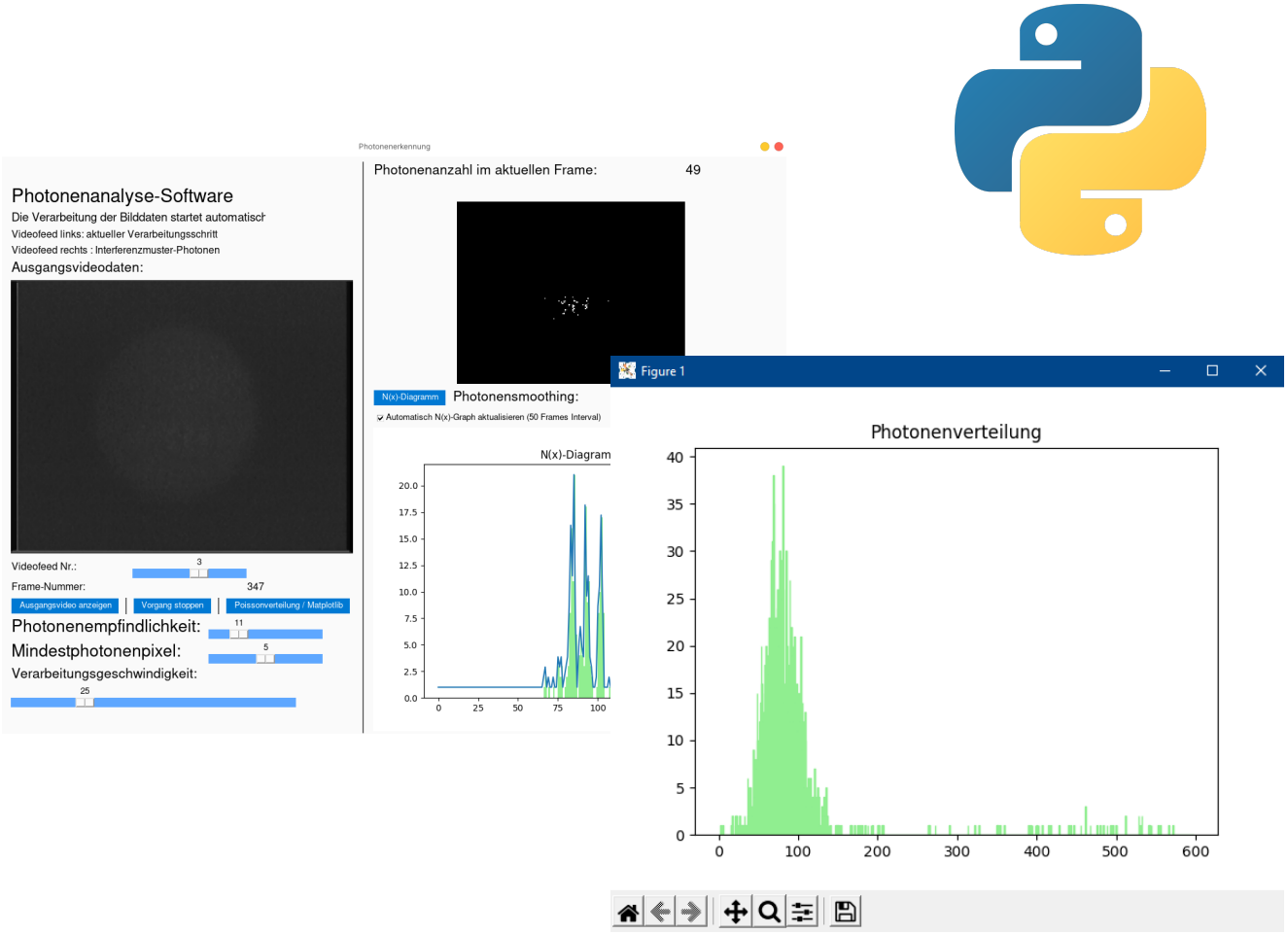
Digitale Verarbeitung von Bilddaten am Doppelspaltexperiment



Name: Daniel Anders

Paul-Gerhardt-Gymnasium

Informatik : 12/1

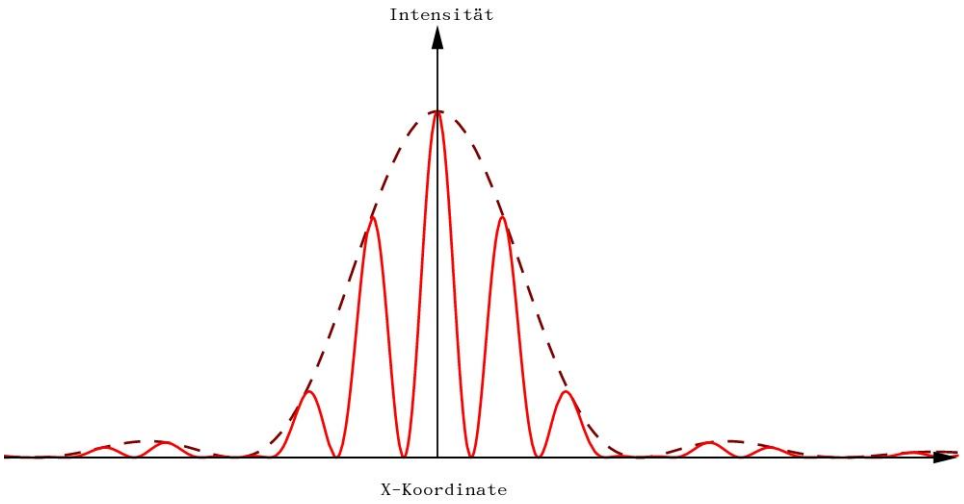
Jahr 2021/22

Inhaltsverzeichnis

1 Vorbetrachtung

1.1 Einleitung

1802 führte Thomas Young erstmals eines der bedeutsamsten Experimente der modernen Physik durch, das Doppelspaltexperiment. Er versuchte das wellenartige Verhalten von Licht am Doppelspalt durch die Bildung eines Interferenzmusters an einem Schirm nachzuweisen. Dieser Versuch legte den Grundstein für die spätere Entstehung der Quantentheorie. Dabei beobachtete er, dass sich die Lichtstreifen am Schirm periodisch und mit sich vom Zentrum stetig verringertem Maxima abbildet. [1 ; 2]



[4]

Idealverlauf des Interferenzmusters am Doppelspalt

Dieses Muster entsteht durch die Überlagerung der einzelnen Wellen, welche sich untereinander in den Minima des Interferenzmusters blockieren und in den Maxima summieren. Aufgrund der fehlenden Messtechnik am Anfang des 19. Jahrhunderts war Thomas Young die Wellen-Teilchen-Dualität [3] der Photonen in Abhängigkeit der Beobachtung des Experiments nicht bewusst.

Mittels einer modernen Kamera, Photonenmultiplikatoren [5 ; 6] wie denen in einem Nachtsichtgerät und der Verwendung digitaler Bildverarbeitung kann man dieses Doppelspaltexperiment nachstellen um die Ergebnisse genauer zu analysieren. Eine selbst programmierte Software, basierend auf der OpenCV2-Bibliothek für Python, sollte diese Auswertung und Analyse veranschaulichen und erleichtern, sodass diese beispielsweise eine Erweiterung zum Unterrichtsgeschehen in Physik darstellt.

Die Programmiersprache Python 3.9 und Bibliotheken wie OpenCV, PySimpleGUI, MatplotLib, SciPy [7] und Numpy [8] wurden für die Erstellung dieser Software in Verwendung gezogen. Die grafische Oberfläche, auch bekannt als GUI [9] (graphical user interface), ermöglicht dem Endnutzer die Zugänglichkeit der Software.

1.2 Programmierung

1.2.1 Vorwort zur Programmierung

Sämtlicher geschriebener Code innerhalb dieser Facharbeit ist exemplarisch und auf ein Minimum gekürzt, sodass dessen grobe Funktionalität besser deutlich wird. Der gesamte Quelltext wurde deshalb *open-source* [10] auf GitHub veröffentlicht und darf von Dritten eingesehen, verwendet und modifiziert werden.

