitskreises $a_1 = \cos(\alpha)$ und $b_1 = \sin(\alpha)$ in Polarkoordinaten. Da diese jedoch für den Radius von 1 Längeneinheit gelten, multiplizieren wir diese noch mit dem Radius der anzusteuernden LED:

$$a = \cos(\alpha) \cdot r \quad (5)$$

$$b = \sin(\alpha) \cdot r \quad (6)$$

Der Mittelpunkt des Koordinatensystems des Kreises, auf dem sich eine einzelne LED bewegt, ist noch auf dem Punkt A(0|0), wie auf Abb. 2 zu sehen ist.

Das Problem ist nun, dass die Pixel eines Bildes nur positive Koordinaten haben. Dies lässt sich dadurch korrigieren, dass man zu den x-Koordinaten die Hälfte der Breite des Bildes und zu den y-Koordinaten die Hälfte der Höhe des Bildes addiert:

$$x = \cos(\alpha) \cdot r + \frac{\text{Breite des Bildes}}{2} \quad (7)$$

$$y = \sin(\alpha) \cdot r + \frac{\text{Höhe des Bildes}}{2} \quad (8)$$

Wir addieren die Hälfte der Höhe des Bildes und die Hälfte der Breite des Bildes, da dies die Mindestgröße ist, damit sich der Graph im ersten Quadranten befindet. Dadurch verändert sich der Mittelpunkt, wie auf Abb. 2 zu sehen ist, auf den Punkt M(35|35).

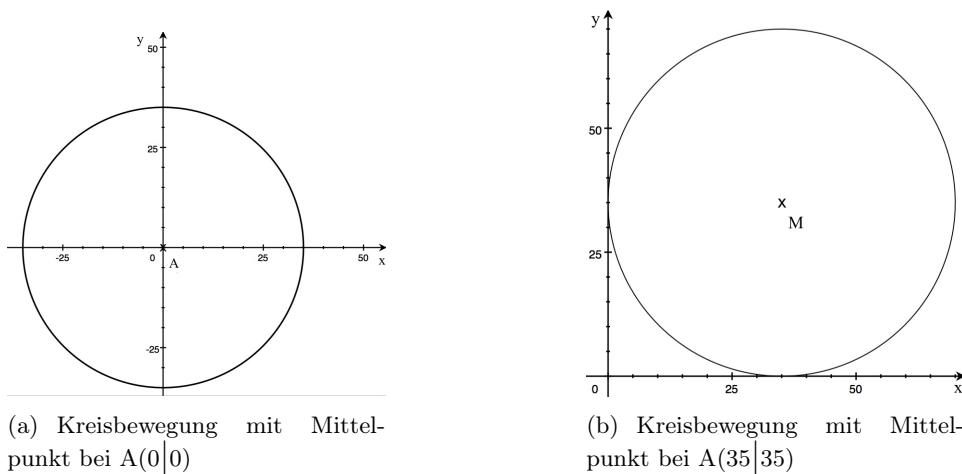


Abbildung 1: Verschiebung des Kreiszentrums

Diese physikalischen Aspekte haben wir in unserem Programm folgendermaßen umgesetzt:

3.3 Welche Mindestgeschwindigkeit wird gebraucht?

Das menschliche Auge nimmt Bilder in einer Frequenz f_{Auge} von 10 bis 12Hz auf¹⁷. Um die Geschwindigkeit bei rund 10Hz zu berechnen, verwenden wir die Gleichung $v = \frac{\Delta s}{\Delta t}$. Auf die Kreisbewegung angewandt entspräche dies $v = \frac{2\pi r}{T}$ in $\frac{m}{s}$. Da $\frac{1}{T}$ der Frequenz entspricht, der Umrechnungsfaktor von $\frac{m}{s}$ in $\frac{km}{h}$ 3,6 ist und $d = 2 \cdot r$ gilt, erhalten wir die Gleichung $v = \pi \cdot f_{\text{Auge}} \cdot d \cdot 3,6$. Durch das Anbringen der vier Streifen am Rad, benötigen wir nur ein viertel der ursprüngliche Geschwindigkeit. Demnach gilt:

$$v = \frac{\pi \cdot f_{\text{Auge}} \cdot d \cdot 3,6}{4} \quad (9)$$

Also beträgt die Mindestgeschwindigkeit rund $v \approx 17 \frac{km}{h}$ bei einem 24“ Rad, um ein vollständiges Bild zu erkennen. Die Geschwindigkeit, bei der schon ein Bild zu erkennen ist, ist jedoch kleiner, weil das Gehirn ein Bild selbst vervollständigt. Dem Gehirn genügt es, einige Konturen wahrzunehmen, um ein Bild oder Objekt zu erkennen, da es den Rest aus Erfahrung und Fantasie ergänzt¹⁸.

