Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Danny Mac Allister

Konzeption und Realisierung eines Prototyps zur automatisierten Zählung von Insekten mittels Bilderkennungssoftware

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Danny Mac Allister

Konzeption und Realisierung eines Prototyps zur automatisierten Zählung von Insekten mittels Bilderkennungssoftware

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Michael Erhard
Zweitgutachter : Dr. Claudius Noack

Abgegeben am 08. März 2019

Danny Mac Allister

Thema der Bachelorthesis

Konzeption und Realisierung eines Prototyps zur automatisierten Zählung von Insekten mittels Bilderkennungssoftware

Stichworte

Raspberry PI; OpenCV; Python; Bilderkennung; Bildverarbeitung

Kurzzusammenfassung

Diese Arbeit umfasst die Entwicklung eines Prototyps, welcher mittels einer Bilderkennungssoftware eine automatisierte Zählung von Insekten ermöglicht.

Die Bildverarbeitung wird mittels der Programmbibliothek OpenCV auf einem Raspberry Pi umgesetzt. Dabei werden verschiedene Bildverarbeitungsschritte validiert und auf ihre Einsetzbarkeit für die automatisierte Erkennung überprüft.

Abschließend werden die Testergebnisse der automatisierten Zählung anschaulich in verschiedenen Graphen bereitgestellt.

Danny Mac Allister

Title of the paper

Development and realization of a prototype for automated counting of insects by using an image recognition software

Keywords

Raspberry PI; OpenCV; Python; Image recognition; Image processing

Abstract

This report is about a development of a prototype for an automated counting system, which can be used for counting Insects with the help of an image recognition software.

The image processing is implemented on a Raspberry Pi by using the program library OpenCV. Various steps of image processing were validated and checked about their applicability for the automated recognition.

Finally, the results of the automated counting were shown in different graphs.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listingverzeichnis

1 Einleitung

Die These *„Stechmücken werden von klassischen Leuchtmitteln eher angezogen, als von modernen LED-Leuchten.“* (Claudius Noack, 2018) veranlasste eine Lösungsfindung zur Belegung oder Widerlegung dieser These.

In Anbetracht der ökologischen Wichtigkeit von Insekten und der eventuelle Einfluss auf das Verhalten der Insekten, durch den steigenden Austausch von klassischen Leuchtmitteln durch moderne LED-Leuchten, stellte sich eine Untersuchung der These als äußerst interessant heraus. Bereits 2015 schrieb die Welt *„Eine Milliarde tote Insekten an Deutschlands Straßenleuchten“* [9] und wies damit auf die Relevanz der Lichtverschmutzung von LED Beleuchtung hin.

Die ökologische Verträglichkeit verschiedener Lampen wurde bereits ausgiebig untersucht. Unter anderem von der Tiroler Umweltanwaltschaft mit dem Projekt „Helle Not“, deren Mission ist es, die Bevölkerung für die Ursachen von Lichterverschmutzung zu sensibilisieren und die Auswirkung von falscher Beleuchtung auf die Umwelt aufzuklären. Das Projekt entstand 1999 aus der Zusammenarbeit der Tiroler Umweltanwaltschaft und Tiroler Landesmuseen. [11] Die Helle Not bezieht sich hier im Bereich der Insektenforschung häufig auf die Studie „Anlockwirkung moderner Leuchtmittel auf nachtaktive Insekten“ von Mag. Dr. P.Huemer et al. [5], in der die Anlockwirkung verschiedener Leuchten auf nachtaktive Insekten untersucht wurde. In dieser Studie kamen speziell konstruierte Lichtfallen zum Einsatz, die in der freien Natur aufgestellt wurden. Der Fangradius einer solchen Lichtfalle ist äußert Artenabhängig, da verschiedene Insektenarten unterschiedlich weit angelockt werden. Bei dem Einsatz von Lichtfallen kann kein Einfluss darauf genommen werden, welche Insektenarten gezielt untersucht werden. Es müssen sämtliche gefangene Insekten im Nachgang aussortiert und gezählt werden. Es kann lediglich versucht werden die Lichtfallen in Gebieten aufzustellen, in denen die gewünschten Insektenarten gehäuft auftreten.

Um einen Einfluss auf die zu untersuchenden Insektenarten und -anzahl zu erhalten ist es nun Ziel dieser Arbeit, einen Prototypen zu konzipieren, mit dem das

Verhalten ausgewählter Insekten untersucht werden kann. Hierbei soll die Möglichkeit bestehen verschiedene Lockmittel und ihre Auswirkung auf das Verhalten verschiedener Arten von Insekten zu untersuchen.

Mittels einer Bildererkennungssoftware soll eine automatisierte Zählung der Insekten in einem festen Testareal realisiert werden. Durch die Erkennung der Positionen, der einzelnen Insekte, kann veranschaulicht werden welche Lockmittel von den Insekten bevorzugt werden. Der Prototyp sollte eine einfache und kostengünstige Möglichkeit bieten, den Prozess der Insektenuntersuchung zu automatisieren und zu vereinfachen. Der Einsatz des Raspberry Pi´s als Herzstück des zu realisierenden Prototyps stand hierbei bereits im Vorhinein fest, da dieser unter anderem im Bereich der Bilderkennung in verschiedenen Projekten zum Einsatz kommt.

Im ersten Teil der Arbeit werden zunächst die Grundlagen dargestellt, auf die der zu realisierende Prototyp beruht. Dabei werden nach den Softwaregrundlagen, die wichtigsten Methoden der Bildverarbeitung vorgestellt, welche im späteren Verlauf ihren Einsatz finden. Anschließend werden der Aufbau des Prototypens und seine Bestandteile erläutert. Nach der Vorstellung des Aufbaus wird die Durchführung der automatisierten Erkennung aufgezeigt, die in eine Testerkennung, in der zunächst schwarze Kreise eingesetzt wurden, und in die Insektenerkennung unterteilt ist. Im anschließenden Teil wird die Auswertung der Insektenerkennung veranschaulicht. Die gesammelten Erkenntnisse werden im Fazit resümiert. Die Arbeit endet in einer Zusammenfassung und einem Ausblick auf die zukünftige Entwicklung und Forschung.

2 Grundlagen

In diesem Kapitel werden die Grundlagen dargestellt, auf denen der zu realisierende Prototyp beruht. Zunächst wird die Programmierumgebung und das zu Grunde liegende Betriebssystem vorgestellt.

2.1 Software

2.1.1 Raspbian

Raspbian ist ein auf Debian und Linux basierendes Betriebssystem. Es handelt sich hierbei um ein eigens für den Raspberry Pi entwickeltes Betriebssystem und ist das gängigste unter allen kompatiblen Betriebssystemen, welches für jegliche Anwendungen des Raspberry Pi´s genutzt wird. Raspbian ist daher eine Wortzusammensetzung von Raspberry und Debian. [7]

Folgend ist zu Veranschaulichung eine Desktop Ansicht des Raspberry Pi´s dargestellt.

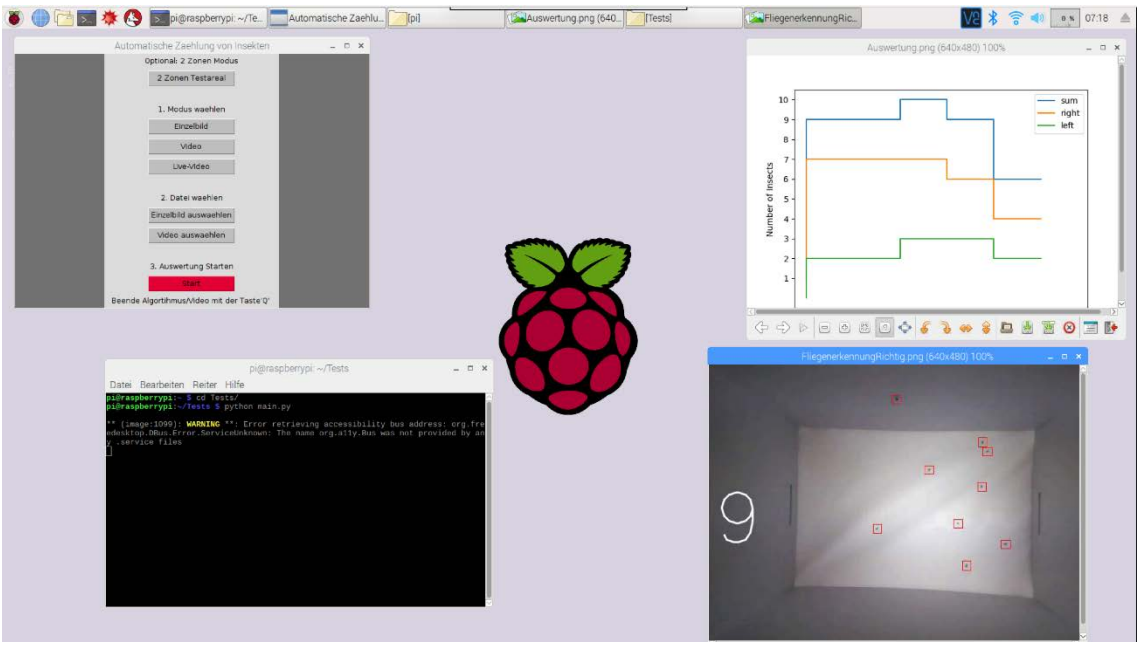


Abbildung 1:Darstellung Raspbian-Desktop mit geöffneten Programmen

2.1.2 Python

Für die Programmierung der Bildverarbeitung stehen verschiedene Programmiersprachen zur Verfügung. Die am meisten verbreitete Programmiersprache, im Zusammenhang zum Raspberry Pi und der nachfolgend erläuterten Bibliothek OpenCv, ist die Programmiersprache Python. Hierbei handelt es sich um eine sehr übersichtliche Programmiersprache. Blöcke, wie zum Beispiel If-Schleifen, werden durch bloßes Einrückungen, anstelle von geschweiften Klammern strukturiert. Für die Anwendung von Python existieren verschiedene Lektüren (z.B. siehe [6]).

2.1.3 OpenCV

Bei OpenCV handelt es sich um eine Programmbibliothek, bei der bereits verschiedene Algorithmen für die Bildverarbeitung implementiert sind. Es handelt sich hierbei um eine frei verfügbare Software, welche seit Juni 2000 in Erscheinung trat. Die Entwicklung wurde seitdem vor allem durch die Firma Intel übernommen.

Das „CV“ in OpenCV steht für „Computer Vision“, welches in der Regel mit „Maschinelles Sehen“ übersetzt wird.

Größter Vorteil von OpenCV ist die große Verbreitung und die bestehende Communitiy, sowie die große Auswahl an bestehenden Algorithmen. OpenCV wird für viele Anwendungen in der Bilderkennung genutzt (u.a. Gesichtserkennung und Objektdetektion), dabei werden große Mengen an Dokumentationen und Beispielen zu der Anwendung von OpenCV auf Webseite https://opencv.org/ bereitgestellt.

2.2 Bildvorverarbeitung

Um Objekte in einem Bild ausfindig zu machen, existieren verschiedene Methoden an Bildvorverarbeitungen. In diesem Kapitel werden unterschiedliche Möglichkeiten der Vorverarbeitung und Merkmalserkennung vorgestellt, die für das Projekt in Frage kommen und deren Vor- und Nachteile dargestellt. Hierbei wurden verschiedene Literaturen zur Grundlagenermittelung herangezogen. Die jeweilige Literatur, in der die verschiedenen Methoden erläutert werden, wird zu Beginn der einzelnen Methoden als Referenz zum Literarturverzeichnis angegeben.

2.2.1 Thresholding

Thresholding [1], in Deutsch auch bekannt als Schwellenwertverfahren, ist meist der erste Schritt in der Bildverarbeitung, um zum Beispiel im späteren Verlauf Objekte besser zu erkennen. Als Grundlage für dieses Verfahren dient immer ein Bild in Graustufen.

Beim Thresholding wird jeder Pixel eines Bildes untersucht und überprüft, ob seine Intensität einen Schwellenwert überschreitet und nachfolgend in Schwarz oder Weiß dargestellt wird. Als Resultat erhält man ein Binärbild, welches nur aus schwarzen und weißen Pixel besteht.

Ein klarer Nachteil bei diesem Verfahren ist, dass zuvor bekannt sein muss, welchen Grauwert ein gesuchtes Objekt hat, um dieses hervorzuheben. Ein falscher Schwellenwert könnte das gesuchte Objekt ausblenden. Zudem hat dieses Verfahren ein sehr großes Problem mit Helligkeitsverläufen. Dunklere Ecken werden nach der Durchführung des Schwellenwertverfahrens ebenfalls Schwarz, sodass diese im späteren Verlauf als Objekte erkannt werden.

2.2.2 Filtering

Filtering [8] ist eine weitere Methodik ein Bild zur Objekterkennung vorzubereiten, um somit das Bild von unerwünschtem Rauschen zu befreien und wichtige Merkmale hervorzuheben. Zu den eher unerwünschten Merkmalen gehören auch kleinere, dunkle Kanten die im späteren Verfahren nicht von der Software erkannt werden sollen.

Bei der Anwendung eines Filters auf ein Bild werden zuvor Matrizen gebildet, die den gewünschten Effekt des Filters umsetzen und dann mit dem zu bearbeiten Bild gefaltet werden. Da es sich bei einem Bild um ein ortsdiskretes Signal handelt, können hier bereits bekannte Faltungstheoreme angewendet werden.
Für die zeitdiskrete Faltung ist allgemein die folgende Definition bekannt:

$$f(n) = a(n) * b(n) = \sum_{k=-\infty}^{\infty} a(k) \cdot b(n-k) \tag{2.1}$$

Da es sich bei einem Bild aber um ein Signal im zweidimensionalen Raum handelt ist hierzu folgende Definition anzuwenden:

$$f(x,y) = a(x,y) * b(x,y) = \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} a(j,k) \cdot b(x-j, y-k) \tag{2.2}$$

Diese Filtermatrizen bezeichnet man in der Bildverarbeitung als Filterkerne (in den folgenden Formeln mit K abgekürzt) und diese sind in ihrer Länge meist variabel. Dabei gilt umso größer der Kern, desto stärker der Effekt.