[https://github.com/Funkez123/nightvision-photon-detector]

1.2.2 Python 3.9

Python ist eine universell einsetzbare Programmiersprache, welche ursprünglich von Guido van Rossum seit dem Jahr 1990 entwickelt wird [11]. Sie ist eine höhere Programmiersprache, die meistens von einem Interpreten abhängig ist. Durch die Verwendung eines Interpreten ist Python eine eher vergleichsweise langsame Programmiersprache, welche allerdings in anderen Bereichen punkten kann. [12]
Durch die einfache leserliche Syntax-Struktur und die oftmals automatische Deklarierung von Variablen (zb. Integer, Float, Long, Double…) ist sie auch für Programmieranfänger sehr geeignet. Python lässt sich auf fast allen Betriebssystemen ausführen und findet im Forschungsbereich große Popularität durch das schnelle und unkomplizierte Programmieren von kompakten Code [13]. Dies wird in folgendem Code-Beispiel ersichtlich:

```
1  ## Beispielcode zur bestimmung gerader/ungerader Zahlen in Python
2  x = 129871398172            ## Zahlendeklarierung
3  if x%2 != 0:                ## if (Rest von (Variable / 2) ungleich null) :
4      print("ungerade Zahl")  ## Print - Ausgabe in der Kommandezeile
5  else:
6      print("gerade Zahl")
```

If-Bedingungen werden in Python durch einfaches Einrücken der gewünschten Befehle mit vier Leerzeichen anhand der Syntax entweder geöffnet oder geschlossen [14].

1.2.3 Programmierumgebung

Die Verwendung einer besonderen Programmier-IDE war bei der Erstellung des Programms nicht notwendig. Der Python-3.9-Interpret wurde systemweit installiert und der Code im Atom-Texteditor [15]

Atom-Editor-Logo

geschrieben. Sämtliche Bibliotheken wurden direkt in Python durch den pip3-Paketmanager [16] installiert, wobei es manchmal ratsam ist Python-Projekte in Containern zu trennen um Probleme mit den Bibliotheksversionen zu vermeiden.

1.2.4 OpenCV-4.5.4

OpenCV-Logo

OpenCV ist eine quelloffene und auf C++ basierende Bibliothek für verschiedenste Programmiersprachen, die sich auf algorithmische Bildverarbeitung und maschinelles Sehen spezialisiert [17]. Sie verarbeitet in Python die Daten von Webcams, Video- und Bilddateien, in dem für jeden Frame ein 2-Dimensionales Array mit allen notwendigen Pixelwerten erstellt wird. Dieses umfangreiche Modul beinhaltet zusätzlich Befehle für die Manipulation von Farbwerten, Auswertung von Bilddaten und für das Weichzeichnen des Frames in verschiedensten Methoden [18]. Folgender Code beschreibt exemplarisch die Einrichtung der OpenCV-Bibliothek in Python, die Initialisierung einer Webcam, die einheitliche Skalierung der Bilddaten auf eine angemessene Bildgröße und das Anzeigen des Frames in einem externen Fenster :

```python
import cv2  ## Importiert OpenCV in Python

cap = cv2.VideoCapture(0)   #Speichert Bilddaten der Kamera in "cap" ab
if not cap.isOpened():      #Überprüft ob Kamera verfügbar ist (mit "if" und "not")
    print("Fehler beim Öffnen der Kamera")
while True:  # while schleife sorgt für Endlosabfrage der Webcam-Frames
    ret, frame = cap.read() # frame = der aktuelle verfügbare Frame
    aufloesung = [720,576]  # gewünschte Auflösung der Bilddaten in Arrayform abgespeichert
    frame = cv2.resize(frame,aufloesung, interpolation=cv2.INTER_LINEAR)
    # Skalieren der 640x480 Webcam-daten auf 720x576px in OpenCV mit der INTER_LINEAR methode
    cv2.imshow('Input', frame)  #zeigt das Skalierte Bild in einem externen Fenster

# schließt alle Fenster und Prozesse
cap.release()
cv2.destroyAllWindows()
```

1.2.5 Matplotlib 3.4.2

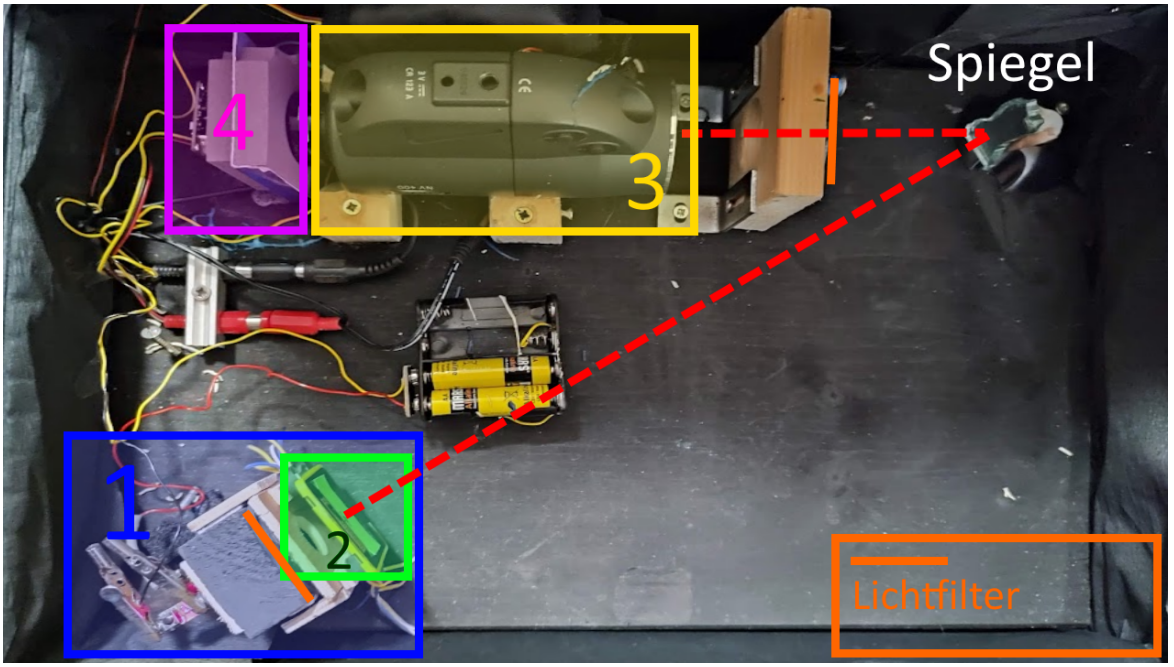

Matplotlib-Logo

Um die gesammelten und berechneten Daten anzeigen und besser auswerten zu lassen wird auf das umfangreiche Python-Modul *Matplotlib* [19] zurückgegriffen. Linienverläufe, Graphen, Histogramme und anderweitige mathematische Darstellungen erleichtern dem Endnutzer die Evaluierung der vorhandenen Daten. Matplotlib ermöglicht es direkt die Achsen der Diagramme, Farben und einzelne Skalierungen näher zu bestimmen. Verwendet wurde es um sowohl das Histogramm für die Häufigkeitsverteilung der Photonen einzelner Frames, sowie das N(x)-Diagramm zu erstellen.
Folgender Code sollte vereinfacht die grundlegendsten Befehle für die Darstellung eines Graphen zeigen:

```python
import matplotlib.pyplot as plt # Matplotlib-Bibliotheksimport
import numpy as np #Numpy für manche Rechenoperationen benötigt

x_array = np.arange(0,15,0.01)  #Erstellt ein Array von 0 bis 15 im interval 0.01
y = np.sin(x_array) # der Y-Wert des Graphen ist y=sin(x)
plt.plot(x_array,y) # plottet den Graphen von f(x) für alle X-Werte des Arrays
plt.show() # zeigt den Graphen in einem externen Fenster an
```

1.3 Der Versuchsaufbau

Innerhalb eines Koffers, welcher vom Außenlicht völlig abgeschirmt ist, befindet sich ein roter Spielzeuglaser **1** mit 650 nm Wellenlänge und >1mW Leistung, der auf einen Doppelspalt **2** mit dem Abstandsparameter d = 0,05 mm gerichtet ist. Ein Spiegel lenkt daraufhin das rote Licht auf das Nachtsichtgerät **3** mit bis zu 20.000-facher Verstärkung [20] (Minox NV 400). Das verstärkte Bildsignal wird folglich von einer empfindlichen Kamera **4** zu einem Composite-Video Signal umgewandelt, welches allerdings noch analog und unverarbeitet ist [21].