3.4 Ansteuerung des Led Streifens

Der verwendete LED-Streifen hat die Modus so werden diese an die zweite LED übertragen, die wiederum nur die ersten 24bit für sich selbst übernimmt und den Rest weiterschickt usw..ellnummer ws2812b und wird wie folgt angesteuert¹⁹: Jede Led des LED-Streifens wird durch ein 24bit Signal angesteuert, das mittels Pulsweitenmodulation vom Raspberry Pi auf diesen übertragen wird. Die Farbanteile Grün, Rot und Blau werden je über acht bit angesteuert, so dass man pro pixel 2^8 also 256 und mit allen drei LEDs 16.777.216 verschiedene Farben anzeigen kann. Die Reihenfolge der Farbansteuerung ist in folgendem Diagramm verdeutlicht:



Abbildung 2: Reihenfolge der Bits bei der Ansteuerung einer LED des LED-streifens

Falls nach diesem 24Bit Signal, innerhalb der Taktung¹⁹ noch weitere Signale an den LED-Streifen übertragen werden, so werden diese über den DOUT(siehe Abbildung 4) anschluß an die zweite LED übertragen, die wiederum nur die ersten 24bit für sich selbst übernimmt und den Rest weiterschickt usw..

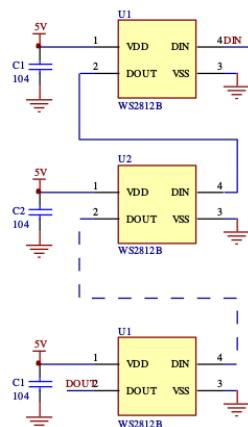


Abbildung 3: Schaltplan des Led-streifens¹⁹

3.5 Programm

Wir haben das Programm in Python 3 geschrieben. Zu Beginn haben wir uns hauptsächlich an dem Programm strandtest.py orientiert, welches in der Adafruit-Neopixel-Library³ enthalten ist. Dieses Programm beinhaltete die Abfolge verschiedener Farben, die der LED- Streifen anzeigt. Anhand dieses Programms konnten wir die Reihenfolge und Darstellung der Farbwerte erschließen, welche in diesem Fall GRB ist.

Programmabschnitt 1:

In unserem fertigen Programm importieren wir anfangs die Module time, math und RPi.GPIO und die Libraries neopixel und PIL, welche wir für das später gebrauchen werden. Daraufhin haben wir verschiedene Variablen definiert, welche das Bild mittels der Funktion `open()` der PIL öffnen, es mit der Funktion `im.load()` laden und uns die Breite und Höhe des Bildes mit `im.size()` ausgeben. Wie man im Prog. 4 später sieht ist es egal, ob das Bild RGB-Werte oder CMYK-Werte besitzt. Um das kommende Program effizienter zu gestalten, haben wir hier schon *breite* und *höhe* als die Hälfte der Breite des Bildes und die Hälfte der Höhe des Bildes definiert. Die im Program benötigten Größen, die Anzahl der LEDs, die Nummer des Pins, der den LED-Streifen ansteuert, die Nummer des Pins, der den Magnetschalter ansteuert, die Frequenz der LEDs, der DMA-Kanal, die Helligkeit und die Anzahl der Streifen, werden daraufhin alle definiert. Zusätzlich haben wir diverse Variablen definiert, welche wir für numerische Schleifen oder zum späteren Definieren benötigen.

```
from time import time, sleep
from math import pi, sin, cos
from PIL.Image import open
from neopixel import Adafruit_NeoPixel, Color
import RPi.GPIO as gp

im = open("/home/pi/Jufo_Bikelight/Release/Ihr_Bild_hier.png")
pix = im.load()
breite, höhe = im.size #die Breite und die Höhe des Bildes wird ausgelesen
breite = breite / 2
höhe = höhe / 2

# LED strip configuration:
LED_COUNT      = 140      # Number of LED pixels.
LED_PIN         = 18       # GPIO pin connected to the pixels (must support PWM!).
MAGNET_PIN     = 17       # Nummer des Magnet Pins
LED_FREQ_HZ    = 800000   # Frequenz der Led's in Hz
LED_DMA         = 5        # DMA Kanal, des Led pins(siehe C code der Library)
LED_BRIGHTNESS = 40       # Set to 0 for darkest and 255 for brightest
LED_INVERT      = False    # Wenn ein transistor zwischengeschaltet ist Aktivieren
ANZAHL_STREIFEN = 4       # Anzahl der verwendeten Led Streifen pro pin
```

Abbildung 4: Programmabschnitt 1

Programmabschnitt 2:

Danach erschaffen wir die Liste *radien*, in der wir 0 fünfunddreißigmal mit einer numerischen Schleife, die wir mittels der Funktion `range()` mit der Untergrenze 0 und der Obergrenze, die der Quotient aus LED_COUNT und ANZAHL_STREIFEN ist, erschaffen, einfügen, da man nicht solch eine Liste in der zweiten numerischen Schleife erschaffen kann, inder man die Radien als einzelne Werte in die Liste einfügt. Der Datentyp des Quotienten ist ein Float. Da die Funktion `range()` nur mit Integer arbeitet, ändern wir den Typen des Quotienten mittels der Funktion `int()` zu einem Integer. Danach erstellen wir eine weitere numerische Schleife für *i* (siehe Prog. 1) und denselben Grenzen und erschaffen damit die Radien. Wir ersetzen die 0 an *i*-ter Stelle durch die Summe aus *i*, 1, und der Variablen *minR*. Die Addition von 1 ist notwendig, da die Aufzählung sonst bei 0 beginne. Die Addition von *minR* ist nötig, da wir die ersten fünf Radien nicht verwenden können, weil sich dort aufgrund der Radnabe keine LEDs befinden. Da wir *i*

im Bereich von 0 bis 35 aufzählen, decken wir alle 35 Radianen ab, die zuerst 0 waren und dann ersetzt werden. Wir haben 35 Radianen, da ein Teilstreifen 35 LEDs besitzt.

```
# erschaffen einer Liste
radian = [0 for x in range(0, int(LED_COUNT/ANZAHL_STREIFEN))]
for i in range(0, int(LED_COUNT/ANZAHL_STREIFEN)): # Erschaffen der Radianen
    radien[i] = int(i+1+minR)
```