Anschließend ist ein Beispiel einer Anwendung von einer 3x3 Filtermatrix auf eine 4x3 Bildmatrix dargestellt.

$$\text{Bild:} \quad B(x,y) = \begin{bmatrix} 1 & 2 & 3 & 3 \\ 9 & 9 & 8 & 8 \\ 2 & 2 & 2 & 2 \end{bmatrix} \tag{2.3}$$

$$\text{Kern:} \quad K(x,y) = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.4}$$

$$B_f(x,y) = \sum_{j=x-1}^{x+1} \sum_{k=y-1}^{y+1} B(j,k) \cdot K(x-j, y-k) \tag{2.5}$$

$$\text{gefiltertes Bild:} \quad B_f(x,y) = \begin{bmatrix} 27 & 35 & 23 & 24 \\ 4 & -3 & -3 & -3 \\ -27 & -35 & -33 & -24 \end{bmatrix} \tag{2.6}$$

Im Beispiel ist erkennbar, dass sich die Bildwerte deutlich geändert haben. Welchen genauen Effekt, u.a. der im Beispiel verwendete Filter auf ein Bild hat wird in diesem Kapitel erläutert. Die Anwendung der jeweiligen Filter wird Anhand eines Testbildes im späteren Verlauf veranschaulicht.

2.2.2.1 Box Filter

Der Box Filter [4] ist der einfachste Filter der verwendet werden kann, auch bekannt als Mittelwertfilter. Hierbei wird ein Areal um einen Pixel festgelegt und unter Berücksichtigung aller Pixel im Areal ein Mittelwert gebildet. Dem Pixel in der Kernmitte wird dieser Mittelwert zugeteilt.

$$K = \frac{1}{K_{width} \cdot K_{height}} \cdot \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix} \tag{2.7}$$

Bei diesem Verfahren wird das Bild gleichmäßig unscharf, welches zwar zu einer Reduzierung von Rauschen führt, aber auch sämtliche Kanten verwischt. Da es allerdings bei der Detektion von Objekten unteranderem auf die Kanten der Objekte ankommt, ist dieser Filter eher weniger geeignet.

2.2.2.2 Gaussian Blur

Gaussian Blur [4] ist die gängigste Methode um Rauschen in einem Bild zu mindern. Dieser Filter verteilt die Gewichtung der Pixel vom Kern aus bis zum Rand der Matrix mit Hilfe der Gauß-Verteilung. Dabei hat das Pixel in der Mitte der Matrix die höchste Gewichtung.

In der Bildverarbeitung wird die Gaußfunktion, um Rechenleistung zu sparen, meistens nur angenähert und mit Hilfe der diskreten Binomialverteilung dargestellt. Dabei werden die Werte der $n\,x\,n$ Filtermatrix mithilfe eines Binoms der Ordnung $n-1$ bestimmt.

Bei einem $3\,x\,3$ Filterkern wird somit ein Binom der 2.Ordnung verwendet.

$$(a+b)^2 = 1a^2 + 2ab + 1b^2 \tag{2.8}$$

Dabei erhalten nun die Kanten der Filtermatrix die Binomialkoeffizienten:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Die Mitte des Kerns erhält nun noch einen Wert, der so gewählt wird, dass in der mittleren Spalte wieder die Binomialkoeffizienten stehen, diesmal aber mit dem Faktor 2. Zudem erhält der Filter, genauso wie der Box-Filter, einen Koeffizienten um sicherzustellen, dass die resultierenden Werte nicht zu groß werden.

$$K = \frac{1}{16} \cdot \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.9}$$

Dank der Separierbarkeit lässt sich die 3x3 Matrix in zwei eindimensionale Vektoren aufteilen. Der Vorteil hierbei ist es, dass die Rechenzeit nochmals reduziert wird, da nun der Rechenaufwand von (N^2) auf $(N+N)$ Rechenschritte gemindert werden kann.

$$K = \frac{1}{16} \cdot \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \frac{1}{4} \cdot \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \cdot \frac{1}{4} \cdot \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \tag{2.10}$$

Für den Prototyp wurde der bereits implementierte Gaußalgorithmus in der OpenCV-Bibliothek verwendet. Dieser wird mit der Funktion „cv2.gaussianblur()“ aufgerufen. Der hier implementierte Algorithmus berechnet die Werte der Filterkerne jedoch mithilfe der Impulsantwort des Gauß Algorithmus.

Dabei wird für die Anwendung auf ein zweidimensionales Bild eine zweidimensionale Form der eindimensionalen Impulsantwort des Gauß Algorithmus verwendet. Diese erhält man durch die Multiplikation beider Richtungen.

$$h(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{x^2}{2\sigma^2}} \tag{2.11}$$

$$h(y) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{y^2}{2\sigma^2}} \tag{2.11}$$

$$h(x) \cdot h(y) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{x^2}{2\sigma^2}} \cdot \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{y^2}{2\sigma^2}}$$

$$h(x,y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.12}$$

Die Standardabweichung σ entspricht dabei dem Radius der Funktion. Die Werte der Filtermaske werden somit von der Funktion in Echtzeit berechnet, sodass verschiedene Filterkerngrößen leicht getestet und umgesetzt werden können.

Zusammenfassend ist der Gaußfilter sehr gut geeignet, um kleinere Strukturen verschwinden zu lassen und somit u.a. Hintergrundrauschen zu reduzieren. Sofern kleinere Objekte und deren Kanten im späteren Verlauf erkennbar bleiben sollen, darf die Größe der Matrix nicht zu groß gewählt werden, da die markanten Merkmale sonst verwischen.

2.2.2.3 Bilateral Filtering

Das Bilateral Filtering [2] ein Verfahren, welches auf einer nichtlinearen Funktion beruht. Dabei basiert der bilaterale Filter grundlegend auf dem Gaußfilter, berücksichtigt jedoch nicht nur den räumlichen Abstand der Pixel, sondern auch die Intensitätsunterschiede der Pixel. Dadurch dass Pixel mit einem geringen Intensitätsunterschied innerhalb des betrachteten Areals eine kleinere Gewichtung erhalten und hohe Intensitätsunterschiede, wie sie bei Kanten vorkommen, höher gewichtet werden, bleiben Kanten nach dem Filtern weiterhin deutlich erkennbar. Die Formel für den bilateralen Filter ist wie folgt definiert:

$$I_p = \frac{1}{W} \sum_{q \in R_p} G_{\sigma_S}(\|p-q\|) \cdot G_{\sigma_I}(|I_p - I_q|) \cdot I_q \tag{2.13}$$

$$W = \sum_{q \in R_p} G_{\sigma_S}(\|p-q\|) \cdot G_{\sigma_I}(|I_p - I_q|) \tag{2.14}$$

p ist hierbei das Ausgabepixel und q ein Nachbarpixel, beide dargestellt als zweidimensionalen Koordinaten. R_P ist ein auf den Pixel p zentriertes Fenster. I_P und I_q enthalten die Farbintensität der beiden Pixel p und q. Die Variable W ist der Gewichtungsfaktor um den ursprünglichen Wertebereich beizubehalten. Der Kern der Funktion besteht aus den beiden Gauß Algorithmen $G_{\sigma_S}(\|p-q\|)$, der räumlichen Gewichtung und $G_{\sigma_I}(|I_p - I_q|)$ der Intensitätsgewichtung der Pixel. σ_S und σ_I sind hierbei die Standardabweichungen der beiden Gauß-Funktionen.

Durch die Einbeziehung der Intensitätsunterschiede werden Kanten mit diesem Filter weniger verwischt, sondern bleiben in ihrer Intensität bestehen. Sofern auch hier die Länge der Matrix nicht einen gewissen Punkt überschreitet, werden kleinere Objekte weiterhin sehr gut detektiert.

Der bilaterale Filter ist ebenfalls bereits in der Bibliothek OpenCV hinterlegt und wird mit „cv2.bilateralFilter()“ aufgerufen.

2.2.3 Sobel Operator

Sobel Operator ist ein Filter zur Kantendetektion, der mit Hilfe der ersten Ableitung der Intensitätswerte, die Kanten eines Bildes hervorhebt. Mit dieser Methode lassen sich Kanten separat in x und in y Richtung verstärken und gleichermaßen Rauschen unterdrücken.

Der Sobel Operator basiert auf dem Gradientenverfahren, [4] welches ein so genanntes Vektorfeld liefert, bei dem ein Vektor in einem Punkt P immer auf die Stelle mit der größten Steigung zeigt.

Unter Ausnutzung der ersten Ableitung eines Signals werden somit Punkte mit der größten Steigung detektiert. Da es sich, wie bereits erwähnt, bei Bildern um ortsdiskrete Signale handelt, wird hierzu die erste Ableitung ins Diskrete approximiert.

$$\text{Erste Ableitung}: \qquad f'(x) = \frac{df}{dx}(x) \tag{2.15}$$

$$\text{Diskrete Approximation:} \qquad \frac{df}{dx}(x) \approx \frac{f(x+1) - f(x-1)}{2} \tag{2.16}$$

Die diskrete Approximation lässt sich mit einer diskreten Faltung realisieren.

$$\frac{f(x+1) - f(x-1)}{2} = 0.5\left(f(x+1) - f(x-1)\right) \tag{2.17}$$

$$0.5\left(f(x+1) - f(x-1)\right) = f(x) * \begin{bmatrix} -0.5 & 0 & 0.5 \end{bmatrix} \tag{2.18}$$

Daraus resultierend erhält man die eindimensionalen Ableitungsfilter in x und in y Richtung für die Erzeugung der Gradientenbilder.

$$G_x = 0.5 \cdot \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \tag{2.19}$$

$$G_y = 0.5 \cdot \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \tag{2.20}$$

Um die Rauschanfälligkeit zu verringern, enthält der Sobel Operator aber noch einen Glättungsfilter, der jeweils orthogonal zur Ableitungsrichtung das Bild glättet. Die vorherigen Filtermatrizen werden mit dem jeweiligen Glättungsfilter gefaltet, als Ergebnis erhalten wir den Prewitt Operator, welchen man als Ursprung des Sobel-Operators ansehen kann.

$$P_x = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.21}$$

$$P_y = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.22}$$

Der Sobel Operator hat nun noch eine stärkere Gewichtung der zentralen Spalte bzw. Zeile, sodass folgende Filtermatrizen den Operator bilden:

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.23}$$

$$K_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.24}$$

Aus den Ergebnissen der beiden Faltungen der Kerne mit dem Originalbild kann im Nachhinein durch Kombination der Teilergebnisse ein Bild erzeugt werden, welches sowohl die Kanten in x als auch in y- Richtung darstellt.

$$K_{ges} = \sqrt{{K_x}^2 + {K_y}^2} \tag{2.25}$$

Dieses Verfahren liefert ein Bild, welches sowohl die Kanten als auch die Richtungen der Kanten deuten lässt. Nachteil bei dieser Anwendung ist, dass die detektierten Kanten noch sehr breit sind, da diese genauso breit sind wie die abgetasteten Steigungsverläufe. Dies hat eine schlechte Kantenlokalisierung zur Folge. Der im späteren Verlauf erläuterte Canny Detektor basiert ebenfalls auf diesen Operator, enthält aber noch weitere Operanten um eine Kantendetektion auf einen Pixel genau zu gewährleisten.

2.2.4 Laplace Filter

Der Laplace Filter [4] ist eine Methode, um Kanten in einem Bild für die spätere Detektion hervorzuheben. Dieser basiert, ähnlich wie der Sobel Operator, auf der Erkennung der Kanten über die Ableitung des Bildes. Hierbei wird aber nicht die erste Ableitung, sondern die zweite Ableitung verwendet.

Maxima und Minima können in einer Funktion über ihre zweite Ableitung detektiert werden und zeigen sich als Nulldurchgang der zweiten Ableitung.

Der Laplace-Operator kommt aus der Vektoranalysis und wird wie folgt definiert:

$$\nabla^2 f(x) = \frac{d^2 f(x,y)}{dx^2} + \frac{d^2 f(x,y)}{dy^2} \tag{2.26}$$

Für die Anwendung auf ein Bild wird der Laplace-Operator ebenfalls diskret approximiert.

$$\begin{aligned} \frac{d^2 f(x,y)}{dx^2} &= \frac{d}{dx}(f(x+1,y) - f(x,y)) \\ &= \frac{d}{dx}(f(x+1,y)) - \frac{d}{dx}(f(x,y)) \\ &= f(x+1,y) - f(x,y) - f(x,y) + f(x-1,y) \\ \frac{d^2 f(x,y)}{dx^2} &= f(x+1,y) - 2f(x,y) + f(x-1,y) \end{aligned} \tag{2.27}$$

Analog erfolgt die Approximation in y-Richtung.

$$\frac{d^2 f(x,y)}{dy^2} = f(x,y+1) - 2f(x,y) + f(x,y-1) \tag{2.28}$$

Führt man die beiden Approximationen zusammen erhält man für den Laplace Operator:

$$\nabla^2 f(x,y) = f(x+1,y) + f(x,y+1) - 4f(x,y) + f(x-1,y) + f(x,y-1) \tag{2.29}$$

Überträgt man dies in die Form einer Filtermatrix erhält man folgenden Filterkern:

$$K = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{2.30}$$

Der Filterkern hebt sowohl horizontale als auch vertikale Kanten hervor, ist jedoch nicht empfindlich für die diagonalen Kanten. [4]

Der Laplace-Filter ist, wie die zuvor genannten Filter, bereits als Funktion in der OpenCV-Bibliothek hinterlegt und wird mit der Funktion „cv2.laplacian()“ aufgerufen.

2.3 Merkmaldetektion

Mit Hilfe von Algorithmen zur Merkmalerkennung in der Bildverarbeitung lassen sich vorhanden Muster in einem Bild erkennen. Hierbei werden markante Punkte gesucht, wie Kanten oder Ecken. Um Kanten in einem Bild zu erkennen werden meist Helligkeitsunterschiede zwischen zwei Pixel gedeutet und beurteilt.

In diesem Kapitel werden verschiedene Algorithmen vorgestellt und die für den Prototyp verwendeten Algorithmen festgelegt.

2.3.1 Canny Edge Detection

Der Canny-Algorithmus [3] ist ein weit verbreiteter Algorithmus zur Kantendetektion. Benannt ist der Algorithmus nach seinem Erfinder John Francis Canny, ein preisgekrönter australischer Wissenschaftler. Er ist eine Kombination aus Filtering und Thresholding. Als Resultat erhält man ein Bild, das lediglich die Objektkanten enthält und erhält somit die perfekte Basis zur Kantenfindung. Der Algorithmus wird in OpenCV mit der Funktion „cv2.Canny()“ aufgerufen und enthält mehrere Bildverarbeitungsschritte.