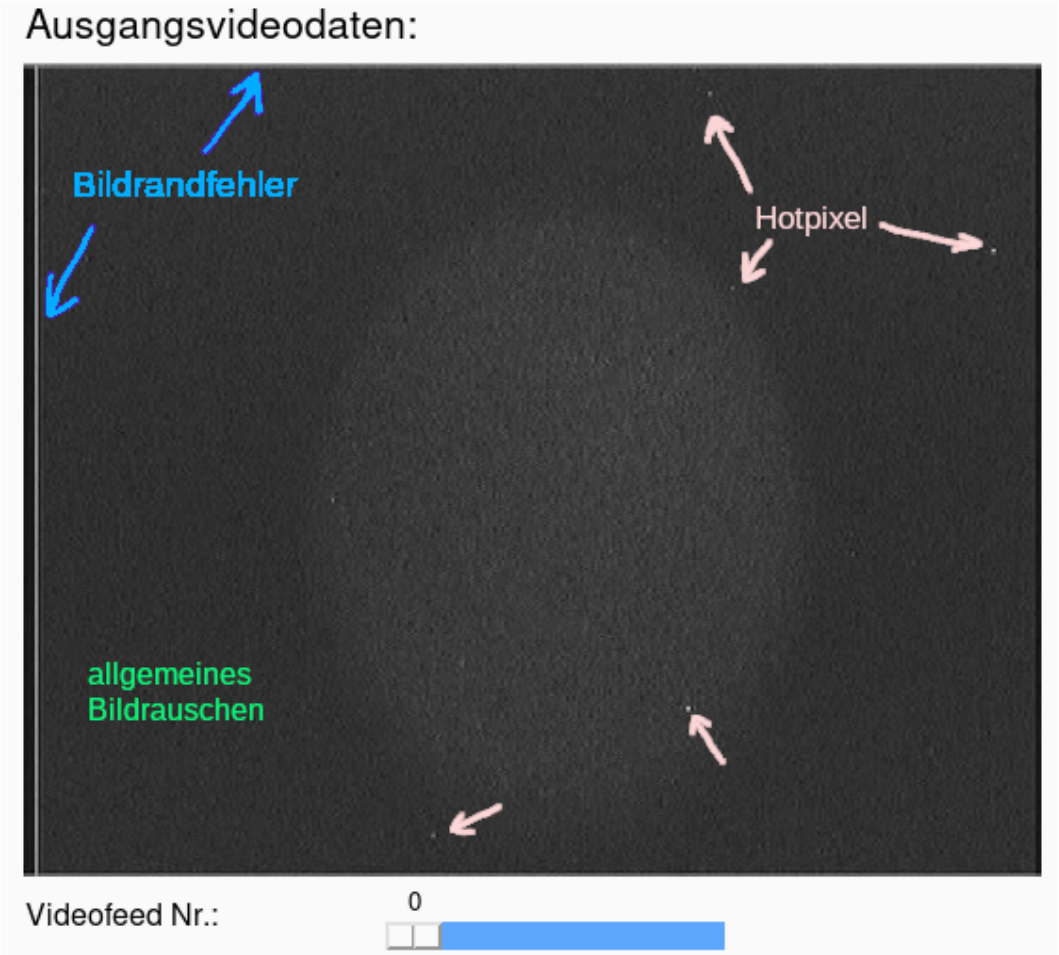


beschriftetes Bild der Versuchsumgebung

1.4 Problematik des Videosignals

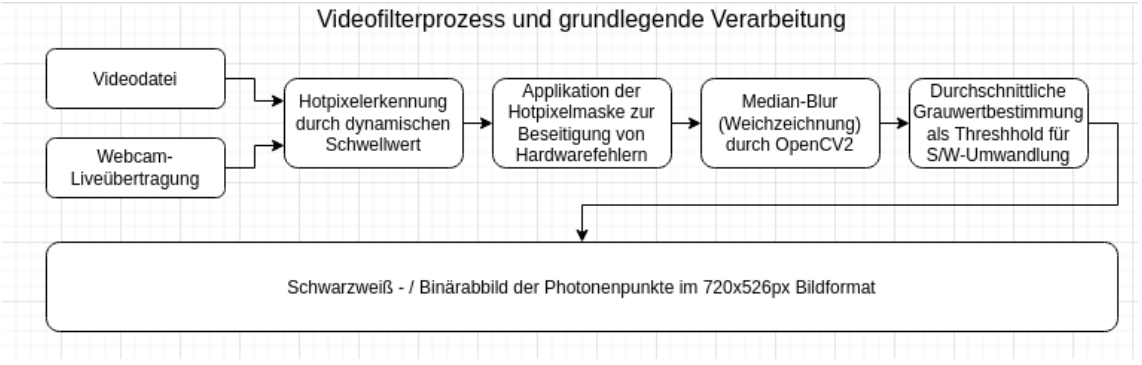
Analoge Videosignale basieren auf der präzisen zeitlichen Spannungsänderung und der genauen Abfrage dieser Werte durch den Empfänger im Nanosekunden-Bereich. [22 ; 23] Parasitäre Kapazität durch elektrische Leiter und Bauelemente stören genau diese Spannungsdifferenzen zwar geringfügig, aber für genaue Messungen einzelner Pixel

merklich [24 ; 25]. Besonders bei einer Farbbildübertragung kommt es regelmäßig zu einer Verfälschung der Pixelwerte (zb. Farbrauschen) und merklichen Verzerrungen am Bildrand. Auffällig ist außerdem auch ein graues Rauschen im gesamten Bild, welches vom zufälligen Auftreten der Photonen getrennt werden muss.
Bei kompletter Abdunklung der Kamera ist ein grauer Hintergrund zu erkennen, der stark von einem unregelmäßigen Rauschen geprägt ist. Diesen kann man einerseits auf ein Rauschen des Kamerasensors, als auch auf die Mikrochips des Moduls [26] zurückführen.
Fehler am Kamerasensor oder an der Mikrokanalplatte zweiter Generation (2nd Gen. +) [27] des Nachtsichtgeräts, auch als Hotpixel bekannt, sind statische Zustände von Photonensensoren, die bei dem Ausgangsbild markante Bildfehler hervorrufen. Diese Hardware-Problematik entsteht oftmals mit der Zeit und wird bei steigender Temperatur der Messelektronik deutlicher [26].



exemplarischer Frame mit beschriebenen Störfaktoren

2 Lösungsansatz

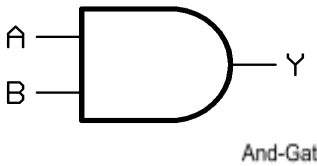


[28]

Die Weichzeichnung des Videos hat direkt zwei erkennbare Vorteile, da sie einerseits den Einfluss der entfernten Hotpixel verringert und gleichzeitig das unerwünschte Rauschen im Messbereich drastisch reduziert [29]. Hierzu wurde der cv2.medianBlur( ) Befehl verwendet, der im 720x576px großen Bild einen Kernel von 5x5px appliziert und diesen so durch Berechnung der Mittelwerte weichzeichnet. Der Median-Blur-Operator (auch Rangordnungsfilter) [30 ; 31] steht hierbei für eine Operation, die nacheinander auf alle Bereiche der Pixelmatrix eines Bildes einen vorbestimmten Kernel (auch eine Matrix) legt und dann den Median der erfassten Werte des Kernels in dessen Zentrum setzt [32 ; 33] . Die Kernelgröße von 5x5px wurde gewählt, da sie das Rauschen, bzw auch die meisten Hotpixel-Fehler spurlos entfernt, und gleichzeitig das gewünschte Aufblitzen einzelner Photonen nicht stört.

```
18  medianframe = cv2.medianBlur(frame, 5)
19  #Weichzeichnen des Frames mit dem Median eines 5x5 Kernels
```

Mit Anbetracht der Tatsache, dass die Bildverarbeitung in Echtzeit ablaufen sollte, wird eine einfache und mathematisch unkomplizierte Beseitigung der Bildfehler benötigt. Hierzu kommen *Masken* [34 ; 35] in Frage, welche zusammen mit der *Bitwise-AND* Operation [36] nur Bildpunkte zulässt, die in der Maske nicht einen Wert von 0 tragen. Das *AND-Gate* (UND-Gatter) lässt sich den logischen Operatoren zuordnen und schaltet erst am Ausgang Y auf einen Wert, wenn sowohl A und B nicht null sind :