Abbildung 5: Programmabschnitt 2

Programmabschnitt 3:

Die zwei Funktionen *line()* und *startPrint()* beziehen sich auf die Herstellung der Startanzeige. Die Funktion *line* erstellt die Abgrenzungen der Startanzeige. Anfangs wird ein leerer String *s* erzeugt. Dieser String erhält dann mit einer numerischen Schleife für *i* (siehe Prog. 1) in dem Bereich von 0 bis *länge*, welcher in der Funktion *startPrint()* gewählt wird, Striche (je nachdem wie *länge* gewählt wird) und wird am Ende der Funktion ausgegeben. Die Funktion *startPrint()* gibt die Startanweisung aus. In der Startanweisung werden die wichtigsten Größen für den Benutzer ausgegeben. Als Parameter für die Funktion *line()* haben wir 50 gewählt, da dies eine geeignete Länge für eine Abgrenzung ist. Am Ende wird die Ausgabe erzeugt, dass man das Programm mittels der Tastenkombination Strg-c beenden kann.

```
# Startanzeige
def line(länge):
    s = ""
    for i in range(0, längen):
        s += "-"
    print(s)

def startPrint():
    str1 = "Led Bikelight"
    str2 = "\nDie Momentanen Einstellungen sind:"
    line(50)
    print(str1)
    line(50)
    print(str2)
    print("")
    print("Anzahl der Led's: " + str(LED_COUNT))
    print("GPIO-Pin: " + str(LED_PIN))
    print("f in Hz: " + str(LED_FREQ_HZ))
    print("DMA Kanal: " + str(LED_DMA))
    print("LED Helligkeit: " + str(LED_BRIGHTNESS))
    print("Invertiertes Signal: " + str(LED_INVERT))
    print("")
    line(50)
    print("Drücke Strg-C zum beenden.")
    line(50)
```

Abbildung 6: Programmabschnitt 3

```

-----
Led Bikelight
-----
Die Momentanen Einstellungen sind:
Anzahl der Led's: 140
GPIO-Pin: 18
f in Hz: 700000
DMA Kanal: 5
LED Helligkeit: 40
Invertiertes Signal: False
-----
Drücke Strg-C zum Beenden.
-----
```

Abbildung 7: Startanzeige

Programmabschnitt 4:

Die Funktion `bildAuslesen()` gibt zu jeder Position der LEDs die x- und y-Koordinaten und die dazugehörigen RGB-Werte im Bild aus. Die x- und die y-Koordinate werden mit den Gleichungen 7 und 8 (siehe Physikalische Beschreibung) berechnet. Da die Funktionen `sin()` und `cos()` auf fünfzehn Stellen genau die Koordinaten ausgegeben, wir sie allerdings nur mit einer Dezimalstelle genau benötigen, werden die Gleichungen 7 und 8 folgendermaßen geändert:

$$x \approx \cos(\alpha) \cdot r + \frac{\text{Breite des Bildes}}{2} \quad (10)$$

$$y \approx \sin(\alpha) \cdot r + \frac{\text{Höhe des Bildes}}{2} \quad (11)$$

Im Programm runden wir sie mit der Funktion `round()` auf die erste Dezimalstelle. Danach werden die RGB Werte an den Koordinaten x und y mit der Funktion `pix[x, y]` (siehe Prog. 1) ausgelesen. Mit Hilfe einer try-except-Anweisung kann unser Programm zwischen Bildern mit RGB-Werten oder CMYK-Werten unterscheiden und mit der Anweisung `return Color(g, r, b)` beziehungsweise `return Color(g, r, b, _)` an den LED- Streifen übergeben. Bei der Variante mit den CMYK-Werten lassen wir die Schwarzweite raus, da wir sie nicht benötigen. Dies geschieht in der Reihenfolge g, r, b (siehe Prog. 1).

```

def bildAuslesen(winkel, rad):
    x = int(round(cos(winkel) * rad + breite)) # Berechnung der X-Koordinate
    y = int(round(sin(winkel) * rad + höhe))   # Berechnung der Y Koordinate

    try:
        r,g,b = pix[x, y]           # Auslesen eines Pixels
    except:
        r,g,b,_ = pix[x, y]

```

Abbildung 8: Programmabschnitt 4

Programmabschnitt 5:

In der Funktion `streifenBedienen()` werden zuerst die Liste mit den Radien `radien`, der LED-Streifen `streifen` (siehe Prog. 6) und die im ersten Programmabschnitt voreingestellte Umlaufzeit T , Zeitabschnitt t und Winkelgeschwindigkeit ω globalisiert, da wir diese über mehrere Funktionen benötigen und die Globalisierung effizienter ist, als sie als Parameter zu übergeben. Die Winkel α , β , γ und δ werden nach Gleichung 4 berechnet. Die Winkelgeschwindigkeit und der Zeitabschnitt werden jedoch in `main()` berechnet. Da die LED-Streifen in einem Winkel von 90° gedreht sind, addieren wir zum vorhin berechneten Winkel 90° beziehungsweise $\frac{\pi}{2}$ hinzu. Weil der Zeitabschnitt t größer wird, wachsen die Winkel an, was die Bewegung der LEDs im Bild simuliert. Dann haben wir n als Anzahl der Pixel des LED-Streifens, u als Anzahl der Pixel auf einen Teilstreifen und M als die Hälfte der Anzahl der Pixel des LED-Streifens definiert. Die erste numerische Schleife zählt bis u hoch, dabei wird i um 1 erweitert und u um 1

verringert. Die Funktion `bildAuslesen()` ermittelt dann für den ersten und dritten Streifen (siehe Anhang Abb. 3) die RGB-Werte. Wir lassen die Radien rückwärts von der Liste `radien` aufzählen, indem wir $u = u - 1$ rechnen. Demnach nimmt bei größer werdendem i u von 35 bis 0 ab und in dieser Reihenfolge werden die Radien aus der Liste `radien` aufgerufen. Die Radien müssen rückwärts aufgerufen werden, da die PIL das Bild vom Zentrum aus ausliest, der erste und dritte LED-Streifen jedoch von außen angeschlossen sind (siehe Anhang Abb. 3). Da u in der ersten numerischen Schleife heruntergezählt wurde, mussten wir u nochmals definieren. Die RGB-Werte werden mit der Nummer der jeweiligen LED der Funktion `streifen.setPixelColor()` übergeben. Wir erhalten für den ersten Streifen die richtige Reihenfolge, da i von 0 bis 35 aufgezählt wird und für den dritten Streifen ebenfalls, weil wir i mit M addieren, weil der dritte Streifen einen halben Streifen vom ersten entfernt ist. Die zweite numerische Schleife zählt von u bis M . Die Funktion `bildAuslesen()` gibt die RGB-Werte des zweiten und vierten LED-Streifens an. Die Radien werden diesmal in der richtigen Reihenfolge aus der Liste entnommen, da diese LED-Streifen von innen angeschlossen sind (siehe Anhang Abb. 3). Anschließend werden die RGB-Werte zusammen mit der Nummer der anzustauenden LED an die Funktion `streifen.setPixelColor()` übergeben. Die Nummerierung der LEDs am zweiten Streifen verläuft von 35 bis 70, da i in der numerischen Schleife hochgezählt wird. Die Nummerierung der LEDs am vierten Streifen geht von 105 bis 140, weil der Streifen 70 LEDs vom zweiten Streifen entfernt war. Diese Aufzählung erreicht man, indem man i mit M addiert. Am Ende von `streifenBedienen()` bringt die Funktion `streifen.show()` die LED-Streifen zum Leuchten.

```
def streifenBedienen(t, w):
    global radien
    global T
    global t
    global streifen
    global w

    alpha = w * t          #Ausrechnen des Winkels in Bogenmaß
    beta = alpha + pi / 2
    gamma = beta + pi / 2
    delta = gamma + pi / 2

    n = streifen.numPixels()
    u = int(n/ANZAHL_STREIFEN)
    M = int(n/2)

    for i in range(u):
        u-= 1
        streifen.setPixelColor(i, bildAuslesen(alpha, radien[u]))
        streifen.setPixelColor(i, bildAuslesen(gamma, radien[u]))

    u = int(n/ANZAHL_STREIFEN)

    for i in range(u, M):
        streifen.setPixelColor(i, bildAuslesen(beta, radien[i-u]))
        streifen.setPixelColor(i + M, bildAuslesen(delta, radien[i-u]))
    streifen.show()
```

Abbildung 9: Programmabschnitt 5

Programmabschnitt 6:

In der Funktion `main()` werden zuerst die benötigten Variablen globalisiert. Dies sind die Umlaufzeit T , der Zeitabschnitt t , die Zeit seit dem Beginn der Umdrehung $t1$, die Liste `radien` mit den Radien, der LED-Streifen `streifen` und die Winkelgeschwindigkeit ω . Danach wird mit der Anweisung `gp.setmode(gp.BCM)` festgelegt, dass GPIO-Nummern verwendet werden. Mit der Anweisung `gp.setwarnings(False)` wird festgelegt, dass keine Warnungen ausgegeben werden, da wir diese nicht benötigen und mit der Anweisung `gp.setmode(MAGNET_PIN, gp.IN)` wird `MAGNET_PIN` (siehe ersten Programmabschnitt) als Eingang festgelegt. Mit der Methode