Jedoch wird vor dem Aufruf in der Regel immer ein Glättungsfilter auf das Bild angewendet. Dieser führt zur Reduzierung des Hintergrundrauschens und im Idealfall zur Schärfung der Kanten. Hierzu eignet sich besonders der bilaterale Filter, oftmals wird jedoch auch der Gaußfilter eingesetzt.

Sobald die Basis für den Canny Algorithmus stimmt, werden mehrere Verarbeitungsschritte innerhalb des Algorithmus ausgeführt:

Als erstes werden die Gradienten im Bild bestimmt. Hierzu wird ein bereits vorgestellter Operator angewendet, der Sobel Operator. Mit Hilfe des Sobel Operators werden die Gradienten und deren Richtungen festgehalten. Wie bereits im Kapitel des Sobel Operators erwähnt, erhält man nach der Anwendung des Operators relativ breite Kanten. Das Ziel beim Canny Algorithmus ist jedoch nur Kanten mit einer Breite von je einem Pixel abzubilden. Um dies zu erreichen werden nicht Maxima entlang der Kanten unterdrückt und nur die lokalen Maxima hervorgehoben. Dieses Verfahren nennt sich non-maximum-suppression.

Abschließend wird noch mit Hilfe des Thresholdingverfahren festgelegt, ab welcher Intensitätsstärke ein Pixel zu einer Kante gehört. Hierzu werden zwei Schwellenwerte, T1 und T2, der Funktion übergeben, wobei das Verhältnis T1:T2 meistens etwa 1:2 beträgt. Der Algorithmus durchsucht das Bild nach einem Pixel dessen Wert größer des Schwellenwerts T2 ist. Beginnend mit diesem Pixel, überprüft der Algorithmus alle Pixel entlang der Kante, ob diese größer als der Schwellenwert T1 sind.

Somit erhält man anschließend ausschließlich Objektkanten, die aus Pixel bestehen deren Intensität größer T1 sind. Durch das Festlegen der richtigen Schwellenwerte, können gezielt unerwünschte Objekte durch den Canny-Algorithmus ausgeblendet werden. Da sich die Intensität der gesuchten Objekte jedoch je nach Anwendungsfall nicht zu den der Hintergrundstrukturen unterscheiden, können im resultierenden Bild noch Kanten von unerwünschten Objekten vorhanden sein. Hierzu muss individuell eine Lösung gefunden werden.

Wie eine solche Lösung konkret aussieht geht aus der späteren Durchführung des Projekts hervor.

2.3.2 Find Contours

Bei der Konturendetektion [3] werden hervorgehobene Kanten zu einer Objektkontur zusammengefasst, um somit schließlich Objekte zu erkennen. Durch die Zuordnung der Konturen zu einem Objekt, lassen sich diese kategorisieren und auswerten.

Die Funktion der OpenCV-Bibliothek „cv2.findcontours()“ ermöglicht es die vorhandenen Kanten in einem Bild zu Konturen zusammenzufassen. Über die Übergabeparameter können verschiedene Modi der Konturerkennung ausgewählt werden.

- **CV_RETR_EXTERNAL**
  - Es werden nur die äußeren Kanten zu Konturen zusammengefasst und in einer Liste gespeichert.
  - Es werden keine Beziehung zwischen den Konturen abgespeichert.

- **CV_RETR_LIST**
  - Es werden alle Kanten, sowohl die inneren, als auch die äußeren Kanten als Kontur erkannt und in einer Liste gespeichert.
  - Es werden keine Beziehung zwischen den Konturen abgespeichert.

- **CV_RETR_CCOMP**
  - Es werden alle Kanten, sowohl die inneren, als auch die äußeren Kanten als Kontur erkannt und in einer Liste gespeichert.
  - Es wird unterschieden zwischen innere und äußere Konturen. Äußere Konturen werden in der ersten Hierarchie Ebene abgelegt und innere in der zweiten.

- **CV_RETR_TREE**
  - Es werden alle Kanten, sowohl die inneren, als auch die äußeren Kanten als Kontur erkannt und in einer Liste gespeichert.
  - Jede Konturebene wird in einer eigenen Hierarchie ebene abgelegt, sodass sich eine Baumstruktur bildet, aus der ersichtlich wird wie

die jeweiligen Konturen zueinanderstehen. Dabei erhalten die äußersten Konturen den Wert 0 und die innersten Konturen den höchsten Wert.

Für die Erkennung der Testobjekte in dem zu realisierenden Prototypen werden lediglich die äußeren Konturen der Objekte benötigt. Dem zufolge wird der Funktion „cv2.findcontours()“ der Parameter „CV_RETR_EXTERNAL“ übergeben. Nachdem Aufruf der Funktion durchsucht der, in der Funktion hinterlegte Algorithmus, das Bild nach weißen Objekten und übergibt dem Anwender eine Python Liste mit der jeweiligen Kontur der Objekte. Anhand dieser Liste lassen sich verschiedene Informationen über die erkannten Konturen abrufen. Es werden unter anderem Informationen wie die Koordinaten und die Größe des abgedeckten Areals eines Objektes übermittelt.
Somit ermöglicht diese Funktion abschließend die Detektion der gesuchten Objekte und liefert alle wichtigen Informationen zur weiteren Auswertung.

3 Hardwareaufbau

Zur Realisierung des Prototyps werden folgende Hardwarekomponenten verwendet. Hierbei getrennt dargestellt in den technischen relevanten Part und dem Aufbau der Laborumgebung.

3.1 Technische Komponenten

3.1.1 Raspberry PI 3 B

Der Raspberry PI 3 B ist das Herzstück des Prototyps. Auf diesem erfolgt die gesamte Software-Implementierung. Es handelt sich hierbei um einen Computer in Größe einer Kreditkarte. Das Modell B besteht aus einem CPU-Core in ARMv6-Architektur mit 700 MHz sowie einem Broadcom Video-Core IV mit H.264 Encoder/Decoder. Zusätzlich enthält der Raspberry Pi ein Arbeitsspeicher mit 512Mbyte.

Um verschiedene Peripheriegeräte anzuschließen hat der Raspberry Pi unterschiedliche Ein- bzw. Ausgänge. (siehe Anhang 2)

3.1.2 Raspberry PI Camera V2.1

Die zur Bildaufnahme verwendete Kamera Raspberry Pi Camera V2.1 ist eine 8-Megapixel Kamera mit einer maximalen Videoauflösung von 1920x1080p. Die Kamera wird über ein 15 poliges Flachbandkabel mit dem Raspberry Pi verbunden.

3.1.3 Beleuchtung

Zur Hintergrundbeleuchtung des Diffusorstoffes wird eine Energiesparlampe (Fotolampe) der Marke ESDDI verwendet. Die Leuchte hat eine Leistungsaufnahme von 85W bei einer Farbtemperatur von 5500K.

3.2 Nicht technische Komponenten

Folgende Materialien werden für den Aufbau des Prototyps verwendet:

- 2 Kästen mit Deckel (Modell: TJENA Firma: IKEA)
    - Modell: TJENA (IKEA)
    - Maße (cm): 30x35x50 (HxBxT)
- Diffusorstoff
- Transparente Folie
- Pappkarton
- Klebeband
- Schrauben (M2x16)
- Muttern (M2)

3.3 Aufbau

Der Prototyp besteht im wesentlich aus zwei übereinanderstehenden Kisten welche durch eine Transparente Folie zueinander getrennt sind. Die Folie verhindert, dass Testobjekte in die obere Kiste gelangen. Die obere Kiste dient dabei als Abstand zwischen, der sich im obersten Deckel befindenden Kamera und dem in der unteren Kiste befindenden Testareal. Somit ist gewährleistet, dass die Kamera den kompletten Raum des Testareals erfasst, indem sich die zu untersuchenden Objekte befinden.

Abbildung 2 : Prototyp

Die Kamera und der Raspberry Pi werden mittels herkömmlichen M2x16 Schrauben und dazu passenden Muttern am Kistendeckel befestigt.

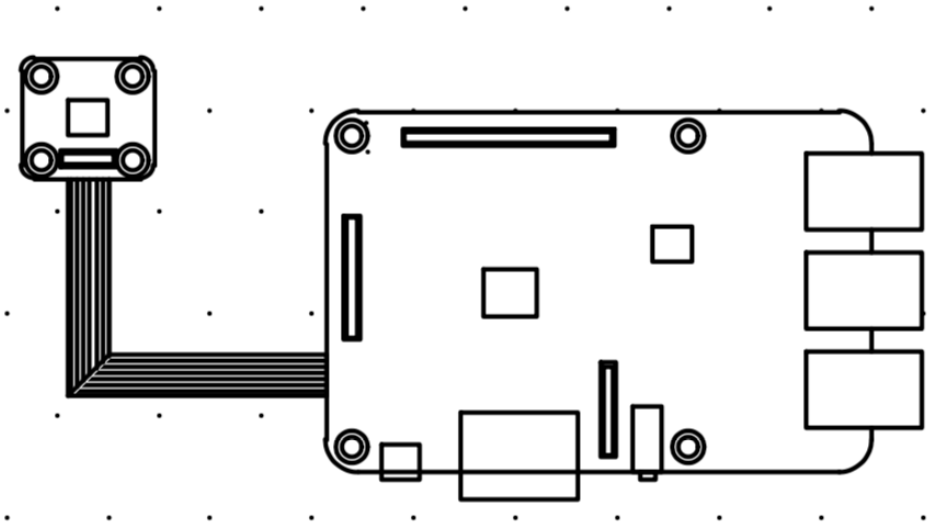


Abbildung 3: Schematische Anbringung Raspberry Pi und Kamera

Der Boden der unteren Kiste ist durch einen lichtdurchlässigen Diffusorstoff ersetzt. Die Beleuchtung des Testareals strahlt entgegen der Kameraausrichtung, sodass ungewollte Schatten vermieden werden, die in der späteren Auswertung zu Fehlern in der Objekterkennung führen könnten. Durch die Anwendung des Diffusorstoffes entsteht eine weiße leuchtende Fläche, die einen guten Kontrast zu den gesuchten Objekten einbringt.

Die Software wird für eine Unterteilung des Testareals in zwei Bereiche vorbereitet. Somit können durch die Unterteilung mittels einer durchlässigen Trennwand zwei Areale geschaffen werden. In denen können anschließend verschiedene Lockmittel untergebracht werden, um diese auf ihre Wirkung auf das Verhalten der Insekten im direkten Vergleich zu untersuchen.

Die Zufuhr der Testobjekte kann durch Abheben des oberen Kartons erfolgen oder durch eine Implementierung einer Seitenklappe. Bei der Realisierung des Prototyps erfolgt die Zufuhr der Insekten über das Abheben des oberen Kartons. Folgend ist eine schematische Zeichnung dargestellt.

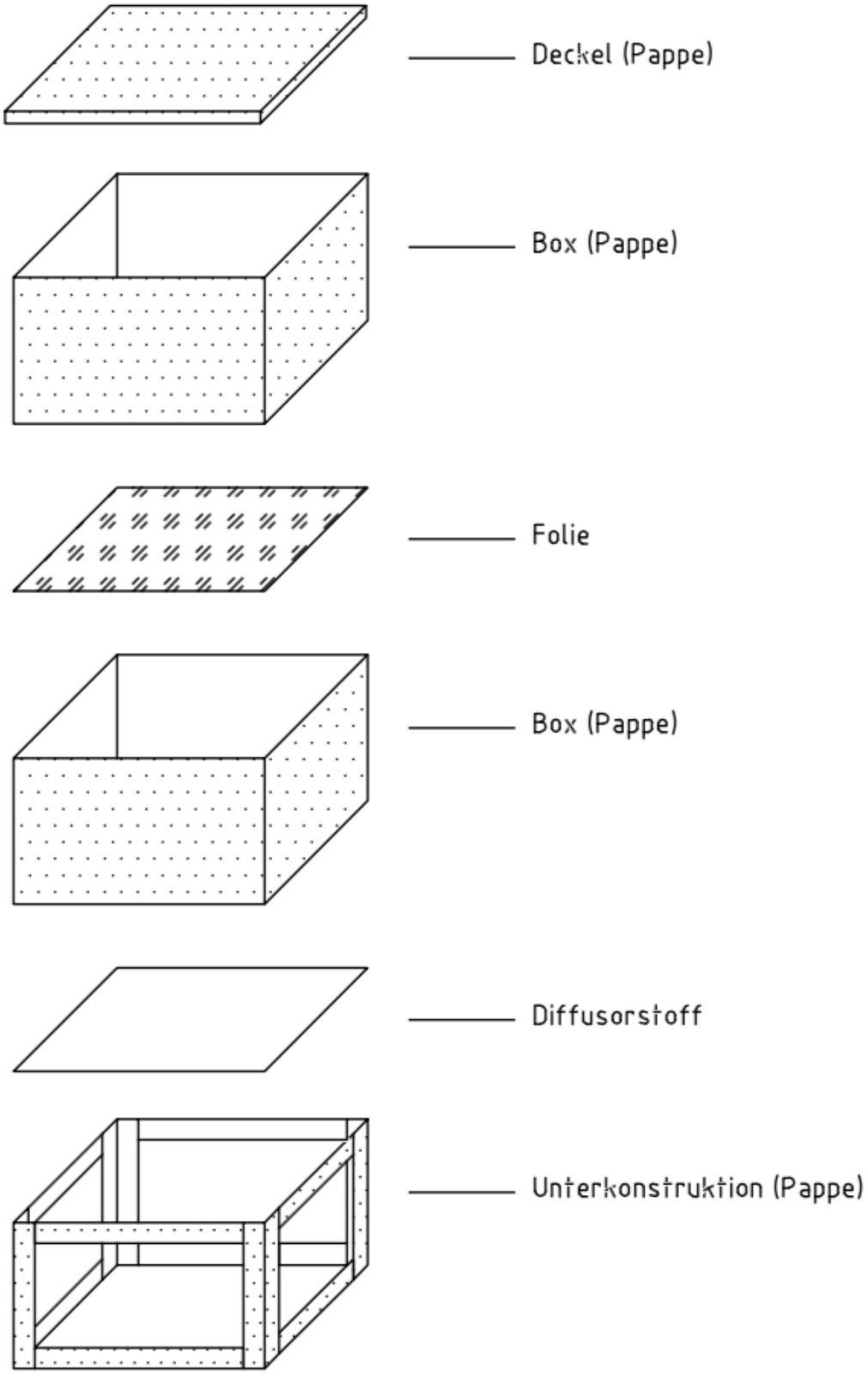


Abbildung 4: Schematische Zeichnung des Prototyps

4 Entwicklung und Tests der Bildauswertealgorithmen

4.1 Einrichtung

Die Realisierung des Prototyps zur automatisierten Zählung von Insekten startet mit der Einrichtung der technischen Komponenten.