[37]
And-Gate

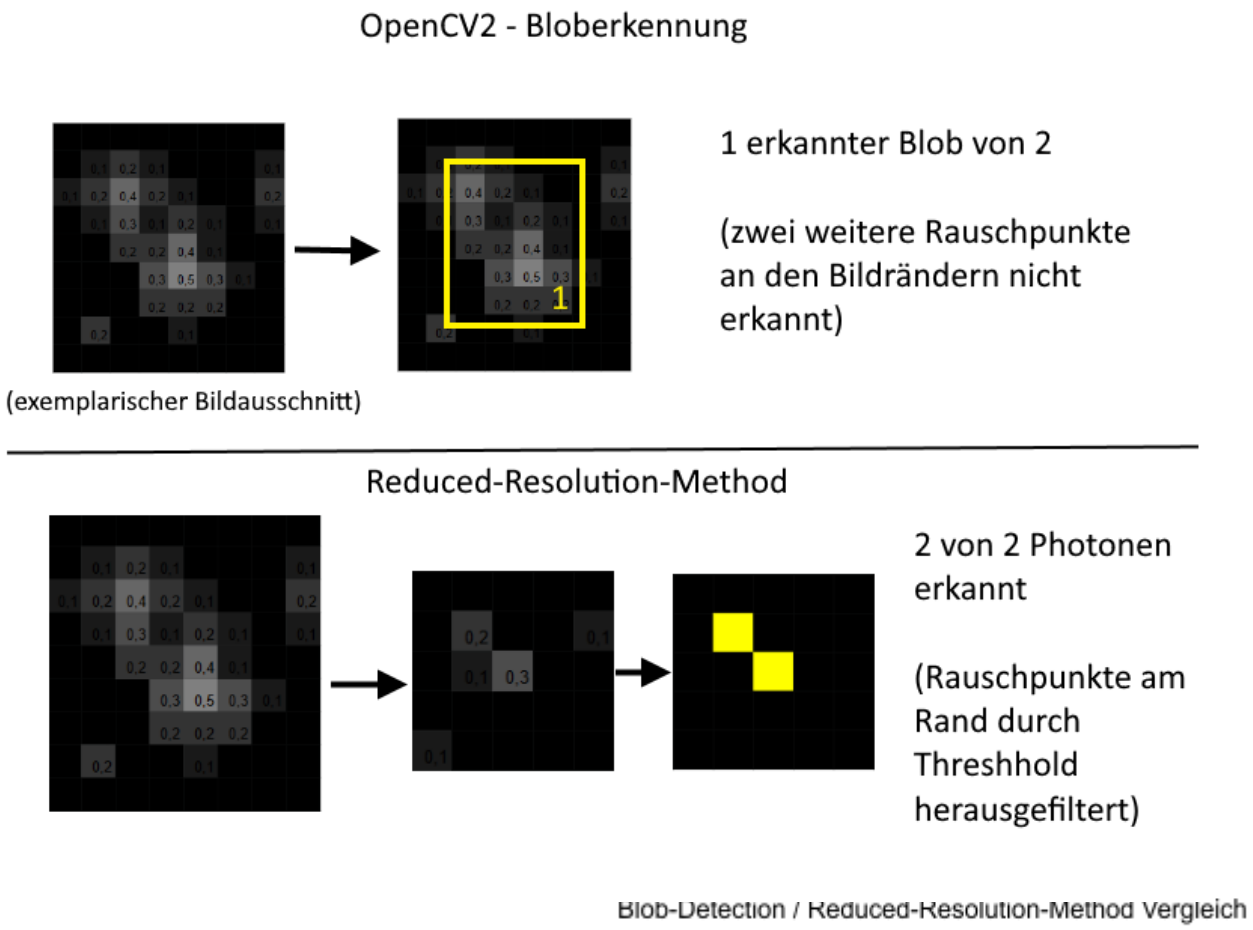
Masken lassen sich am besten als eine Art Filter oder Schablone beschreiben, die nur bestimmte Areale eines Bildes übernehmen lässt und die ungewollte Stellen leer oder schwarz lässt. Ein selbstbestimmter und dynamischer *Schwellwert* [38] würde automatisch auffällige Bildfehler im Video erkennen, und ein Differenzbild [39] ermöglichen.
Der abgebildete Abschnitt des Codes stellt die Befehle und dessen Verwendung für die Implementierung genau dieses Lösungsansatzes dar :

```
#hp_image = erster verfügbarer Frame (mit Hotpixeln)
# frame = aktueller Frame
#berechnet ein B/W Bild mit einem festgelegten Schwellenwert
hotpixel_th = 130
th, mask = cv2.threshhold(hp_image, hotpixel_th,255,cv2.THRESH_BINARY)
mask_inv = cv2.bitwise_not(mask)
#wandelt die Pixel der Maske zusätzlich in ein Binärformat (0 oder 1) um
masked = cv2.bitwise_and(frame,frame, mask=mask_inv) #Bildet die Maske ab
```

2.1 Photonenzählung

Um die genaue Anzahl der eintreffenden Photonen im Ausgangsframe zu bestimmen kommen mehrere Methoden in Frage, die jeweils ihre eigenen Vor- und Nachteile haben. Die sicherlich bekannteste und erstmal einleuchtendste Methode wäre die integrierte *Bloberkennung* [40] in OpenCV. Sie hat den Vorteil, dass sie parameter-technisch sehr flexibel und simpel in den Code zu integrieren ist. Andererseits ist diese Methode bei relativ kleinen Blobs in der Größenordnung weniger Pixel ziemlich fehlerbehaftet und würde die Statistik verfälschen [41]. Ein weiterer zu beachtender Faktor ist die Verarbeitungsgeschwindigkeit dieses Prozesses. Auch wenn die meisten Prozessoren diese Aufgaben in Echtzeit problemlos bewältigen können, ist es im Hinblick einer mobilen Nutzung für *Single-Board-Computer* wie den RPI-3 oder Pine64 eine riesige Herausforderung [42], da hier trotz Optimierungen wirklich nur maximal 10 Bilder pro Sekunde erreicht werden können. Somit würden im Webcam-Modus mehr als 50% der nutzbaren Frames entfallen.

Die stattdessen verwendete Methode ist sowohl CPU-ressourcensparend, als auch genau genug um die Abweichung zu der reellen Photonenanzahl möglichst gering zu halten. Dabei wird das schwarz-weiße Ausgangsbild auf eine geringere Auflösung von 180x144px linear mit dem Befehl cv2.resize( ) herunterskaliert [43]. Die Pixel sind groß genug um das Aufblitzen eines Photons als Pixelwertänderung beizubehalten und ist dennoch resistent zu dem “Verschmelzen” [44] zweier Photonen zu einem. Besonders helle Stellen im Bild, wie die bei Anhäufungen mehrerer Photonen können benachbarte Pixelfelder mit einem Grauwert belegen, der fälschlicherweise mitgezählt wird. Ein Threshold-Wert kann dieses Problem jedoch minimieren. Der Vergleich beider Methoden wird in folgender Darstellung deutlich :



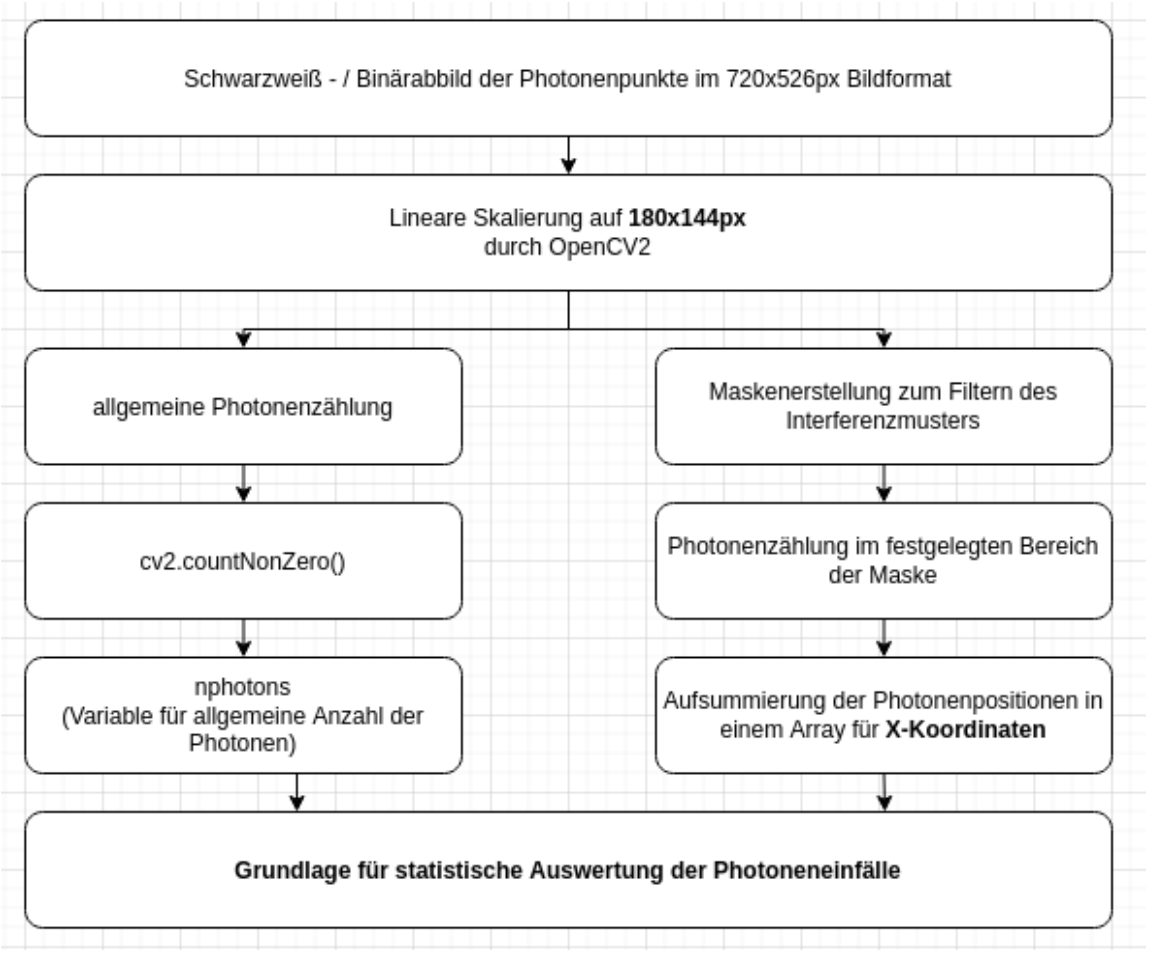
Blob-Detection / Reduced-Resolution-Method Vergleich