Adafruit_NeoPixel wird mit den Eigenschaften *LED_COUNT*, *LED_PIN*, *LED_FREQ_HZ*, *LED_DMA*, *LED_INVERT*, *LED_BRIGHTNESS* das Objekt *streifen*, also der LED-Streifen erstellt. Die Funktion *streifen.begin()* initialisiert den LED-Streifen und die Funktion *startPrint()* gibt die Startanzeige aus. In der darauf folgenden while-Schleife wird festgelegt, dass die Zeit seit dem Beginn der Umdrehung gleich der Systemzeit ist. Danach wird die Bedingung formuliert, dass, falls der Magnetschalter *MAGNET_PIN* nicht auslöst, die Winkelgeschwindigkeit ω mit der Gleichung 2 berechnet wird. In dieser if-Anweisung befindet sich eine while-Schleife, welche auslöst, wenn der Magnetschalter nicht auslöst. In dieser while-Schleife wird ein Zeitabschnitt berechnet und da diese while-Schleife mehrere male auslöst werden mehrere Zeitabschnitte berechnet, die man für die Winkelberechnung benötigt (siehe Prog. 5). Ebenfalls wird die Funktion *streifenBedienen()* aufgerufen, welche den LED-Streifen dann zum Leuchten bringt. Wenn der Magnetschalter auslöst, also die if-Anweisung nicht zutrifft, wird die Umlaufzeit nach der Gleichung 1 berechnet. Da wir anfangs (siehe Prog. 1) T und t definiert haben, erhalten wir *UnboundLocalError* nicht.

```

def main():
    global radien
    global streifen
    global T
    global t
    global t1
    global radien
    global streifen
    global w

    T=2

    gp.setmode(gp.BCM)          # Welche Nummern für die Pins verwendet werden
    gp.setwarnings(False)        # Keine Warnungen
    gp.setup(MAGNET_PIN, gp.IN)  # Anschluss

    #Erschaffen des Led-Streifen-Objekts
    streifen = Adafruit_NeoPixel(LED_COUNT, LED_PIN, LED_FREQ_HZ, LED_DMA,
                                LED_INVERT, LED_BRIGHTNESS)

    streifen.begin()           #Initialisieren des LED-Streifens
    startPrint()               #Drucken der Startanzeige

    while True:
        t1 = time()            #Startzeit der Umdrehung t1

        if gp.input(MAGNET_PIN) == False:  #Zeitmessung einmal je Umdrehung

            w = 2 * pi / T  #Berechnen der Aktuellen Winkelgeschwindigkeit

            while gp.input(MAGNET_PIN) == False:
                t = time() - t1           #Ausrechnen der größe des Zeitabschnitts

                streifenBedienen(t, w)
                streifen.show()
            T = time() - t1             #Ausrechnen von T nach T = t2 - t1

```

Abbildung 10: Programmabschnitt 6

Programmabschnitt 7:

Die Funktion *main()* wird aufgerufen, indem eine if-Anweisung auslöst, wenn der Name des Programms *main* ist, was hier der Fall ist.

```

if __name__ == '__main__':

```

Abbildung 11: Programmabschnitt 7

3.6 Aufbau und Montage

3.6.1 Installation des Raspberry Pi's

Als erstes haben wir eine microSD-Karte mit dem Betriebssystem Raspbian⁶ bespielt und haben einen WLAN-USB-Stick von seinem Gehäuse entnommen und nach einem Bild⁷ mit dem Raspberry Pi Zero verbunden. Danach haben wir die SD-Karte in einen Computer gesteckt und haben in die Datei `/etc/wpa_supplicant/wpa_supplicant.conf` Netzwerkdetails hinzugefügt⁸. Danach haben wir den Raspberry Pi an einen Monitor angeschlossen, ihn zum ersten mal gebootet und mit Hilfe des Befehls `raspi-config`⁹ den Zugriff auf den Raspberry Pi durch SSH ermöglicht. Seinen Hostnamen haben wir zu Spokescreen geändert. Danach konnten wir über jeden PC in unserem Heimnetzwerk, auf welchem ein SSH-Client installiert war, auf das Terminal des Raspberry Pis zugreifen. Nun haben wir git, den Github Client für Linux, auf dem Raspberry Pi installiert und unser Github Repository¹⁰ auf den Pi mit dem Befehl `git clone https://github.com/RubenHaag/Jufo_Spokescreen` heruntergeladen. Danach haben wir die Python Image Library⁵, zum Auslesen eines Bildes mit Python, und die Adafruit Neopixel Library³, zum Ansteuern der LED-Streifen installiert. Um nun unsere LED-Streifen zu testen, haben wir uns zuerst einen Schaltplan mit Hilfe des Tools Fritzing¹¹ gezeichnet.