Der Raspberry Pi wird mit dem Betriebssystem Raspbian bespielt, welches direkt über die offizielle Website des Herstellers, des Raspberry Pi´s, bezogen wird. Hierzu bietet der Hersteller ein Tutorial zur korrekten Installation und Einrichtung an.

Die Kamera kann direkt über das 15-polige Flachbandkabel mit dem Raspberry Pi verbunden werden und muss lediglich über die internen Einstellungen einmalig freigeschaltet werden.

Für die Bedienung des Raspberry Pi wird ein lokales Netzwerk und ein separater PC bzw. Laptop benötigt. Für die Verbindung wird eine SSH Verbindung eingerichtet. Die Desktopansicht des Raspberry Pi´s erreicht man über einen VNC Viewer. Hierzu wird die IP-Adresse und die festgelegten Nutzerdaten benötigt.

Nach der erfolgreichen Einrichtung des Raspberry Pi´s wird die Programmierumgebung Python implementiert. Hierzu existieren mehrere Einrichtungsanleitungen, in denen ebenfalls bereits die Installation der Softwarebibliothek OpenCV enthalten sind. In diesem Fall wurde eine Anleitung der Webseite „Raspberry Pi Tutorials“ [10] verwendet.

Mit dem Abschluss der Installation der Softwareumgebung ist der Raspberry Pi fertig eingerichtet und zur Implementierung des Programmcodes bereit.

4.2 Testerkennung

4.2.1 Testbildaufnahme

Für die Aufnahme von Testbildern wird der Python Code auf dem Raspberry in einer eigenen Datei geschrieben. Dabei muss die Datei die Endung „.py“ aufweisen. Anschließend kann der Programmcode über das Terminal des Raspberry Pi´s aufgerufen und ausgeführt werden. Hierzu wird das Kommando „python [DATEINAME].py“ verwendet. Für die Nutzung der speziellen Funktionen von OpenCV und anderen Bibliotheken werden die jeweiligen Bibliotheken am Anfang eines Programmcodes eingebunden.

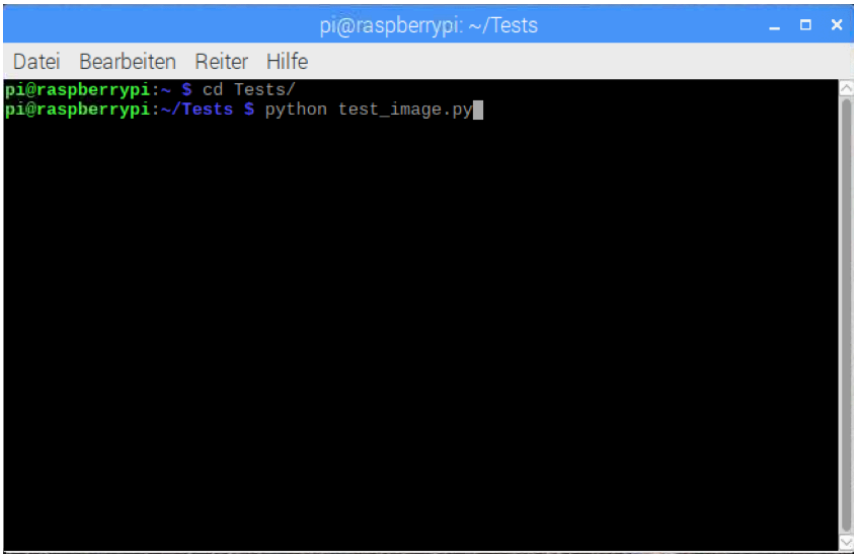


Abbildung 5: Terminalaufruf Aufnahme Testbild

Bei der Aufnahme des ersten Testbildes, handelt es sich um ein Standbild. Hierzu muss die Kamera des Raspberry Pi´s, wie im folgenden Code erkenntlich, aufgerufen werden und das Bild mit Hilfe der OpenCV Bibliothek gespeichert werden. Das folgende Programmbeispiel veranschaulicht zugleich, wie in OpenCV Medien geladen und gespeichert werden.

```python
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
import cv2

# initialize the camera
camera = PiCamera()
rawimage = PiRGBArray(camera)

# allow the camera to warmup
time.sleep(0.1)

# grab image and covert for to display with OpenCV
camera.capture(rawimage,0)
image = rawimage.array

# display the image on screen and wait for a keypress
cv2.imshow("Image", image)
cv2.waitKey(0)

#write image in PNG-File
cv2.imwrite("Testbild.png",image)
```

Listing 1: test_image.py Quellcode

Für die ersten Erkennungsversuche werden aus einem schwarzen Pappkarton kreisförmige Stücke gestanzt. Der Durchmesser eines Pappstücks beträgt 5-6mm. Diese Größe passt in etwa zu den gängigen kleinen Fluginsekten (u.a. Stechmücken, Fruchtfliegen und Alltagsfliegen), die für den Prototyp in Frage kommen. Das resultierende Bild wird folgend als Testbild bezeichnet.

Wie in Abbildung 6 zu erkennen ist die Kamera noch nicht ideal justiert. Für die ersten Erkennungsversuche ist dies jedoch nicht ausschlaggebend.

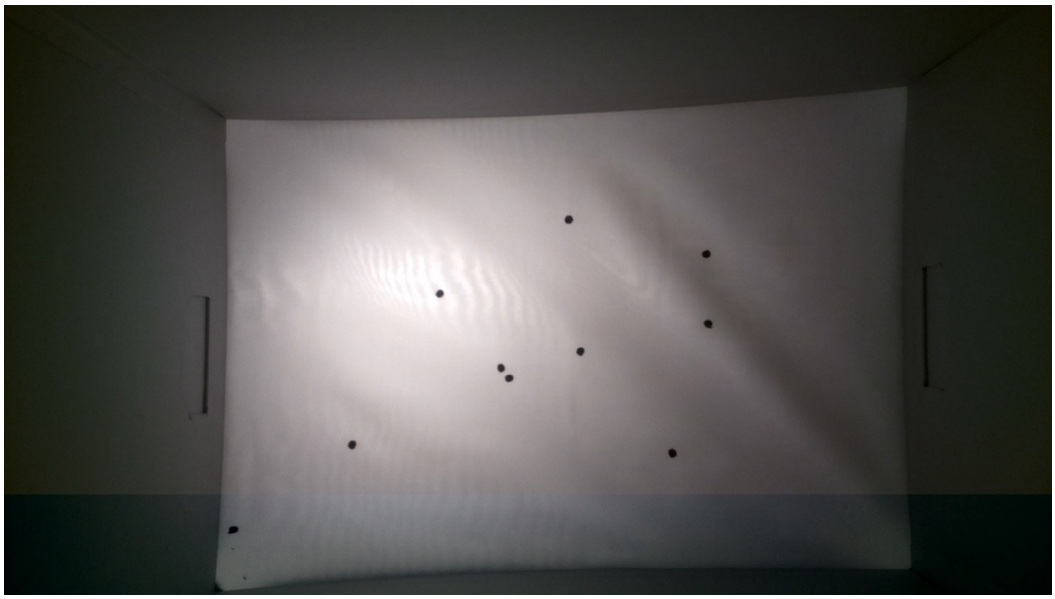


Abbildung 6: Testbild

Die Testbilder werden zunächst immer als Grauwertbild für die weitere Verarbeitung benötigt, dies geschieht über eine einfache Konvertierung mit Hilfe der OpenCV Bibliothek. Das Grauwertbild wird im Anschluss für die Objekterkennung vorverarbeitet.

4.2.2 Bildvorverarbeitung (Testbild)

In diesem Kapitel werden die unterschiedlichen Verfahren aus dem Kapitel Bildvorverarbeitung auf das Testbild angewendet und validiert.

4.2.2.1 Thresholding (Testbild)

Um ein gutes Thresholding Ergebnis zu erreichen, ist es von Vorteil sich zunächst das Histogramm des Grauwertbildes anzusehen. Das Histogramm eines Bildes stellt den Anteil eines Grauwertes im Bild graphisch dar. Die Grauwerte eines Bildes liegen dabei zwischen den Werten 0 und 255, wobei einer niedriger Grauwert ein dunkler Pixel und ein hoher Grauwert ein heller Pixel bedeutet.

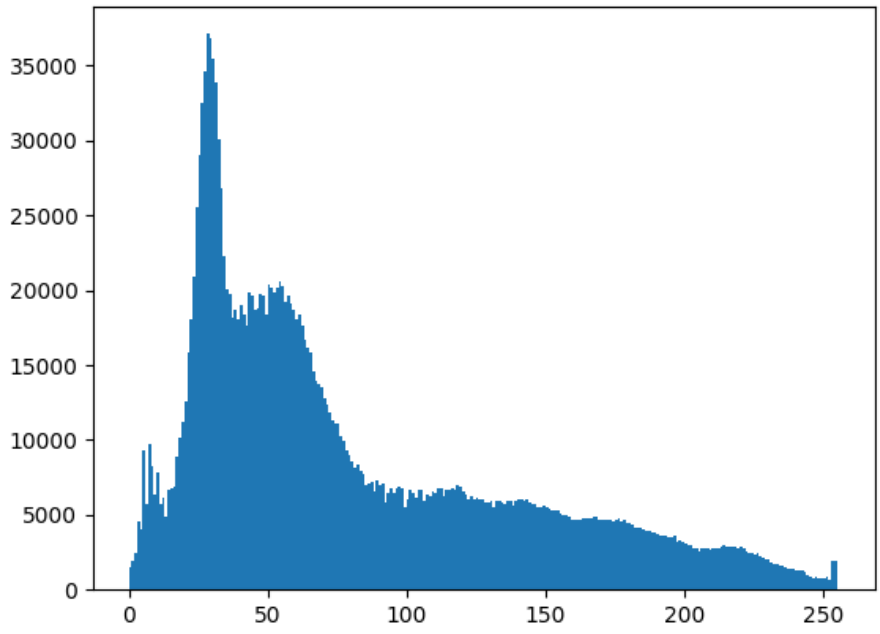


Abbildung 7: Histogramm (Testbild)

Das Histogramm zeigt einen hohen Anteil von dunklen Pixeln. Wenn nun mehrere Schwellenwertfenster auf das Testbild gelegt werden, welche jeweils 75 Grauwerte beibehalten, beginnend mit den niedrigen Grauwerten, erhält man folgende Grafiken.

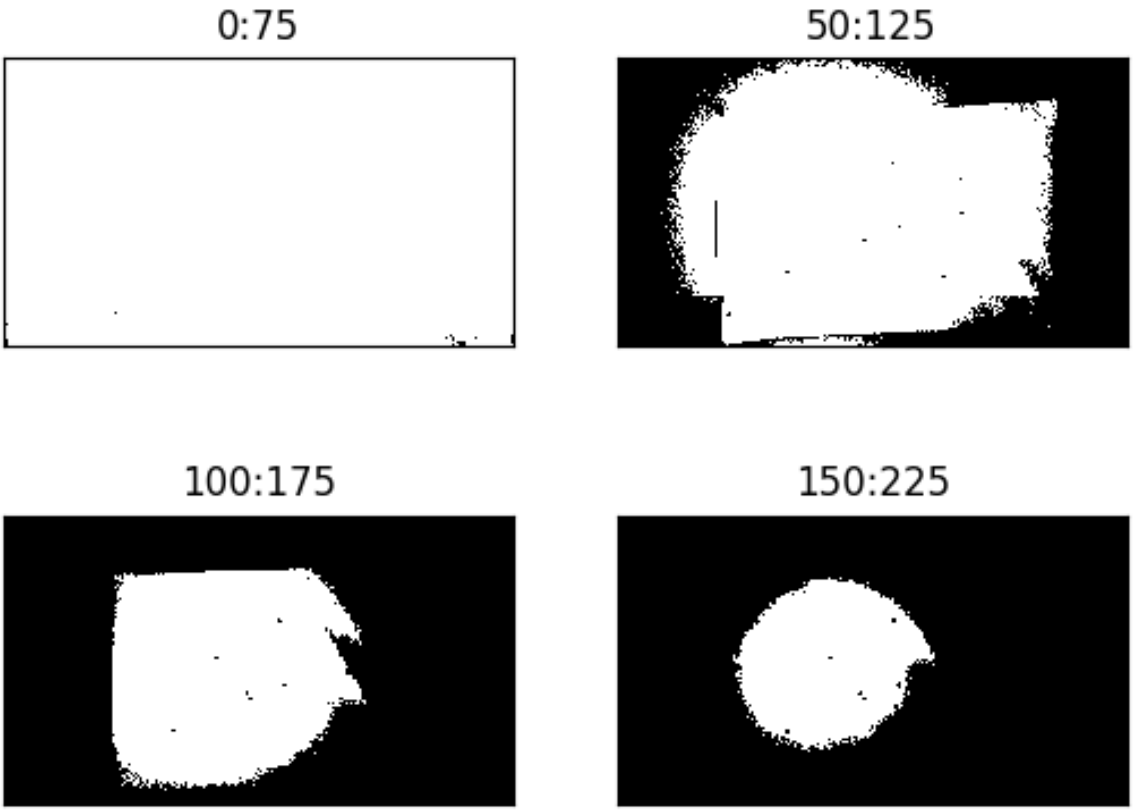


Abbildung 8: Thresholding (Testbild)

Das Ergebnis des Thresholding Verfahrens ist, wie in der obigen Abbildung zu erkennen, eher schlecht. Durch die starken Helligkeitsverläufe lassen sich die Punkte schlecht hervorheben ohne, dass die unterbeleuchteten Kistenränder ebenfalls dargestellt werden.