2.1.1 Implementierung im Code (Photonenzählung)

Zur Reduzierung der Auflösung wurde hauptsächlich auf OpenCV's integrierte Befehle zurückgegriffen und die Werte der einzelnen Zwischenbilder werden in Python 3.9 als 2-dimensionale *Arrays* abgespeichert [45].
Die 180x144 große Pixelmatrix wird dann im laufenden *Loop* mittels OpenCV-4.5.4 nach Pixeln abgefragt, welche ungleich 0 sind [46]. Die der temporäre Wert der Photonenanzahl im aktuell bearbeiteten Frame wird dann in einer dafür vorgelegten Variable gehalten und erstellt unter anderem die Grundlage für die statistische Auswertung des Experiments.

```
34  dimension = (180,144) #gewünschte Größe für Reduced_Res_Methode
35  #Skaliert das Bild linear auf gewünschte Größen
36  framedresize = cv2.resize(frame,dimension,interpolation=cv2.INTER_LINEAR)
37  nphotons = cv2.countNonZero(framed2) #Zählung der Photonen im Bild
```

Zusätzlich zur allgemeinen Photonenzählung wird ein weiterer Bildabgleich erstellt, der zusammen mit einer Maske den genauen Bereich des Interferenzmusters im Ausgangsbild markiert. Somit werden Fehler durch Bild-/Kamerarauschen noch weiter reduziert und es lässt sich dadurch genauer ein N(x)-Diagramm anhand der Photoneneinfallspunkte in relation zur X-Achse erstellen. Die Erstellung der Histogramme ist auf die regelmäßige Aktualisierung der Photonen-Arrays in Echtzeit angewiesen.



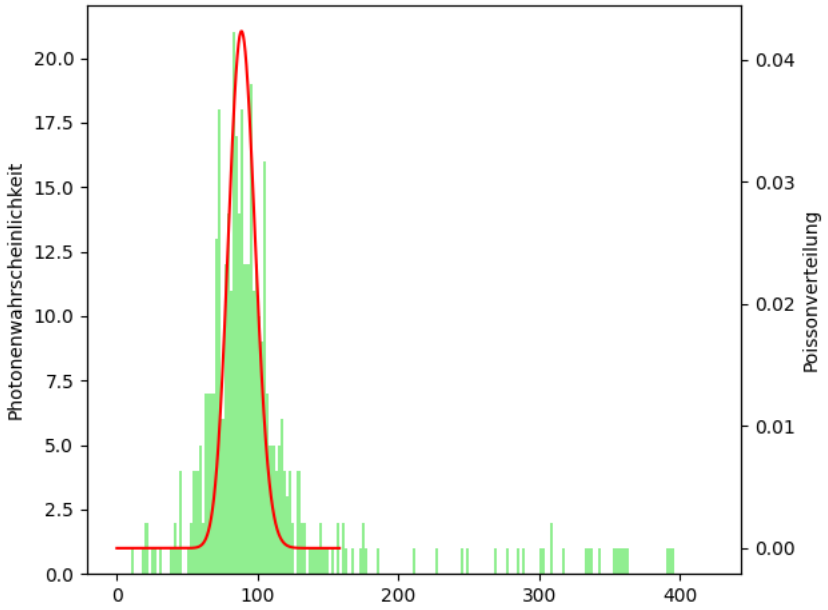
[28]

3 Vorverarbeitung für die Statistische Auswertung

3.1 Häufigkeitsverteilung

Ein besonderes Phänomen in der Beobachtung der einzelnen Lichtblitze wird ersichtlich wenn man die Anzahl dieser innerhalb eines Frames in einem Histogramm darstellt. Photonen tendieren entweder in Klumpen (*photon-bunching*) oder vereinzelt (*anti-bunching*) aufzutreten [47]. Deren Unterscheidung ist von der Position einzelner Messwerte in Relation zur erstellten Poisson-Verteilung [48] abhängig.
Werte außerhalb der idealen Wahrscheinlichkeitsverteilung sind den Photonenklumpen zuzuordnen, wohingegen Werte unterhalb des Graphen das *anti-bunching* widerspiegeln. Folgende Gleichung beschreibt hierbei den Verlauf der Poisson-Verteilung[49]:

$$P(X = x) = \frac{\lambda^x}{x!} * e^{-\lambda}, \mathrm{x} \in \mathbf{N}$$



Das Histogramm wurde aus 140 Sekunden Videomaterial (3360 Frames) erstellt und zeigt eindeutig die Ähnlichkeit zur Poisson-Verteilung. Auf der X-Achse befinden sich die Werte der gezählten Photonen im jeweiligen Frame und auf der Y-Achse die Photonenwahrscheinlichkeit, welche als Balken (*Bin*) [50] gezeigt wird.

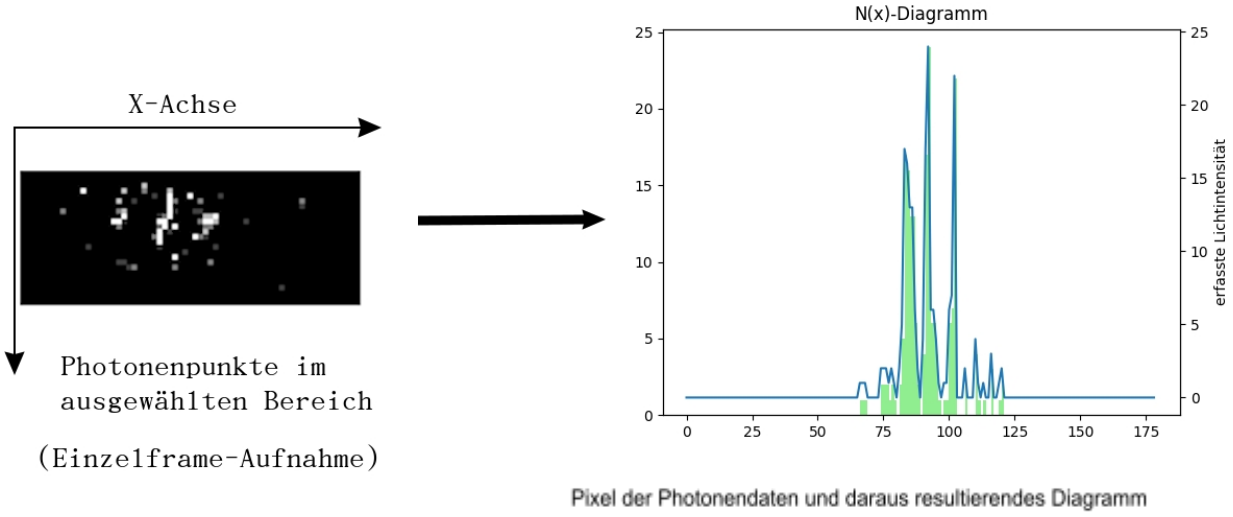
```
10  def draw_poisson_histo():  # definiert eine neuen Befehl (kann global aufgerufen werden)
11      ## erstellt eine Menge von Zahlenwerten für das Histogram basierend auf gesammelten Daten des Arrays phot_anzahl[]
12      sample = np.random.poisson(phot_anzahl)
13      # Array für die einzelnen Balken des Histograms, Interval 1, von 0 bis maximalwert der Photonen:
14      bin = np.arange(0,max_x,2) # max_x = vorher gefundene höchte Photonenanzahl zur besseren Skalierung
15      # Das Histogram der Photonenverteilung
16      n, bins, patches = plt.hist( sample, bins=bin, color='lightgreen', label='Photonenverteilung')
17      plt.plot()
18      plt.show()
```

Der Erwartungswert λ errechnet sich hierbei aus dem *Median* [51] aller gesammelten Werte des dafür vorgesehenen Arrays “sample[ ]” .

```
lmd = (np.median(sample))  ## Auslesen des Erwartungswertes aus dem Histogram
```

3.2 Das Interferenzmuster / N(x)-Diagramm

In Anbetracht eines vorher festgelegten Bereich lassen sich einzelne Photonenklumpen besonders häufig beobachten. Das Auftreten dieser Klumpen ist an den Maxima des durch das im Doppelspalt entstehenden Interferenzmuster am höchsten. Die Zwischenspeicherung der X-Koordinaten eintreffender Lichtpunkte ermöglicht das Erstellen eines genaueren N(x)-Diagramms über einen Zeitraum der mehrere Frames erfasst. Ein Schieberegler lässt den Nutzer genau die Anzahl der verwendeten Photonen festlegen.