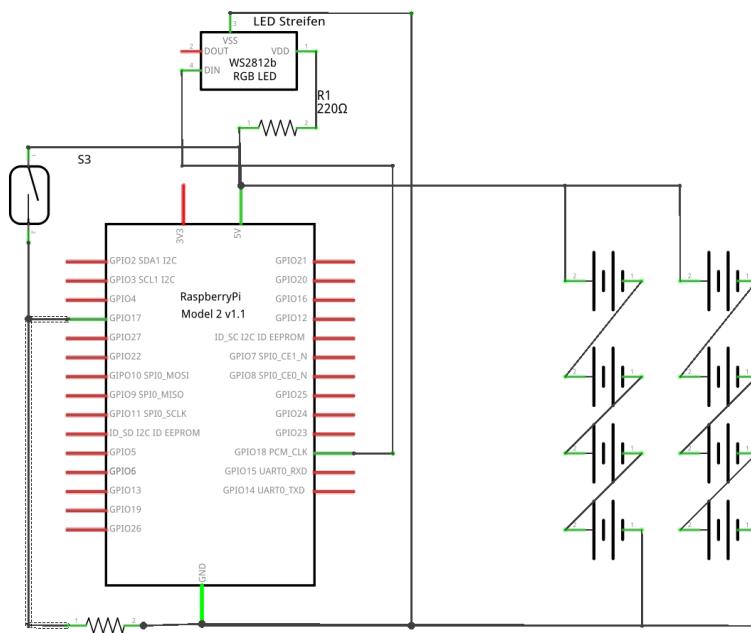


Abbildung 12: Schaltplan

Der Raspberry Pi ist mit einem seiner Ground-Pins mit der Masse verbunden und mit einem der beiden 5-Volt-Pins direkt mit der Spannungsquelle verbunden. Der LED Streifen ist ebenfalls mit dem 5-Volt- und Ground-Pin verbunden und wird über den GPIO-18-Pin des Raspberry Pis angesteuert, da dieser als einer der wenigen die für die Ansteuerung des Streifens essentielle Pulswidtemodulation (PWM) unterstützt. Um die Zeit nun einmal pro Umdrehung zu messen, verwenden wir das Reed-Relais (Magnetschalter) eines Fahrradtachos. Durch diese Relais fließt, wenn er in die Nähe des an der Gabel befestigten Magneten kommt, Strom, da sich die Kontakte im Relais schließen, welcher über den GPIO-17-Pin des Raspberry Pis ausgelesen wird. Zur Ab-

sicherung des Raspberry Pis gegen eventuelle Spannungsschwankungen ist der Magnetschalter zusätzlich noch über einen Widerstand von 7Ω mit dem Ground-Pin verbunden. Die Stromversorgung liefern zwei USB-Powerbanks mit je 5000 mAh. Sie liefern eine konstante Spannung von je 5 V, besitzen eine relativ hohe Energiedichte und sind trotzdem vergleichsweise leicht. Diese sind parallel geschaltet, damit die Spannung den Raspberry Pi nicht überlastet, und stellen die Stromversorgung mit 5 V dar. Die Powerbanks haben wir angeschlossen, indem wir zwei USB Kabel abisoliert und mit Standard-Pin-Anschlüssen versehen haben (Polung siehe Lit. [4]). Diesen Schaltplan haben wir zuerst auf einem Breadboard zusammengebaut, um zu testen, ob die LED-Streifen und der Magnetschalter funktionieren. Als einen ersten Test für den LED-Streifen haben wir das bei der Adafruit NeoPixel Library³ mitgelieferte Programm „Strandtest.py“ verwendet.

3.6.2 Aufbau und Funktion des ersten Tests mit dem LED-Streifen

Für den Magnetschalter haben wir ein Programm geschrieben, um die Zeit zwischen zwei Signalen von diesem zu messen (siehe Programm). Da alle Komponenten ohne Probleme funktionierten, haben wir nun den LED-Streifen in vier Teile à 36 LEDs geteilt und für einen größeren Spielraum haben wir von jedem Streifen eine LED entfernt. Zuerst haben wir versucht, die vier unterschiedlichen LED-Streifen an vier unterschiedlichen GPIO-Pins anzuschließen, bis wir festgestellt haben, dass der verwendete Raspberry Pi 3 und der Raspberry Pi Zero, den wir später anschließen wollten, nur 6 GPIO-Pins haben, die PWM unterstützen. Um auf jeder Seite vier LED-Streifen anzubringen, hätten wir jedoch mindestens 8 Pins benötigt, also haben wir die vier LED-Streifen in Reihe geschaltet und so nur den PWM fähigen GPIO-18-Pin verwendet. Daraufhin haben wir den Schaltplan auf eine Platine übertragen und die vier LED-Streifen durch unterschiedlich lange Kabelstücke miteinander verbunden, um sie später entsprechend auf dem Reifen zu platzieren. Die Platine haben wir nun in eine wasserdichte Verteilerbox eingebaut und mit Kabelbindern an den Speichen befestigt. Nach ersten misslungenen Versuchen mit Kunststoffhalterungen haben wir uns aus Stabilitätsgründen für eine Märklin-Metall-Halterung entschieden, um die Elektronik sicher an dem Speichenrad zu befestigen. Die Halterung haben wir um die Nabe herum in einem rechtwinkligen Kreuz angeordnet. Dann haben wir die LED-Streifen auf feste Kunststoffstreifen geklebt und diese dann mit Kabelbindern an dem Metallkreuz befestigt. Dieses wurde wiederum an einer Fahrradgabel montiert und fixiert. Dann haben wir die Felge über einen Märklin Metall-Motor angetrieben, um eine gleichbleibende, einstellbare Geschwindigkeit zu erhalten, die der Geschwindigkeit eines durchschnittlichen Fahrradfahrers entspricht. Der finale Aufbau ist in Abb. 14 dargestellt:

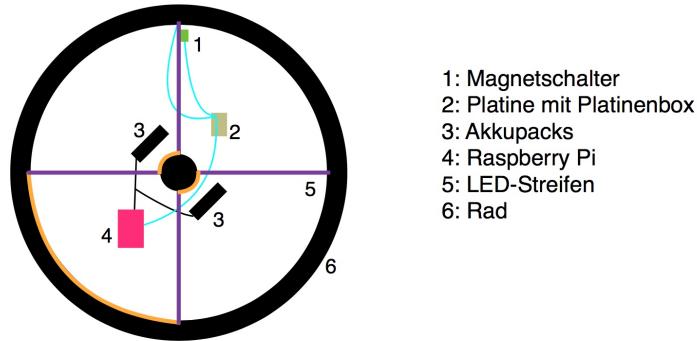


Abbildung 13: Skizze des Aufbaus

3.7 Welche Mindestgeschwindigkeit wird gebraucht?

Das menschliche Auge nimmt Bilder in einer Frequenz f_{Auge} von 10 bis 12Hz auf¹⁷. Um die Geschwindigkeit bei rund 10Hz zu berechnen, verwenden wir die Gleichung $v = \frac{\Delta s}{\Delta t}$. Auf die