Das Thresholding Verfahren ist dem zufolge, als erste Anwendung für eine Bildvorverarbeitung, bei diesem Prototyp nicht zu empfehlen. Um ein besseres Ergebnis zur Objekterkennung zu erreichen werden im nächsten Kapitel verschiedene Filter angewendet.

4.2.2.2 Filtering (Testbild)

Zunächst wird die Anwendung des Box Filters veranschaulicht. Um den Effekt des Filters zu verdeutlichen werden vier verschiedene Filterkerngrößen dargestellt.

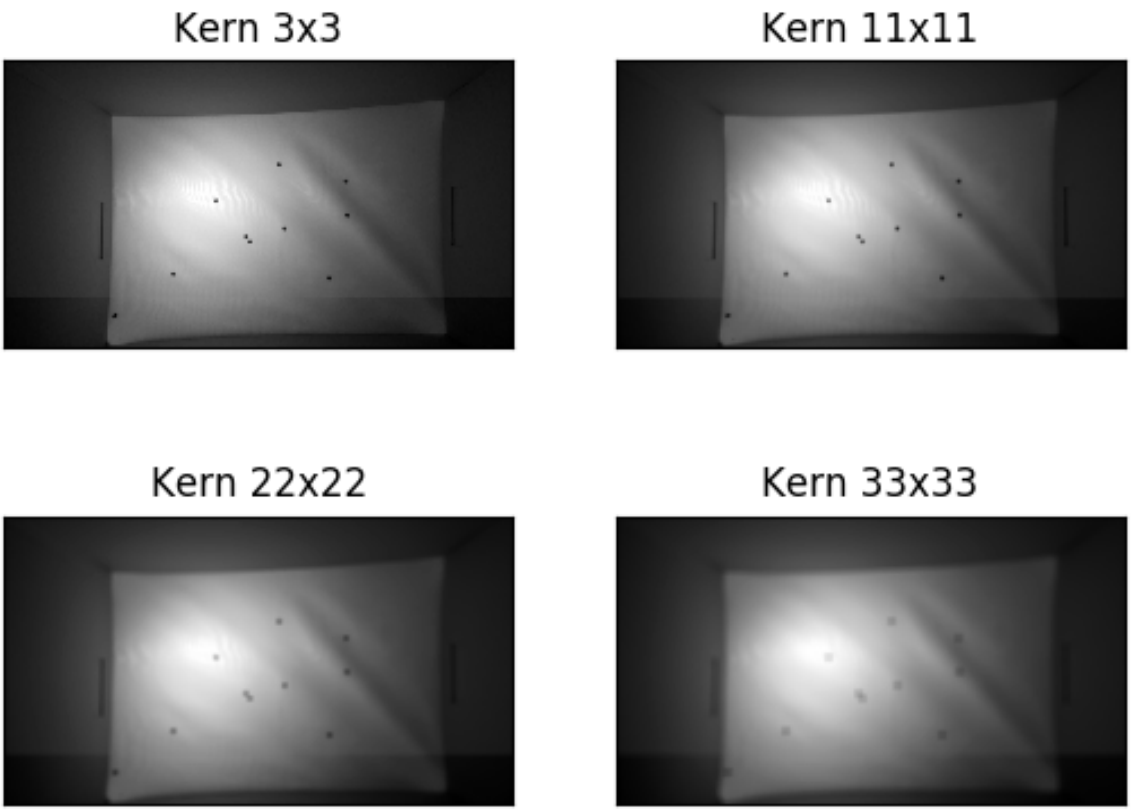


Abbildung 9: Box Filter (Testbild)

Beim Boxfilter werden die Punkte im Testareal mit steigender Größe des Filterkerns sehr verwaschen und heben sich nur wenig vom Hintergrund ab. Dies und die Tatsache, dass die Kanten sehr unscharf werden führt zu einer schlechten Objekterkennung.

Bei der nächsten Abbildung handelt es sich um die Anwendung des Gauß Filters. Hierbei werden ebenfalls verschiedene Kerngrößen dargestellt. Die Standardabweichung σ wird bei der OpenCV Funktion des Gauß Algorithmus automatisch anhand des Filterkerns berechnet.

Durch die höhere Gewichtung der zentralen Pixel ist das Ergebnis (siehe Abbildung 10) deutlich besser als das des Boxfilters. Die Testpunkte heben sich, selbst bei größeren Filterkernen noch deutlich vom Hintergrund ab und verwischen nicht ganz so stark.
Um die gesuchten Objekte jedoch nicht zu sehr zu verwischen wird eine Filtermatrix, die zwischen einer 3x3 und 11x11 Matrix liegt, empfohlen.

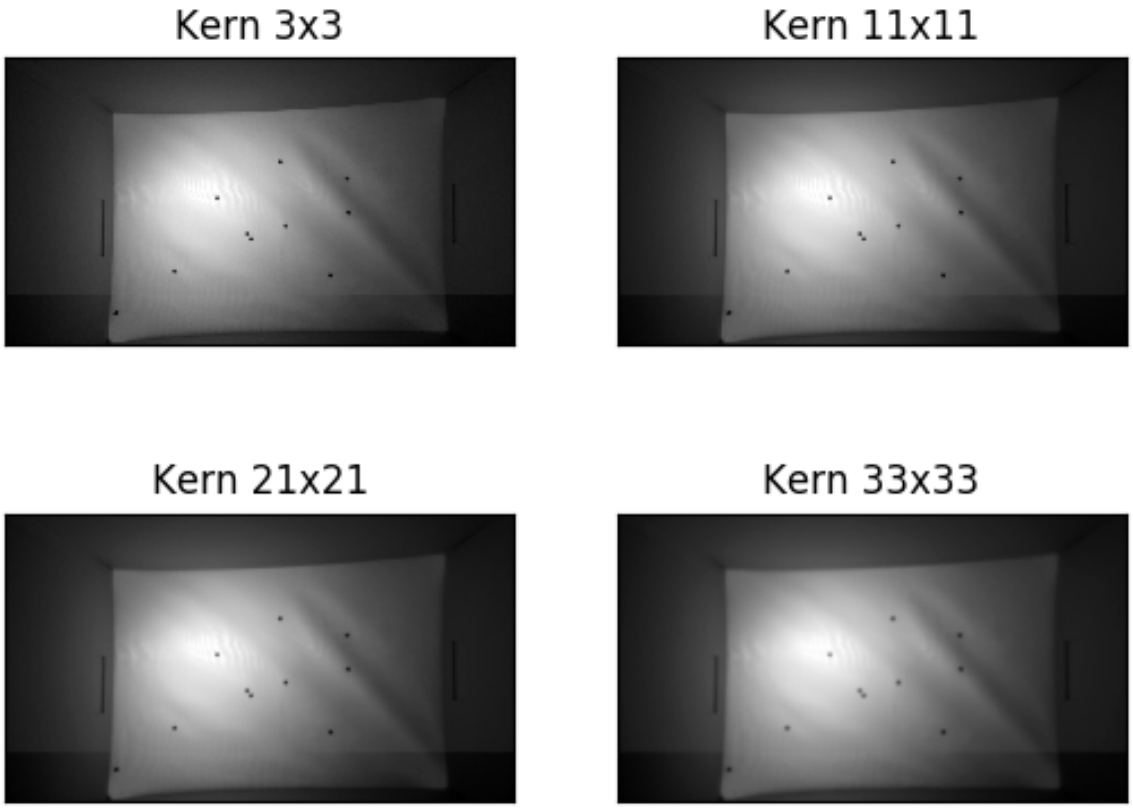


Abbildung 10: Gauß Filter (Testbild)

Der nächste Filter ist der Filter mit der höchsten Erwartung. Der bilaterale Filter verspricht, wie bereits erwähnt eine Glättung des Bildes während die Objektkanten weiterhin bestehen bleiben. Jedoch weist die Abbildung 11 auf den ersten Blick keinen deutlichen Unterschied zu der Abbildung 10 des Gaußfilters auf. Erst bei der Kantenerkennung mit den Insekten, stellte sich heraus, dass die Testobjekte, bei der Anwendung des Gaußfilters, weitestgehend schlechter erkannt wurden.

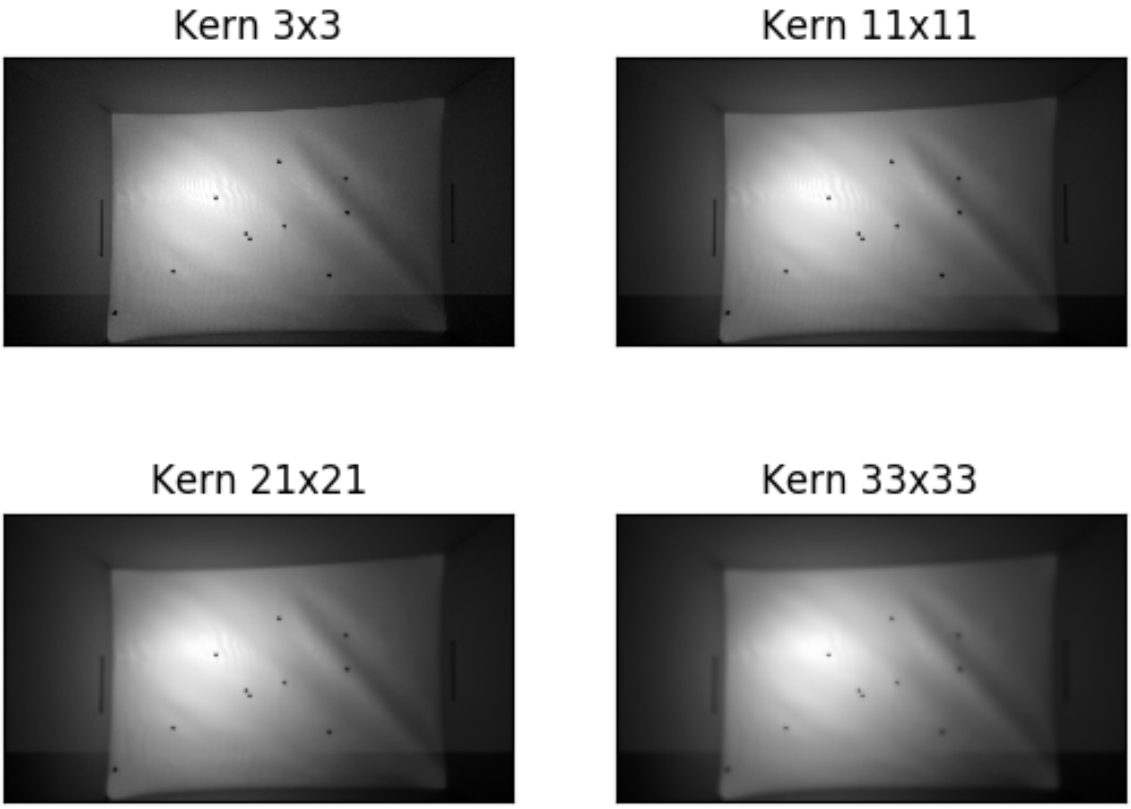


Abbildung 11: Bilateral Filter (Testbild)

Nach der Anwendung des Bilateralen Filters, mit einer 9x9 Matrix, ist das Testbild für die weitere Merkmaldetektion vorbereitet.

4.2.3 Merkmaldetektion (Testbild)

Wie in den Grundlagen erläutert werden bei der Merkmaldetektion verschiedene Algorithmen zur Objekterkennung angewendet. Da die Canny Edge Detection das beste Resultat in der Kantenhervorhebung liefert wurde diese für die Realisierung des Prototypens gewählt. Im Nachgang werden die Objekte mit der Konturenfindung identifiziert.

4.2.3.1 Canny Edge Detection (Testbild)

Die nächste Abbildung zeigt das Resultat der Canny Edge Detection. Die Kanten der Testpunkte wurden hierbei, über das Gradientenverfahren bestimmt und dann über ihre Maxima erkannt. Die Abbildung enthält nun lediglich die hervorgehobenen Kanten und ist somit die optimale Grundlage, um mit Hilfe der Konturenfindung die Objekte zu detektieren und zu zählen.

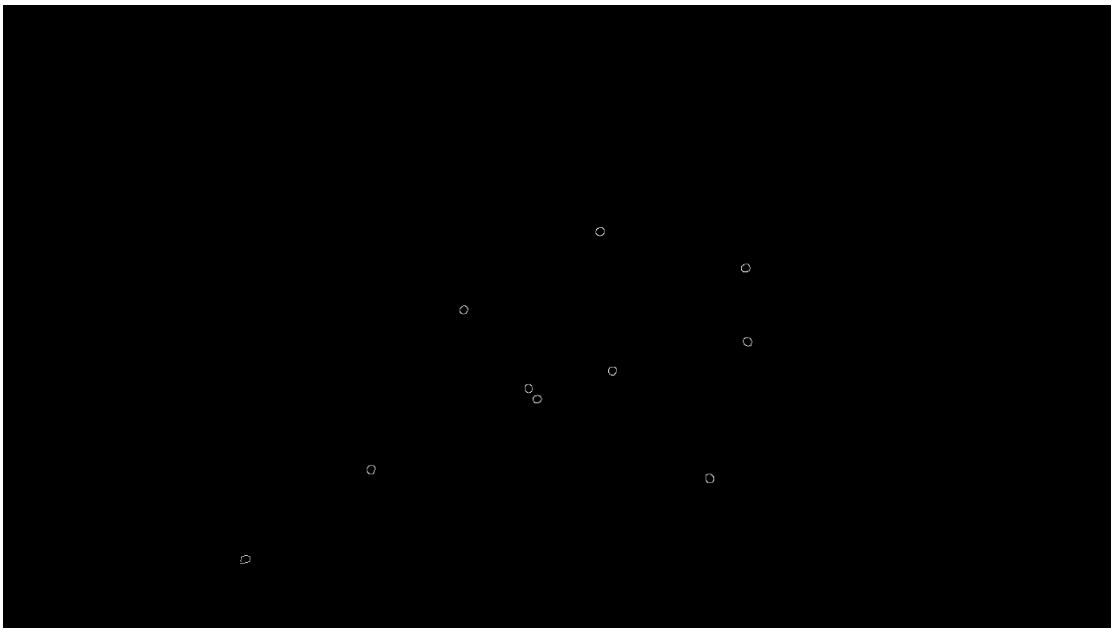


Abbildung 12: Canny Edge Detection (Testbild)

4.2.3.2 Find Contours (Testbild)

Die Konturen werden mit der bereits vorgestellten Funktion cv2.findContours() erkannt. Die Umrahmung der einzelnen Punkte erfolgt mittels einer weiteren, in der OpenCV Bibliothek enthaltenen, Funktion erreicht. Die Funktion cv2.rectangle() erzeugt hierbei farbige Rechtecke um eine, der Funktion übergebenen, Koordinate. Die Koordinaten der einzelnen Punkte erhält man über die Funktion cv2.boundingRect(), welche auf das resultierende Array der Konturenfindung angewendet wird.