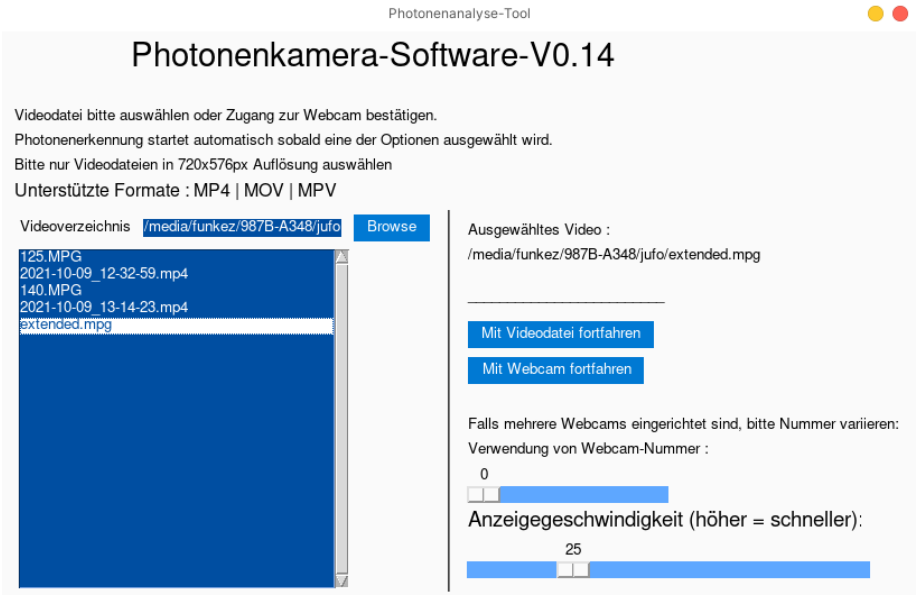


Pixel der Photonendaten und daraus resultierendes Diagramm

4 Die Benutzeroberfläche

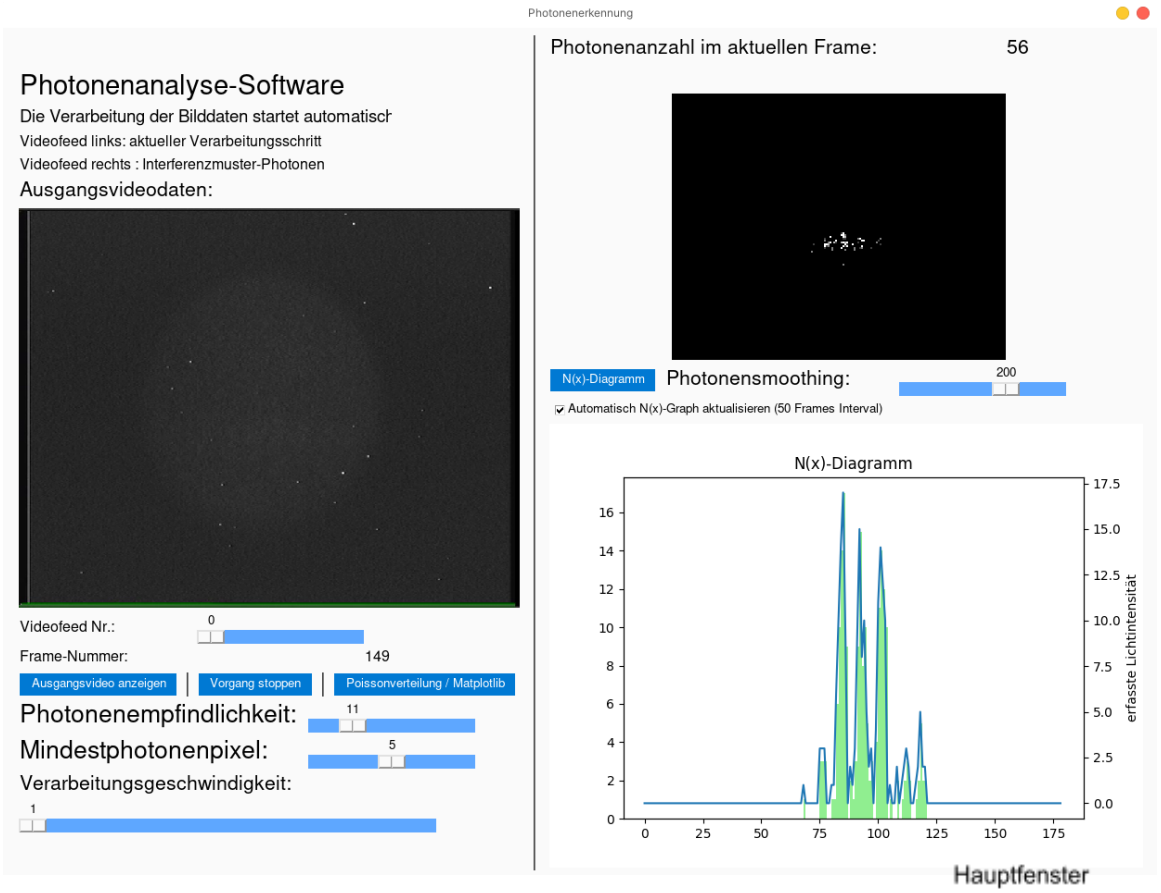
Um den Endbenutzer eine unkomplizierte Nutzung der Software zu ermöglichen, wird eine grafische Benutzerfläche, auch bekannt als GUI benötigt. Sie beinhaltet Knöpfe, Eingabefelder, Menüs und Schieberegler, die dem Nutzer die Möglichkeit gibt mit der Software zu interagieren und wichtige Parameter für die Bildverarbeitung zu bestimmen. PySimpleGUI sollte die Lösung für genau diese Aufgaben darstellen und ist ein Python-Modul, welches auf der GUI von unter anderem *Tkinter* basiert [52 ; 53].

Für die Software wurde einerseits ein einfaches Einrichtungsfenster erstellt, bei dem man die Auswahl zwischen der Verwendung eines Videos oder einer Webcam hat. Dort kann man in einem Dateiexplorer die Videodatei, bzw mittels Schieberegler die Nummer der Webcam des Betriebssystems festlegen.



Einrichtungsfenster der Software

Das Hauptfenster hingegen beinhaltet zwei 2 Bildausgaben und ein Canvas [54] (rechts digitale “Leinwand” für Darstellungen), auf dem Matplotlib dann das N(x)-Diagramm anzeigt. Weitere Textausgaben zeigen beispielsweise die derzeitige Frame-nummer oder die Anzahl der Photonen im aktuell verarbeiteten Bild. Optional hat der Nutzer die Möglichkeit externe Fenster von OpenCV öffnen zu lassen [55], die teils sogar eine genauere Bildanalyse der Zwischenframes unter Linux zulässt.



Es wurde für den Hintergrund der GUI und die Knöpfe ein helles Layout gewählt, da somit alle Elemente auch an beispielsweise Beamern in unterschiedlichen Lichtverhältnissen weiterhin erkennbar bleiben. Die Einrichtung der GUI wird auch als Front-End-Entwicklung [56] bezeichnet, da sie an erster Stelle die Schnittstelle zwischen dem Code und dem Endnutzer darstellt. Ein benutzerfreundliches Layout muss somit vor dem Rendern der Oberfläche vorher in einem *Layout*-Array deklariert werden. Hierbei werden Kommas und eckige Klammern für die Abgrenzung einzelner Elemente verwendet und folgen strengen Syntax-Regelungen [57]. Ein Komma steht nach der Deklarierung aller Benutzerelemente und grenzt diese einzeln voneinander ab. Eine eckige Klammer “[ ]” um beispielsweise einen Knopf erstellt genau diesen in einer neuen Zeile. *Columns* zählen auch zu solchen Elemente und verhalten sich wie eine Art Tabelle, die die Oberfläche vertikal in zwei Hälften spalten. Die Änderung eines Zustandes oder Wertes durch die Interaktion eines Endnutzers mit beispielsweise einem Schieberegler bezeichnet man als Event [58], dessen *Value* (deu.: Wert) mittels If-Befehl nach einer Bedienung abgefragt werden kann.