Kreisbewegung angewandt entspräche dies $v = \frac{2\pi r}{T}$ in $\frac{m}{s}$. Da $\frac{1}{T}$ der Frequenz entspricht, der Umrechnungsfaktor von $\frac{m}{s}$ in $\frac{km}{h}$ 3,6 ist und $d = 2 \cdot r$ gilt, erhalten wir die Gleichung $v = \pi \cdot f_{Auge} \cdot d \cdot 3,6$. Durch das Anbringen der vier Streifen am Rad, benötigen wir nur ein viertel der ursprüngliche Geschwindigkeit. Demnach gilt:

$$v = \frac{\pi \cdot f_{Auge} \cdot d \cdot 3,6}{4} \quad (12)$$

Also beträgt die Mindestgeschwindigkeit rund $v \approx 17 \frac{km}{h}$ bei einem 24“ Rad, um ein vollständiges Bild zu erkennen. Die Geschwindigkeit, bei der schon ein Bild zu erkennen ist, ist jedoch kleiner, weil das Gehirn ein Bild selbst vervollständigt. Dem Gehirn genügt es, einige Konturen wahrzunehmen, um ein Bild oder Objekt zu erkennen, da es den Rest aus Erfahrung und Fantasie ergänzt¹⁸.

4 Ergebnis

Uns ist es gelungen, eine interaktivere, stilvollere und zum gleichen Maße sichere Lösung, im Vergleich zu einer Warnweste zu bauen. Es ist damingehend stilvoller, da man mit diesem Speichendisplay jedes Bild seiner Wahl anzeigen lassen kann, welches zusätzlich die Sicherheit eines Fahrradfahrers erhöht. Unser Speichendisplay ist dazu noch günstiger als vergleichbare Modelle. Wie man anhand der Auflistung der Kosten für Materialien sieht, beschränken sich die Materialkosten auf nur zirka 100 € (siehe 6.3). Aufgrund der Trägheit des Auges des Betrachters und der konstant hohen Geschwindigkeit des Rades, entsteht für den Betrachter der Eindruck eines ruhenden Bildes. Somit gelingt es uns, ein scheinbar ruhendes Bild auf dem Speichendisplay zu produzieren. Es ist uns damit auch gelungen, ein Stoppschild und andere Verkehrszeichen auf dem Speichendisplay anzuzeigen.



Abbildung 14: Skizze des Aufbaus

4.1 Ergebnisdiskussion

Nach einem ersten Start unseres Programmes haben wir festgestellt, dass der Raspberry Pi Zero, den wir wegen seiner geringen Größe und seinem kaum spürbaren Gewicht verwenden wollten, viel zu langsam ist, um ein hochauflösendes Bild darzustellen (siehe Abb. 7a). Der deutlich schnellere Raspberry Pi 3 erstellt hierbei einen viel besseres Bild (siehe Abb. 7b). Dies sieht man auch auf den beiden Bildern:

Aus diesem Grund haben wir uns für den schnelleren Raspberry Pi 3 entschieden, den wir auch in



(a) Smily mit Raspberry Pi Zero (b) Smily mit Raspberry Pi 3

Abbildung 15: Vergleich zwischen dem Raspberry Pi Zero und dem Raspberry Pi 3

Zukunft weiter verwenden werden. Der passt jedoch leider in keine für uns momentan verfügbare wasserdichte Verteilerbox. Daher ist das Speichendisplay momentan an einem Fahrrad noch nicht bei jedem Wetter einsetzbar.

4.2 Geplante Verbesserungen

Wir haben vor, unsere gesamte Apparatur wasserfest zu machen, damit man bei fast allen Wetterbedingungen mit dem Speichendisplay fahren kann. Im Laufe der weiteren Projektweiterentwicklung wollen wir die LED-Ansteuerung-Software durch selbstentwickelte Software ersetzen, um dieses Projekt ganz unser Eigen nennen zu können. Zum Steuern des Raspberry Pi planen wir den Raspberry Pi eine Website hosten zu lassen, auf die ein Nutzer über eine W-LAN Verbindung zum Raspberry Pi zugreifen und Bilder von seinem Endgerät auf den Raspberry Pi hochladen kann. Zusätzlich möchten wir insgesamt drei Magnetsensoren an das Rad montieren, um damit die Drehrichtung des Rades zu ermitteln. So können wir das Bild unabhängig von der Drehrichtung darstellen.

5 Zusammenfassung

Alles in allem ist das Speichendisplay eine interaktive, multifunktionale, kostengünstige und sicherheitsfördernde Anzeige, die jedoch noch kleine Kinderkrankheiten hat. Mit diesem Speichendisplay erhoffen wir uns, mehr Menschen zum sicheren Fahrradfahren, auch im Dunkeln, zu begeistern.