In der folgenden Abbildung wurden die eben erwähnten Funktionen angewendet. Das Resultat ist eine erfolgreiche Erkennung und Zählung der Testpunkte.

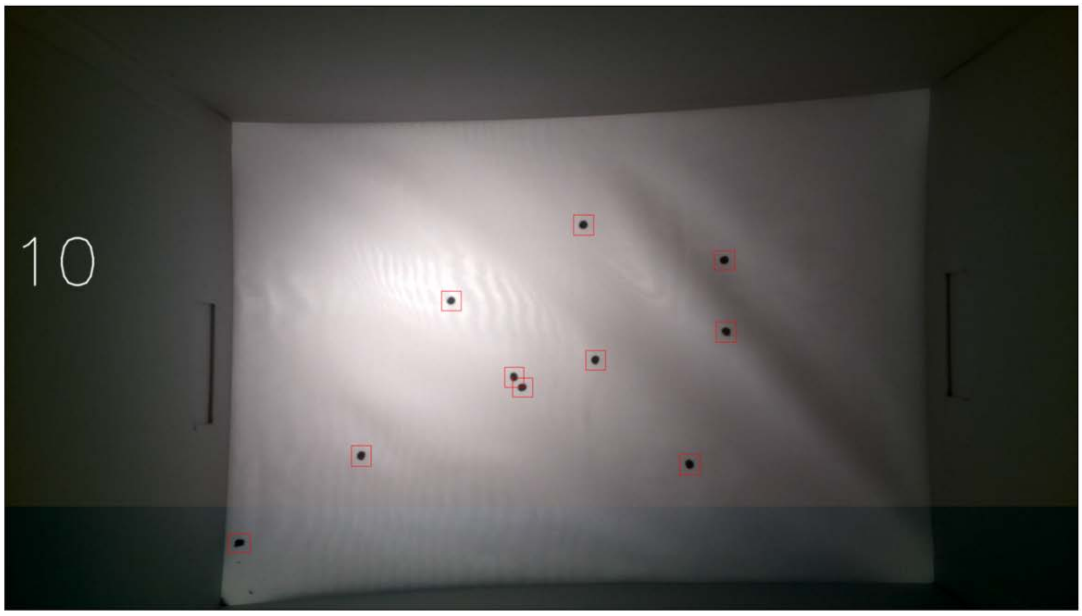


Abbildung 13: Objekterkennung und Zählung (Testbild)

4.3 Insektenerkennung

Nach der erfolgreichen Erkennung von Objekten in dem aufgenommenen Testbild erfolgt die Implementierung der automatischen Erkennung der Insekten.

4.3.1 Insektenauswahl

Die Auslegung des Prototypens war ursprünglich auf die Erkennung von Stechmücken angedacht. Da Stechmücken nur zu bestimmten Jahreszeiten auftreten und während der Bearbeitungszeit keine Exemplare für die Validierung des Prototyps zur Verfügung standen, wird für die weitere Durchführung auf eine andere Insektengruppe zurückgegriffen.

Als Testobjekte wurden nun Stubenfliegen (lat. musca domestica) aus dem Lebendfutterbereich des Tierfachhandels verwendet. Bei den Fliegen handelt es sich um flugunfähige Fliegen, mit einem durchschnittlichen Durchmesser zwischen 0,4 und 0,6cm. Der durchschnittliche Durchmesser ähnelt den der Stechmücken und eignet sich daher gut als Alternative. Sie weisen gerade in den ersten Tagen nach dem Schlüpfen eine hohe Aktivität auf und bieten somit die Möglichkeit, die Erkennung von bewegten Objekten mittels des zu realisierenden Prototyps, zu testen.

4.3.2 Bildaufnahme

Für die Veranschaulichung der Insektenerkennung wird in diesem Kapitel auf ein Ausgangsbild zurückgegriffen, welches lediglich einen Ausschnitt eines aufgezeichneten Videos zeigt. Das Ausgangsbild steht somit beispielhaft für eine Vielzahl von hintereinander folgenden Bildern aus denen eine Erkennung des kompletten Videos abgeleitet werden kann. Das aufgezeichnete Video wird, mit Hilfe eines abgewandelten Programmcodes aus dem Kapitel Testbild, mit einer Bildrate von 20fps aufgenommen (Siehe Anhang 3).

Wie im Ausgangsbild (Abbildung 14) zu erkennen wurde die Kamera und die Beleuchtung für die Aufnahmen nochmals nachjustiert. Der obere Karton, der für den Abstand zwischen dem Testareal und der Kamera sorgt, ist für die Größe des Testareals dennoch nicht ausreichend Hoch. Für die Umsetzung des Prototypens in späteren Studien wird empfohlen diesen in der Höhe zu verlängern.

Abbildung 14: Ausgangsbild mit Fliegen

Das Ausgangsbild enthält neun Exemplare der Stubenfliege, welche es zu erkennen gilt.

Die Testobjekte halten sich sowohl auf dem Boden als auch an den Wänden des Testareals auf. Auf Grundlage der angewendeten Bildverarbeitungsschritte auf das Testbild, werden nun auf das Ausgangsbild die verschiedenen Filter und Merkmaldetektionen angewendet.

4.3.3 Filtering (Ausgangsbild)

Für die Bildvorverarbeitung wurde sich für die Anwendung des bilateralen Filters entschieden. Wie bereits in der Anwendung auf das Testbild erläutert, glättet dieser das Ausgangsbild unter Berücksichtigung der Objektkanten. Dabei wird ein Filterkern mit einer 3x3 Matrix angewendet, da ein größerer Filterkern in diesem Fall die Insekten verwischt hätte.

4.3.4 Merkmaldetektion (Ausgangsbild)

Nach der erfolgreichen Filterung wird nun auf das gefilterte Bild die Kantenhervorhebung, mittels des Canny Algorithmus, durchgeführt. Als Ergebnis erhält man nun ein ähnliches Kantenbild, wie bei den Testaufnahmen.

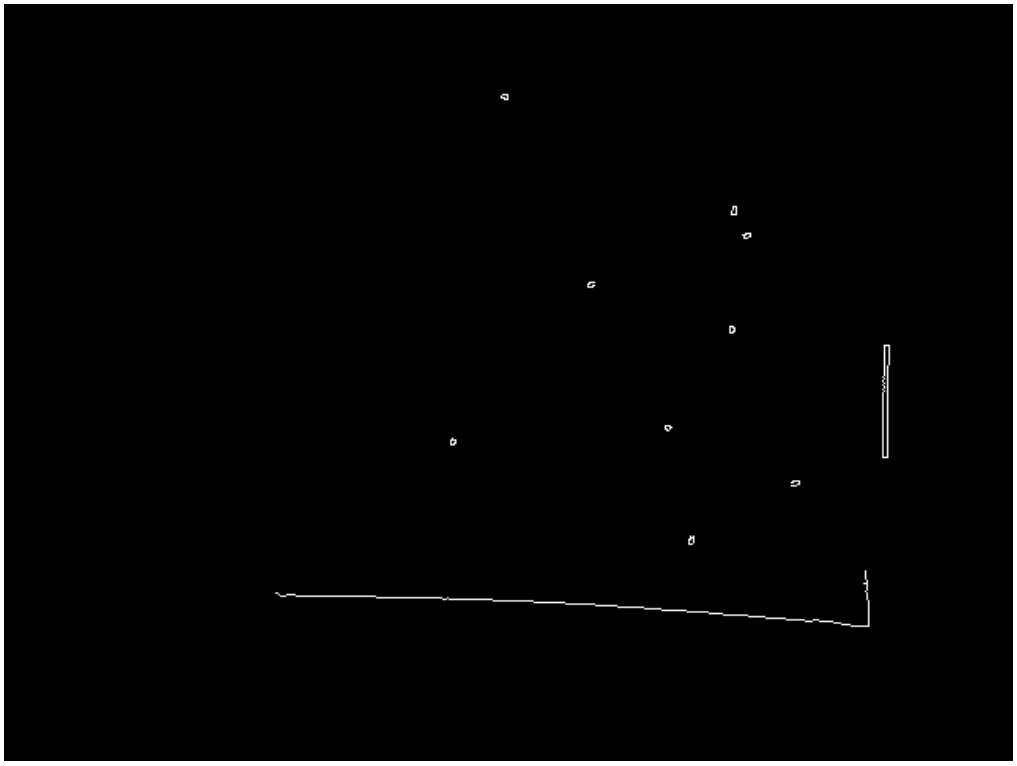


Abbildung 15: Ausgangsbild mit Fliegen (Edges)

Die Schwellenwerte für den Canny Algorithmus werden automatisch, anhand der Mittelwerte der vorhandenen Grauwerte, ermittelt.

Trotz der zu vorigen Filterung sind nach dem Canny Algorithmus noch Kanten im Bild enthalten, die nicht zu den gesuchten Testobjekten gehören. Dies ist dadurch zu erklären, dass zum einen der Filterkern kleiner, als bei den Testaufnahmen, gewählt wurde und dass die Beleuchtung, durch eine andere Positionierung der Leuchte, deutlich ausgeprägter ist. Die bessere Ausleuchtung hat den Vorteil, dass sich die Objekte deutlicher vom Hintergrund abheben, allerdings auch die Folge, dass eine Konturenfindung, wie sie bei dem Testbild erfolgte, zu einer fehlerhaften Erkennung führte.

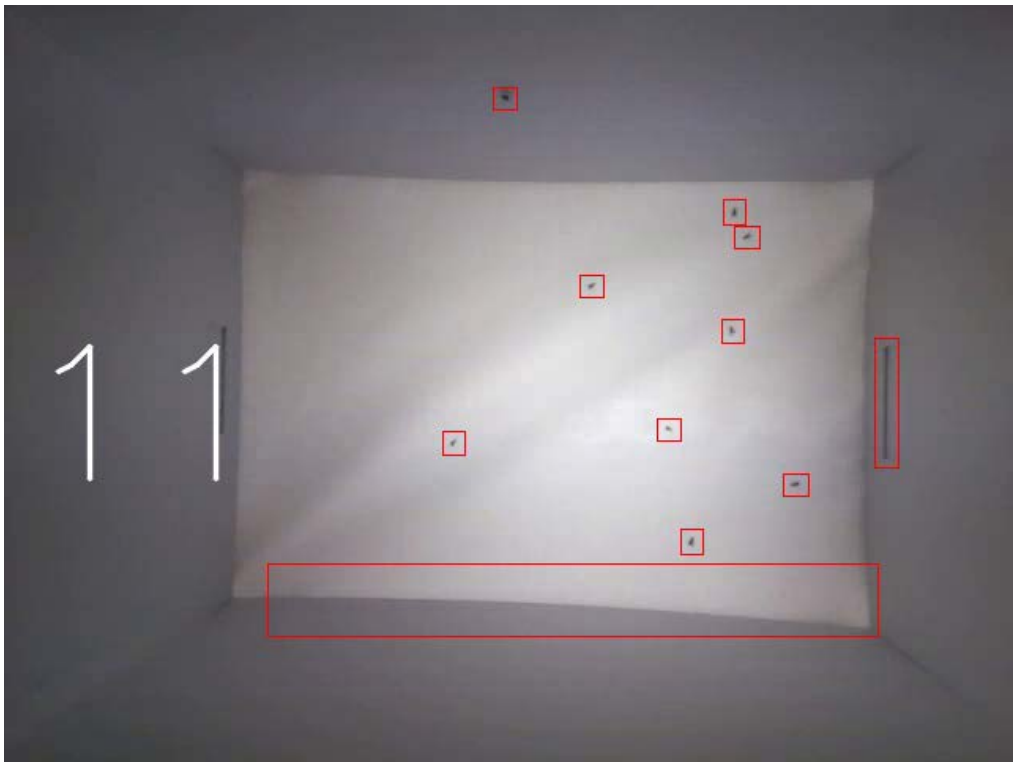


Abbildung 16: Ausgangsbild mit Fliegen (mit fehlerhaften Erkennung)

Wie aus der Abbildung 16 zu erkennen, wurde das ehemalige Loch, welches als Tragegriff für den Karton genutzt wurde und der Übergang zwischen Diffusorstoff und Karton erkannt. Diese fehlerhaften Erkennungen lassen sich entweder durch eine mechanische Anpassung beheben, oder durch eine kleine Anpassung im Programmcode. Für die Behebung der fehlerhaften Erkennung wurde der Code dahin gehend modifiziert, dass Konturen, die eine bestimmte Größe über oder

unterschreiten nicht mehr nach der Konturenfindung eingerahmt und gezählt werden.

Hierzu müssen die Parameter im Programmcode individuell angepasst und auf das gesuchte Objekt abgestimmt werden. Da im Testareal nur wenige unterschiedlich Konturengrößen existieren, ist eine Ermittlung der richtigen Begrenzungen ein kurzweiliges Problem. In der anschließenden Abbildung erhalten wir nun eine korrekte Erkennung der neun Exemplare.