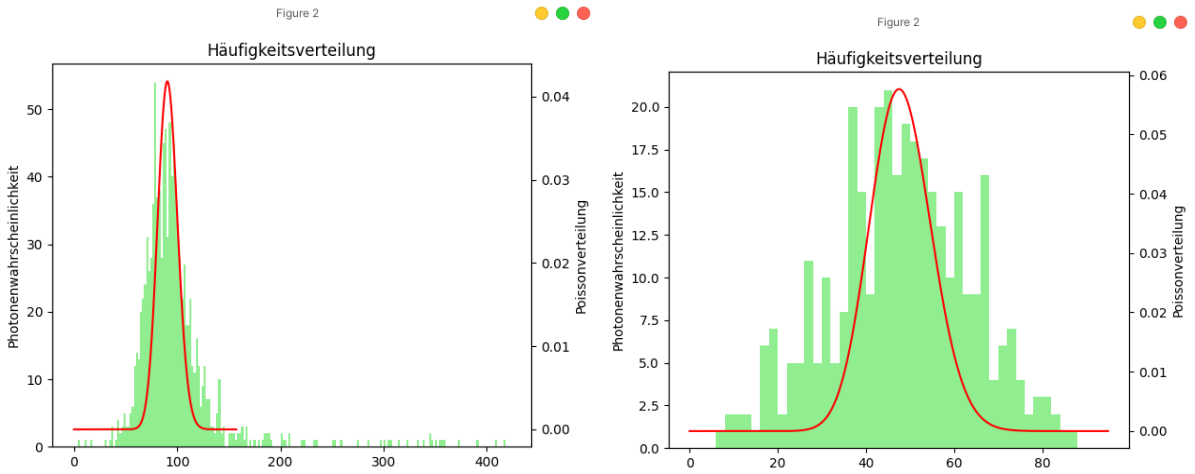
```python
import PySimpleGUI as sg # importieren der PySimpleGUI Bibliothek

layout = [ [sg.Text('Lorem ipsum dolor sit amet')], #Beispieltext
           [sg.Button('Ein Knopf')]]    # folgende Zeile ein Knopf
# richtet das Fenster "Testprogramm" vor bestimmten Layout ein
window = sg.Window('Testprogramm', layout)
while True:  # While-Schleife für stetige Abfrage der Events
    event, values = window.read() #ließt Input werte des Fensters mit jedem Durchlauf der Schleife
    if event == "Ein Knopf":  # Wenn der Knopf gedrückt wird/bzw dieses Event ausgeführt wird:
        sg.Popup("der Knopf wurde gedrückt") # neues Popupfenster mit Text
    if event in (None, 'Abbrechen'): # Wenn Programm geschlossen wird, dann Python: "break"
        break
window.close() #schließt Fenster am Ende
```

5 Auswertung

In Betrachtung der sich vorher gesetzten Ziele und Anwendungsbereiche lässt sich festhalten, dass der Code grundlegend voll funktionsfähig ist und in der Lage ist Photonen in einem vorher unverarbeiteten Video oder sogar in Echtzeit aus einem Live-Feed durch eigenständige Filterung auszusortieren, sie zu zählen und statistische Diagramme zu erheben. Die grafische Oberfläche wurde so gestaltet, dass sie benutzerfreundlich und übersichtlich ist. Das Programm funktioniert sowohl unter Windows als auch Linux, wodurch der Gebrauch der Software auch an einem kompakten RaspberryPi-3 unabhängig von einem Computer möglich ist.

5.1 Diagramm-Auswertung

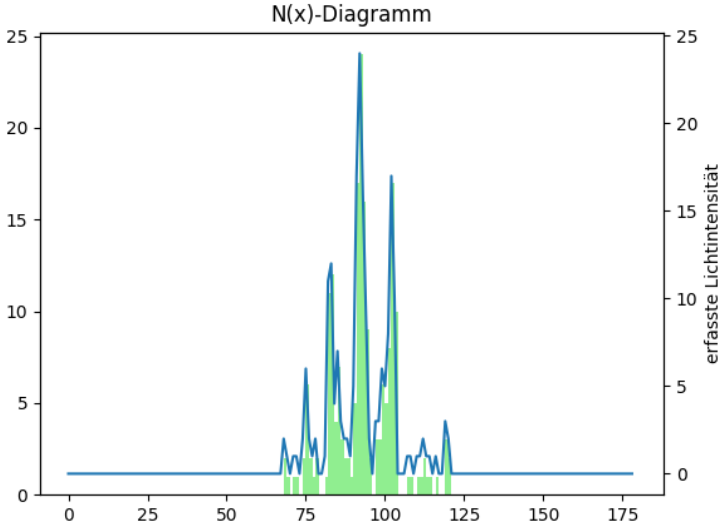
Beide Diagramme zeigen nach der Beseitigung einiger Störfaktoren grundlegend eine Ähnlichkeit zu ihren gewünschten Verläufen. Die Histogramme zeigen abhängig von der Photonenempfindlichkeit teils unterschiedliche Graphen. Dies hängt damit zusammen, dass bei einer zu großen Empfindlichkeit auch ungewolltes Rauschen mit in den Datensatz hineingelangt und einen großen Einfluss auf den Diagrammverlauf nimmt. Die Häufigkeitsverteilung der Photonen nähert sich in unter den richtigen Messbedingung und mit steigender Laufzeit der Poisson-Verteilung an. Dessen Ähnlichkeit wird deutlicher wenn man insgesamt mehr Photonen in den einzelnen Ausgangs-Frames hat, bzw. sich die X-Achse des Histogramms (Photonen-Anzahl eines Frames) erweitert.



Die Verteilung links besitzt einen Erwartungswert *Lambda* = 91 und die X-Achse ist nur bis zu einem Wert von 500 (nphotons) skaliert, da am Anfang der Messdatenerfassung kurzzeitig Außenlicht durch ein kleines Kabelloch an der Seite des Koffers für wenige Frames gelangt ist und somit kurzzeitig die Messergebnisse gestört hatte. Dadurch, dass die Aufnahmen jedoch über 3000 Frames zur Auswertung bereitstellen hatten die ersten wenigen Frames keinen bedeutsamen Einfluss auf den Verlauf des Graphen.

Im Diagramm rechts wurde mit derselben Empfindlichkeit und absolut frei von Außenlicht eine Messung mit jeweils nur 1400 Frames durchgeführt. Darauf sieht man neben einem Erwartungswert von 47 Photonen zusätzlich das Phänomen des *Photonen-Bunching* und *Anti-Bunching* deutlicher.

Bei der Auswertung des N(x)-Diagramms wird es jedoch etwas schwieriger, da man hier mit jeder Neuberechnung aller 50 Frames etwas andere Diagramme erhält. Diese haben jedoch alle die markanten und erkennbaren Maxima und Minima, die es auch im Idealverlauf des Interferenzmuster eines Doppelspaltexperiments gibt. Auffällig ist, dass die Maxima bei einer X-Achseneinteilung von 180 Pixeln durchschnittlich 9px auseinander sind. Eine andere Wellenlänge des Lasers oder die Verwendung eines anderen Doppelspalts sollten erwartungsgemäß genau diesen durchschnittlichen Abstand beeinflussen [59].



5.2 Schwierigkeiten bei der Entwicklung

Bei der gesamten Programmierung gab es tatsächlich keine größeren Schwierigkeiten, die die gesamte Entwicklung für längere Zeit gestoppt haben. Verbesserungswürdig ist dabei definitiv die Performance des Programms. Python 3.9 ist keinesfalls perfekt und nicht sonderlich für grafische Verarbeitung optimiert [60]. Dies liegt unter anderem daran, dass Python grundlegend auf einem einzigen *Thread* [61] ausgeführt und interpretiert wird. Grafische Oberflächen benötigen eine ständige Abfrage sämtlicher Elemente und blockieren teils während einer Iteration die Ausführung von weiterem Code. Um diesen Umstand zu verbessern stellt Python die Möglichkeit zur Verfügung mittels eines Zusatzmoduls *Threading* [62] zu ergänzen um Prozesse parallel laufen zu lassen. Dies ändert allerdings nichts an der Tatsache, dass PySimpleGUI eine zwar funktionierende, allerdings nicht schnelle Anbindung zu OpenCV hat. Hierfür werden die Rohdaten des OpenCV-Bildbuffer abgefragt [63] und dann bei PySimpleGUI im nächsten gezeichneten Frame im Single-Core-Modus anzeigt. Dieser Prozess benötigt einen unproportional hohen Rechenaufwand der CPU und braucht in Abhängigkeit der dargestellten Bildgröße auf leistungsschwachen Geräten teils einige Zentisekunden. Das Layout der GUI ist aus Limitierungen einzelner Bibliotheken zueinander nicht dynamisch und könnte auf kleineren Bildschirmauflösungen sogar zu Darstellungsproblemen führen. Dadurch entfällt die sonst vom Benutzer übliche Skalierung der Anwendung und es wird eine Bildschirmauflösung von 1920x1080px empfohlen.