6 Anhang

6.1 Quellen- und Literaturverzeichnis

- 1)
<http://www.pd-f.de/wp-content/uploads/kalins-pdf/singles/themenblatt-die-fahrradwelt-in-zahlen.pdf> [Zugang am 03.01.2017, um 11:46 Uhr]
- 2)
http://www.monkeylectric.com/monkey_light_pro/ [Zugang am 10.01.2017, um 16:56 Uhr; Monkeylectric; Monkey Light Pro]
- 3)
<https://learn.adafruit.com/neopixels-on-raspberry-pi/software> [Zugang am 07.01.2017, um 17:08 Uhr; Toni DiCola; Adafruit-Neopixel-Library]
- 4)
<http://www.elektronik-kompendium.de/sites/com/0902081.htm> [Zugang am 9.1.2017, um 15:55]

- Uhr; Patrick Schnabel; USB-Stecker mit Steckerbelegung]
- 5)
<https://github.com/python-pillow/Pillow> [Zugang am 15.01.2017, um 11:37 Uhr; python- Pillow; Pillow]
- 6)
<https://www.raspberrypi.org/downloads/raspbian/> [Zugang am 15.1.2017 um 11:54 Uhr; Raspberry Pi Foundation; Raspbian]
- 7)
<https://i1.wp.com/www.novaspirit.com/wp-content/uploads/2016/10/wireing.jpg?ssl=1> [Zugang am 15.1.2017, um 09:38 Uhr; Novaspirit; USB Male Connector]
- 8)
<https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md> [Zugang am 15.1.2017, um 10:37 Uhr; Raspberry Pi Foundation; WLAN-Verbindung]
- 9)
<https://www.raspberrypi.org/documentation/configuration/raspi-config.md> [Zugang am 15.1.2017, um 10:14 Uhr; Alex Bradbury; Raspberry Pi Konfigurationseinstellungen]
- 10)
https://github.com/RubenHaag/Jufo_Spokescreen [Zugang am 15.1.2017, um 11:59 Uhr; Ruben Haag; Projektordner]
- 11)
<http://fritzing.org/download/> [Zugang am 18.01.2017, um 13:02 Uhr; Friends-of-Fritzing e.V.; Fritzing]
- 12)
https://www.conrad.de/de/abzweigkasten-l-x-b-x-h-75-x-40-x-37-mm-5229-licht-grau-ip54-629536.html?gclid=CPfQ7JrP29ECFcG7GwodF3UFtQ&insert_kz=VQ&hk=SEM&WT.srch=1&WT.mc_id=g oogle _pla&s_kwcid=AL!222!3!173789970902!!g!!&ef_id=WBpFUQAABUMWEwed:20170124202753:s [Zugang am 26.01.2017, um 16:41 Uhr; Conrad Electronic SE; Platinenbox]
- 13)
https://www.notebooksbilliger.de/raspberry+pi+3+model+b+arm+cortex+a53/?nbb=pla.google_&wt_cc2=913-0001_Hardware_257367&gclid=CMeX0KL759ECFYGw7Qod2q8CRQ [Zugang am 26.01.2017, um 16:45 Uhr; reichelt elektronik GmbH & Co. KG; Raspberry Pi 3]
- 14)
http://www.ebay.de/itm/like/252355489432?lpid=106&chn=ps&ul_noapp=true [Zugang am 27.01.2017, um 15:29 Uhr; eBay Europe S.'a.r.l.; Kabel]
- 15)
http://www.ebay.de/itm/WS2812B-5050-RGB-144LED-1M-String-Strip-Lights-Addressable-Waterproof-H3/111860633668?_trksid=p2141725.c100338.m3726&_trkparms=aid%3D222007%26algo%3DSIC.MBE%26a_o%3D1%26asc%3D20150313114020%26meid%3D9977e8ae4b7e42c8990b9f5cc22f9914%26pid%3D100338%26rk%3D1%26rkt%3D21%26sd%3D301963844528 [Zugang am 24.01.2017, um 18:47 Uhr; eBay Europe S.'a.r.l.; LED-Streifen]
- 16)
https://www.amazon.de/Anker-PowerCore-Powerbank-Ladegerät-Smartphone/dp/B01CU1EC6Y/ref=sr_1_1?s=ce-de&ie=UTF8&qid=1483383358&sr=1-1&keywords=akkupack [Zugang am 27.01.2017, um 16:41 Uhr; Amazon EU S.à r.l.; Powerbank]
- 17)
 Paul Read, Mark-Paul Meyer: „Restoration of motion picture film“, Woburn, MA, USA 2000, S. 24
- 18)
 Annette Lauber, Petra Schmalstieg: „Band 2: Wahrnehmen und Beobachten“, Stuttgart 2012, S. 9
- 19)

<https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf> [Zugang am 20.2., um 8:00 Uhr; ; Datei des LED-Streifens] (Anmerkung: Die Zugangsdaten beziehen sich stets auf den Zeitpunkt des letzten Zugriffes)

6.2 Danksagungen

Wir vom Projekt „Speichendisplay“ bedanken uns sehr bei der Fahrradwerkstatt der Holzkirche Licherfelde für die Spende einer Fahrradgabel, ohne die unser Projekt nicht möglich gewesen wäre. Ebenfalls bedanken wir uns bei unserem Betreuer René Gorriz vom Paulsen-Gymnasium, der uns bei unserem Projekt tatkräftig unterstützt.

6.3 Kostenkalkulation und Materialliste

Material	Kosten
Raspberry Pi 3 ^[13]	36,99 €
Platinenbox ^[12]	1,29 €
Kabel ^[14]	2,58 €
Halterung	15,00 €
LED-Streifen ^[15]	34,28 €
Akkupack ^[16]	10,99 €
Summe	101,13 €