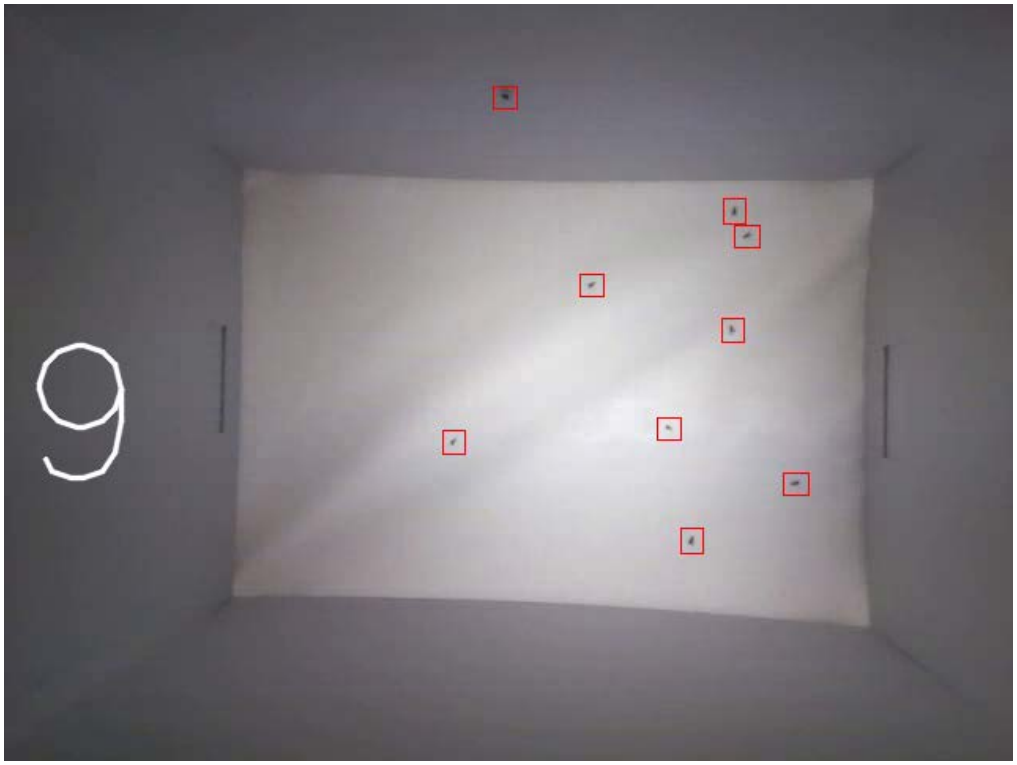


Abbildung 17: Ausgangsbild mit Fliegen (mit korrekter Erkennung)

4.4 Graphische Benutzeroberfläche (GUI)

Für die Vereinfachung der Bedienbarkeit des Prototypens, wurde eine graphische Benutzeroberfläche (GUI) entwickelt, wie sie in Abbildung 18 zu sehen ist.

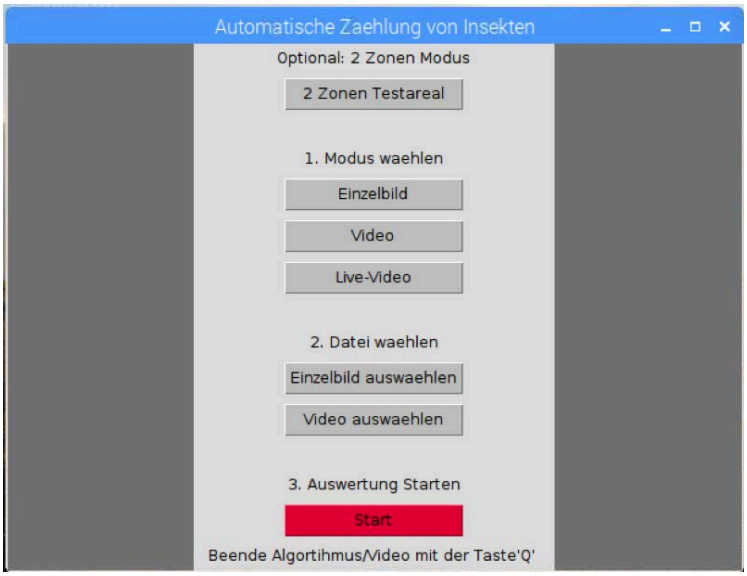


Abbildung 18: Graphische Benutzeroberfläche

Die GUI ermöglicht es zwischen verschiedenen Modi auszuwählen, um somit unterschiedliche Studien durchzuführen. Zum einen besteht die Möglichkeit, über den Button mit der Bezeichnung „Einzelbild“ den Modus einzustellen, um ein einzelnes Bild auszuwerten. Das zur Auswertung benötigte Bild, wird mittels des Buttons „Einzelbild auswählen“ über einen einfachen File-Dialog geladen.

Der Modus „Video“ ermöglicht es, ein zuvor aufgenommenes Video auszuwerten. Das dafür benötigte Video muss vorher separat, mit dem beigelegten Programm, aufgenommen und über den Button „Video auswählen“ aufgerufen werden.

Der dritte Modus enthält die Live-Video-Funktion, bei der direkt auf die Pi-Kamera zugegriffen wird um eine Auswertung über das aktuelle Live-Bild zu erhalten. Die Besonderheit bei den beiden Modi mit Videofunktion ist es, dass während der Auswertung die gesuchte Konturgröße über eine Trackbar geändert werden kann. Dies ist vor allem hilfreich bei der ersten Analyse von neuen Testobjekten. Die übergebenen Größen müssen in Pixel angegeben werden.

Abbildung 19: Trackbar Konturgröße

Die Konturgrößen im Einzelbild Modus müssen zuvor im Programmcode angepasst sein.

Alle Modi werden, nach erfolgter Auswahl, über den Start-Button gestartet. Dabei schließt sich die GUI und der Algorithmus wird ausgeführt. Während der Algorithmus die Bilder auswertet, wird das jeweilige Bild mit den erkannten Testobjekten angezeigt. Die Beendung des Algorithmus erfolgt über die Tastatur mit der Taste „Q“ oder nach Ende des Videos. Anschließend wird dem Nutzer die Auswertung offengelegt.

Zu jedem Modus besteht die Möglichkeit, die Unterteilung des Testareals in zwei Zonen zu wählen. Hierzu muss der entsprechende Button mit ausgewählt werden. Eine korrekte Auswahl wird durch farbliche Hinterlegung der jeweiligen Buttons verdeutlicht.

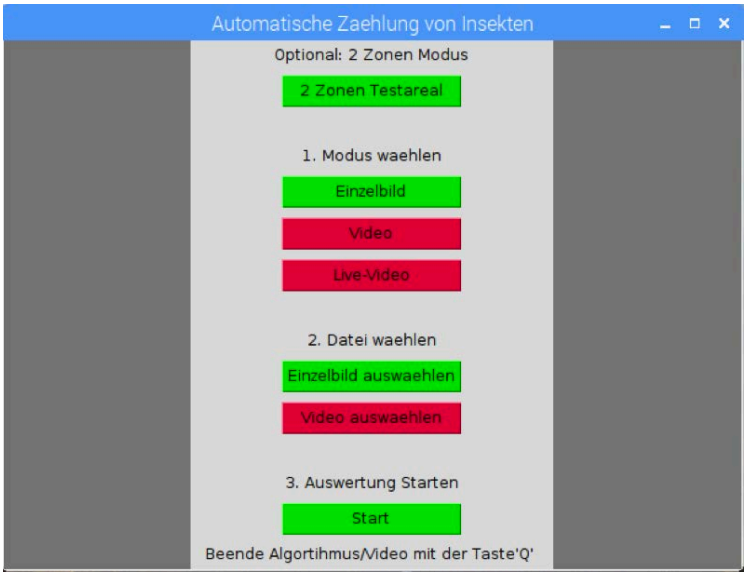


Abbildung 20: Graphische Benutzeroberfläche (mit korrekter Auswahl)

Für die korrekte Anwendung wurden mehrere Fehlermeldungen implementiert. Bspw. wird bei dem Versuch im Modus „Einzelbild“ ein Video zu laden, eine Fehlermeldung gezeigt, welche dem Anwender auffordert zuvor den richtigen Modus zu wählen.

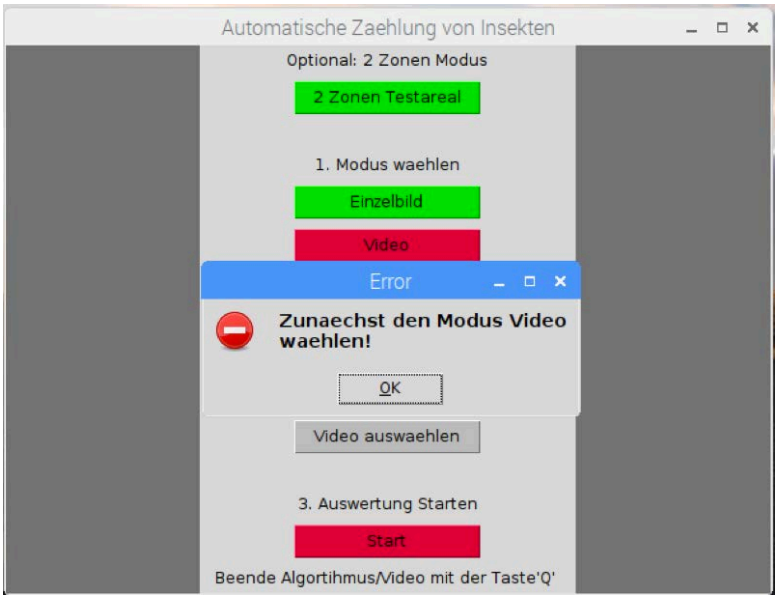


Abbildung 21: Graphische Benutzeroberfläche (mit falscher Auswahl)

5 Testergebnisse zur Insektenzählung

Für die weitere Auswertung der Insektenaktivität kann der Prototyp noch erweitert werden. Durch das Einfügen einer Insektendurchlässigen Trennwand in der Mitte des Testareals, kann das Areal in zwei Zonen unterteilt werden. Dies ermöglicht eine Untersuchung von zwei verschiedenen Lockstoffen und deren Einfluss auf die Insekten.

Im Programm wurde eine zwei Zonen Variante hinterlegt, durch die das Areal in zwei Testberieche getrennt werden kann. Wie in der Abbildung 22 zu erkennen wird eine virtuelle Linie gezogen und den beiden Bereichen ein separater Zähler zugeordnet. Visuell wird die Zuordnung der Insekten durch unterschiedlich farbige Rechtecke erkenntlich.

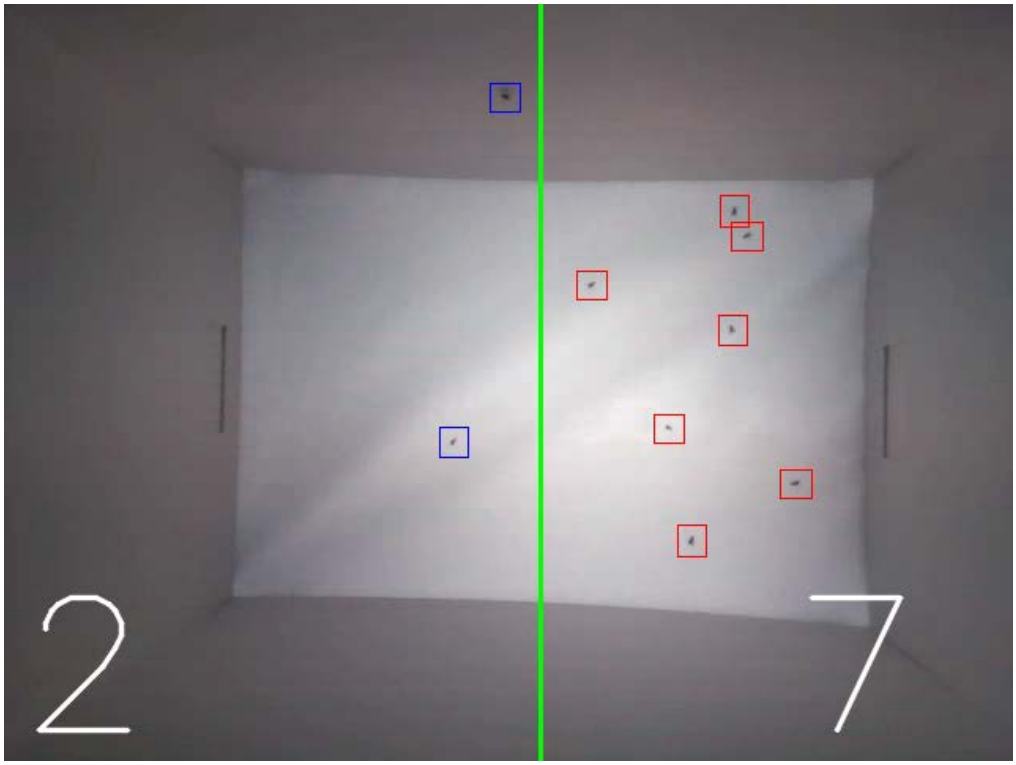


Abbildung 22: Ausgangsbild mit Fliegen (2-Zonen Erkennung)

In diesem Fall sind in der linken Hälfte zwei Exemplare und in der rechten Hälfte sieben Exemplare der Stubenfliegen erkannt wurden. Die Zähler werden mit jedem Bild aktualisiert. Die Positionen der einzelnen Insekten werden somit bei einer Bildrate von 20fps insgesamt zwanzig Mal pro Sekunde ermittelt. Für die

Auswertung ist jedoch die Information ausreichend, wie viele Insekten sich pro Sekunde in der jeweiligen Hälfte aufhalten. Daher wurde für die graphische Auswertung, die erkannten Objekte über 20 Bilder gemittelt und graphisch dargestellt. Dies hat den positiven Nebeneffekt, dass eventuelle Fehlerkennungen von anderen Kanten, die für wenige Millisekunden auftreten, weitestgehend nicht berücksichtigt werden.

Nachdem die Auswertung des aufgezeichneten Videos gestartet wurde, wird das ausgewählte Video, dem Anwender, mit den aktuellen Zählerständen (siehe Abbildung 22) angezeigt. Darauf folgend werden automatisch zwei Grafiken geöffnet, die die Anzahl der Testobjekte über die aufgenommene Zeit darstellen.

In der folgenden Grafik wurde ein Video über knapp 2 Minuten ausgewertet.