6 Selbstständigkeitserklärung

Hiermit bestätige ich, dass diese Facharbeit selbstständig verfasst wurde, ohne auf weitere Hilfsmittel oder Quellen zurückzugreifen, die nicht in den Quellen vermerkt wurden.

7 Quellen

Stand [21.Nov.2021] [letzte Veränderung 14:19]

Format : [TT.MM.JJJJ : SS:MM-Uhr]

[1] https://www.leifiphysik.de/optik/beugung-und-interferenz/grundwissen/doppelspalt [13.10.2021 : 18:35]
[2] https://www.leifiphysik.de/optik/wellenmodell-des-lichts/geschichte/thomas-young-1773-1829 [13.10.2021 : 18:55]
[3] https://de.wikipedia.org/wiki/Doppelspaltexperiment [13.10.2021 : 20:21]
[4] https://www.geogebra.org/m/sGcuTjME (unter eigener grafischer Bearbeitung) [13.10.2021 : 21:10]
[5] https://www.cosmos-indirekt.de/Physik-Schule/Photomultiplier [14.10.2021 : 18:00]
[6] https://www.spektrum.de/lexikon/physik/mikrokanalplatte/9726 [14.10.2021 : 18:41]
[7] https://scipy.org/ [14.10.2021 : 17:12]
[8] https://numpy.org/doc/stable/user/whatisnumpy.html [14.10.2021 : 19:32]
[9] https://www.computerhope.com/jargon/g/gui.htm [14.10.2021 : 19:39]
[10] https://opensource.org/ [14.10.2021 : 20:21]
[11] https://de.wikipedia.org/wiki/Python_(Programmiersprache)[14.10.2021 : 20:35]
[12] https://medium.com/swlh/a-performance-comparison-between-c-java-and-python-df3890545f6d [14.10.2021 : 20:41]
[13] https://www.opensourceforu.com/2017/06/python-user-friendly-language-coding/ [14.10.2021 : 21:12]
[14] https://studyflix.de/informatik/python-if-3169 [14.10.2021 : 21:30]
[15] https://atom.io/ [14.10.2021 : 21:40]
[16] https://pip.pypa.io/en/stable/getting-started/ [14.10.2021 : 21:55]
[17] https://docs.opencv.org/4.x/d1/dfb/intro.html [14.10.2021 : 22:13]
[18] https://blog.codecentric.de/2017/06/einfuehrung-in-computer-vision-mit-opencv-und-python/ 14.10.2021 : 22:35]
[19] https://matplotlib.org/stable/tutorials/index.html [18.10.2021 : 18:45]
[20] https://www.natuerlich-jagd.de/waid-und-werk/nachtsichttechnik-im-detail-analog-und-digital/ [18.10.2021 : 18:35]
[21] https://de.wikipedia.org/wiki/Composite_Video [18.10.2021 : 20:15]
[22] http://www.glennchan.info/articles/technical/composite/composite-video.htm [21.10.2021 : 21:12]
[23] https://en.wikipedia.org/wiki/Composite_artifact_colors [21.10.2021 : 21:14]
[24] https://resources.pcb.cadence.com/blog/2019-how-parasitic-capacitance-and-inductance-affect-your-signals [21.10.2021 : 22:11]
[25] https://datasheets.maximintegrated.com/en/ds/MAX9517-MAX9524.pdf [21.10.2021 : 23:14]
[26] https://de.wikipedia.org/wiki/Bildrauschen [21.10.2021 : 23:31]
[27] https://www.omegon.eu/de/beratung/nachtsichtgeraet/c,9067 [21.10.2021 : 23:49]
[28] https://app.diagrams.net/ (Diagramm eigenständig bei DrawIO erstellt) [22.10.2021 : 00:55]
[29] https://docs.gimp.org/2.10/de/gimp-filter-median-blur.html [22.10.2021 : 01:41]
[30] https://theailearner.com/tag/cv2-medianblur/ [22.10.2021 : 02:23]
[31] https://russianblogs.com/article/49341015221/ [24.10.2021 : 18:35]
[32] https://it.wikipedia.org/wiki/Filtro_mediano [24.10.2021 : 16:20]
[33] https://docs.opencv.org/4.x/dc/dd3/tutorial_gausian_median_blur_bilateral_filter.html [24.10.2021 : 17:11]
[34] https://www.py4u.net/discuss/71437 [24.10.2021 : 17:41]
[35] https://www.pyimagesearch.com/2021/01/19/image-masking-with-opencv/ [25.10.2021 : 17:51]
[36] https://pro.arcgis.com/de/pro-app/latest/tool-reference/spatial-analyst/how-bitwise-math-tools-work.htm [25.10.2021 : 18:01]
[37] https://upload.wikimedia.org/wikipedia/commons/5/5d/Logic-gate-and-us.png [25.10.2021 : 19:21]
[38] https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html [25.10.2021 : 19:48]
[39] https://docs.opencv.org/3.4/d0/d7a/classcv_1_1SimpleBlobDetector.html [25.10.2021 : 19:51]
[40] https://www.pyimagesearch.com/2017/06/19/image-difference-with-opencv-and-python/ [26.10.2021 : 20:20]

[41] https://answers.opencv.org/question/71691/single-blob-multiple-objects-ideas-on-how-to-separate-objects/ [26.10.2021 : 21:11]
[42] https://www.pyimagesearch.com/2017/10/09/optimizing-opencv-on-the-raspberry-pi/ [26.10.2021 : 22:27]
[43] https://www.tutorialkart.com/opencv/python/opencv-python-resize-image/ [26.10.2021 : 23:41]
[44] https://github.com/opencv/opencv/issues/9096 [29.10.2021 : 16:11]
[45] https://www.oreilly.com/library/view/learning-opencv-3/9781491937983/ch04.html [29.10.2021 : 16:51]
[46] https://www.programcreek.com/python/example/89360/cv2.countNonZero [29.10.2021 : 17:25]
[47] https://www.spektrum.de/lexikon/optik/photonenstatistik/2545 [30.10.2021 : 17:51]
[48] http://www.mikomma.de/phstat/photostatistik.html [30.10.2021 : 17:56]
[49] https://matheguru.com/stochastik/poisson-verteilung.html [30.10.2021 : 18:21]
[50]https://www.khanacademy.org/math/statistics-probability/displaying-describing-data/quantitative-data-graphs/a/histograms-review [30.10.2021 : 18:41]
[51] https://de.statista.com/statistik/lexikon/definition/85/median/ [30.10.2021 : 18:52]
[52] https://de.acervolima.com/einfuhrung-in-pysimplegui/ [01.11.2021 : 19:12]
[53] https://pysimplegui.readthedocs.io/en/latest/ [01.11.2021 : 20:11]
[54] https://www.python-kurs.eu/tkinter_canvas.php [01.12.2021 : 20:18]
[55] https://www.geeksforgeeks.org/python-opencv-cv2-imshow-method/ [04.11.2021 : 17:12]
[56] https://www.arocom.de/fachbegriffe/frontend-development [05.11.2021 : 23:59]
[57] https://github.com/PySimpleGUI/PySimpleGUI/blob/master/DemoPrograms/Demo_Layout_Generation.py [06.11.2021 : 21:56]
[58] https://github.com/PySimpleGUI/PySimpleGUI/issues/4292 [07.11.2021 : 18:30]
[59]https://www.leifiphysik.de/optik/beugung-und-interferenz/versuche/wellenlaengenbestimmung-mit-dem-doppelspalt [09.11.2021 : 20:21]
[60] https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python [13.11.2021 : 00:35]
[61]https://www.quora.com/Is-Python-single-threaded-or-multithreaded [18.10.2021 : 18:35]
[62] https://realpython.com/intro-to-python-threading/ [20.10.2021 : 18:35]
[63] https://qiita.com/Gyutan/items/b4781ee1baa4966699ef [21.10.2021 : 14:19]

Bilderverzeichnis (Bildquellen):

Atom-Editor-Logo :

https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcShrJ957I5HO-BNFs0F7lcRVa5T87FwcbSMyT3jolHj52gVYNAlznCbKQbFdKf0j5wyQzU&usqp=CAU [21.10.2021 : 20:09]

OpenCV-Logo :

https://3.bp.blogspot.com/-yvrV6MUueGg/ToICp0YIDPI/AAAAAAAAADg/SYKg4dWpyC43AAfrDwBTR0VYmYT0QshEgCPcBGAYYCw/s1600/OpenCV_Logo.png

[21.10.2021 : 20:13]

MatplotLib-Logo :

https://de.wikipedia.org/wiki/Matplotlib#/media/Datei:Matplotlib_icon.svg

[21.10.2021 : 20:17]

Python-Logo :

https://upload.wikimedia.org/wikipedia/commons/thumb/c/c3/Python-logo-notext.svg/2048px-Python-logo-notext.svg.png

[21.10.2021 : 20:21]