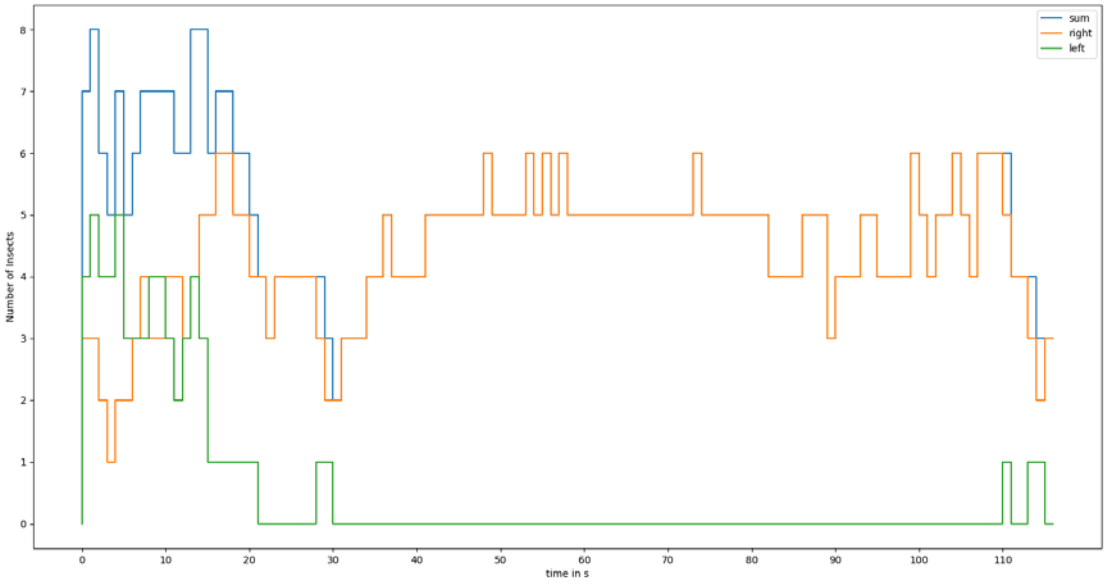


Abbildung 23: Auswertung über Zeit Grafik 1

In der Auswertung wird zum einen die Gesamtanzahl der erkannten Testobjekte angezeigt, als auch die jeweiligen Anzahlen der beiden Testareale. Mit Hilfe der Auswertungsgrafik kann validiert werden, wie viele Testobjekte sich im jeweiligen Areal aufgehalten haben. Die Gesamtanzahl zeigt hierbei, ob über die gesamte Zeit die gleiche Anzahl an Objekten erkannt wurden. Eine Anhebung über einen kürzeren Zeitraum, könnte auf eine Erkennung einer zusätzlichen Kante hinweisen. Eine kürzere Senkung der Gesamtanzahl kann auf eine kurze Überlagerung zweier Testobjekte hindeuten.

Die vielen Senkungen und Erhöhungen der Gesamtanzahl lassen sich derzeit auf die Problematik zurückführen, dass die Testobjekte die Möglichkeit haben sich außerhalb des aufgezeichneten Bereichs zu bewegen. Dies kann durch eine mechanische Anpassung des Prototyps behoben werden. Insbesondere der Kameraabstand muss hierzu angepasst und vergrößert werden.

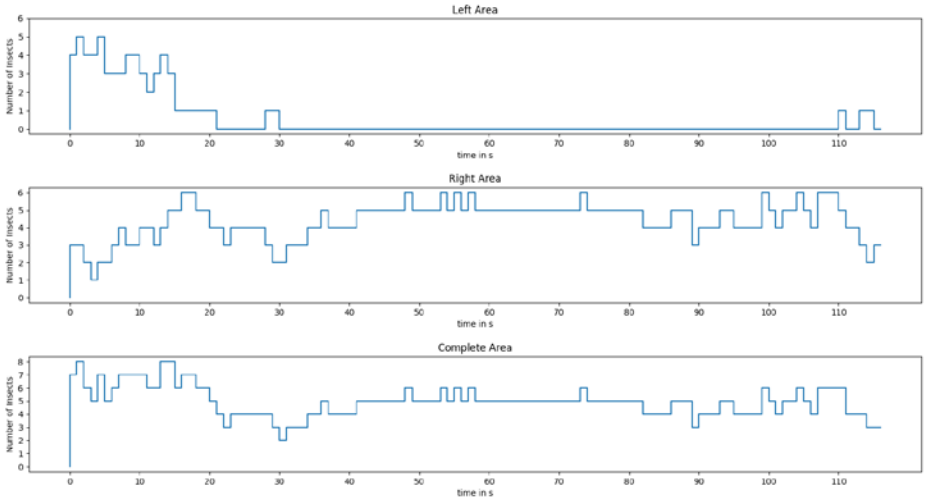


Abbildung 24: Auswertung über Zeit Grafik 2

Die ausgegebenen Grafiken lassen sich als Bilddatei exportieren. Und können als Grundlage für die weitere Forschung verwendet werden.

6 Fazit

In dieser Arbeit wurde der Frage nachgegangen, wie die vorangestellte These über die Anlockwirkung von klassischen Leuchtmitteln gegenüber modernen LED-Leuchten auf herkömmliche Stechmücken, bewiesen oder widerlegt werden kann.

Hierzu entstand die Zielsetzung, einen Prototypen zu konzipieren und zu realisieren, mit dem man eine automatische Zählung von Insekten, innerhalb eines gewissen Bereiches, mittels einer Bilderkennungssoftware durchführen und auswerten kann.

Dazu musste zunächst ein Prototyp gebaut und ein System für die Bildaufnahme und Bildauswertung ermittelt werden. Hierzu wurde sich für die Umsetzung mit dem Raspberry Pi entschieden. Dies ergab eine komfortable Lösung, die sowohl die Auswertung, wie auch die Aufzeichnung der Insektenaufnahmen in einem Gerät ermöglicht. Durch die Entwicklung der graphischen Benutzeroberfläche lässt sich das System leicht bedienen.

Für Vorbereitung der Bilderkennung wurde sich für eine Kombination aus dem Bilateralen Filter und den Merkmaldetektionen mittels des Canny Algorithmus und der in OpenCV implementierten Funktion der Konturenfindung „cv2.findCountours“ entschieden. Der Bilaterale Filter übernahm hierbei die Funktion, das aufgenommene Bild zu glätten, ohne die vorhandenen Objektekanten zu verwischen. Durch die Anwendung des Canny Algorithmus, wurden die vorhanden Objektkanten hervorgehoben und ein binäres Bild erzeugt welches lediglich aus die hervorgehobenen Kanten enthielt. Abschließend konnten die Objektkonturen durch die Konturenfindung detektiert und gezählt werden.

Mit Hilfe des realisierten Prototyps besteht damit die Möglichkeit, das Verhalten von ausgewählten Insekten gegenüber verschiedener Lockstoffe zu testen. Der Prototyp eignet sich vor allem für kleine Kriech- und Fluginsekten mit einem Durchmesser von 4-7mm. Andere Größen konnten nicht getestet werden. Bei abweichenden Größen besteht jedoch die Möglichkeit, die minimale und maximale Konturgröße anzupassen.

Vor allem die implementierte Zwei-Zonen-Variante ermöglicht es, zwei verschiedene Lockstoffe gegeneinander zu vergleichen. Diese können u.a. Licht-, Duft-, Farb- und Wärmequellen sein. Durch die Hervorhebung der Insekten, mittels der farbigen Rechtecke, kann bereits beim Betrachten der Aufzeichnung eine Tendenz auf das Verhalten der Insekten geschlossen werden. Die graphische Auswertung im Anschluss ermöglicht eine sekundengenaue Zuordnung der Insektenposition.

Der aktuelle Aufbau des Prototyps ist, eine sehr kostengünstige Variante, aus Pappkarton, welche sich nicht optimal für den Einsatz zur Erkennung der Insekten eignet. Die Insekten haben die Möglichkeit sich außerhalb des Erkennungsbereichs zu bewegen und verfälschen somit das Endergebnis. Wie bereits im Kapitel der Insektenerkennung erwähnt, ist der Kameraabstand nicht optimal zur Größe der Testfläche.

Um zu einem marktfähigen Produkt zu kommen empfiehlt sich eine stabilere Lösung, mit der u.a. der Kameraabstand besser eingestellt werden kann. Hierzu können z.B. leichte Holzplatten und verstellbare Metallgerüste verwendet werden.

In dieser Arbeit konnte nicht endgültig das Verhalten von Mücken im Bezug zu unterschiedlichen Lichtquellen nachgewiesen werden, da es nicht möglich war während der Bearbeitungszeit, an lebende Mücken zu gelangen. Zur Entwicklung und dem Testen der Software, sowie des Prototypens, wurden ersatzweise Stubenfliegen herangezogen.

Wünschenswert wäre eine Langzeitstudie mit verschiedenen Insekten durchführen zu können, um die Funktion der Software in größerem Umfang zu testen. Auch die Verwendung von verschiedenen Lockstoffen wäre interessant. Dabei könnte der Nutzen auch auf andere Bereiche ausgeweitet werden. Bspw. könnte der Prototyp auch im Bereich der Pflege- und Parfumindustrie angewendet werden.

7 Zusammenfassung & Ausblick

In der dargelegten Arbeit wurde die Konzipierung und Realisierung eines Prototyps zur automatisierten Zählung von Insekten dargestellt.

Hierzu wurde eine Bilderkennungssoftware auf einen Raspberry Pi implementiert, welcher zur Erkennung von Kriech- und Fluginsekten, mit einem Durchmesser von 4-7mm dient. Der entwickelte Aufbau des Prototyps entspricht hierbei einer kostengünstigen Variante, welche für weitere Experimente einer Anpassung bedarf. Die Arbeitsergebnisse zeigen, dass der aktuelle Kameraabstand nicht optimal zur Testarealgröße ist. Um eine bessere Version des Prototyps zu erhalten, empfiehlt sich eine stabilere Lösung mit einer verstellbaren Kamerahalterung.

Die automatische Zählung und die Möglichkeit der Anpassung des Algorithmus für unterschiedliche Testobjektgrößen, ermöglicht eine variable Anwendbarkeit des entwickelten Programms.

Die Ergebnisse der Arbeit können als Grundlage für die weitere Forschung von Insektenverhalten verwendet werden. Da das Verhalten von Mücken im Bezug zu unterschiedlichen Lichtquellen, aufgrund der nicht Verfügbarkeit von Stechmücken nicht nachgewiesen werden konnte, sind weitere Studien hierzu wünschenswert. Der entwickelte Prototyp kommt auch für weitere Branchen in Betracht u.a. die Pflege- und Parfumindustrie, könnte hiermit die Lockwirkung von Duftstoffe auf verschiedene Insekten testen.

Zusammenfassend zeigt diese Arbeit eine erfolgreiche Implementierung einer Bilderkennungssoftware im Zusammenhang mit einer graphischen Auswertung, welche für die zukünftige Forschung der ökologischen Verträglichkeit von unterschiedlichen Lockstoffen eingesetzt werden kann.

Literaturverzeichnis

[1]: Bradski, Gary; Kaehler, Adiran (2008): Learning OpenCV. Computer Vision with the OpenCV Library, USA 2008, 1.Auflage

[2]: Braun, Georg (2011): Effiziente kantenerhaltende Glättung und ihre Anwendung in der Praxis (2011)

[3]: Doxygen(2018): OpenCV. Open Source Computer Vision, https://docs.opencv.org/4.0.0/index.html (Aufgerufen am 20.02.2019)

[4]: Erhard, Angelika (2008): Einführung in die Digitale Bildverarbeitung. Grundlagen, Systeme und Anwendungen, Deutschland 2008, 1.Auflage

[5]: Mag. Dr Huemer, Peter; Mag. Kühtreiber, Hannes; Mag. Tarmann, Gerhard (2010) : Anlockwirkung moderner Leuchtmittel auf nachtaktive Insekten. Ergebnisse einer Feldstudie, http://www.hellenot.org/fileadmin/user_upload/PDF/WeiterInfos/10_AnlockwirkungInsektenFeldstudie_TLMFundLUA.pdf (Aufgerufen am 20.02.2019)

[6]: Klein, Bernd (2013): Einführung in Python 3. In einer Woche programmieren lernen, Deutschland 2013

[7]: Kofler, Michael; Kühnast, Charly; Scherbeck, Christoph (2014): Raspberry Pi. Das umfassende Handbuch, Bonn 2014, 1.Auflage

[8]: Rodner, Erik; Süße, Herbert (2014): Bildverarbeitung und Objekterkennung. Computer Vision in Industrie und Medizin, Wiesbaden 2014

[9]: Stein, Annett(2015), Straßenlaternen mit LED haben Schattenseiten, https://www.welt.de/wissenschaft/umwelt/article145194509/Strassenlaternen-mit-LED-haben-Schattenseiten.html (Aufgerufen am 18.02.2019)

[10]: Stern, Felix: Raspberry Pi Tutorials, https://tutorials-raspberrypi.de/ (Aufgerufen am 20.02.2019)

[11]: Tiroler Umweltanwaltschaft (2015), Helle Not, http://www.hellenot.org (Aufgerufen am 18.02.2019)

Anhänge

Anhang 1: Bilder vom Prototyp

Anhang 2: Datenblätter

Raspberry Pi 3 Model B

Specifications

Processor	Broadcom BCM2837 chipset. 1.2GHz Quad-Core ARM Cortex-A53 802.11 b/g/n Wireless LAN and Bluetooth 4.1 (Bluetooth Classic and LE)
GPU	Dual Core VideoCore IV® Multimedia Co-Processor. Provides Open GL ES 2.0, hardware-accelerated OpenVG, and 1080p30 H.264 high-profile decode. Capable of 1Gpixel/s, 1.5Gtexel/s or 24GFLOPs with texture filtering and DMA infrastructure
Memory	1GB LPDDR2
Operating System	Boots from Micro SD card, running a version of the Linux operating system or Windows 10 IoT
Dimensions	85 x 56 x 17mm
Power	Micro USB socket 5V1, 2.5A

Connectors:

Ethernet	10/100 BaseT Ethernet socket
Video Output	HDMI (rev 1.3 & 1.4 Composite RCA (PAL and NTSC)
Audio Output	Audio Output 3.5mm jack, HDMI USB 4 x USB 2.0 Connector
GPIO Connector	40-pin 2.54 mm (100 mil) expansion header: 2x20 strip Providing 27 GPIO pins as well as +3.3 V, +5 V and GND supply lines
Camera Connector	15-pin MIPI Camera Serial Interface (CSI-2)
Display Connector	Display Serial Interface (DSI) 15 way flat flex cable connector with two data lanes and a clock lane
Memory Card Slot	Push/pull Micro SDIO

Key Benefits

- Low cost
- 10x faster processing
- Consistent board format
- Added connectivity

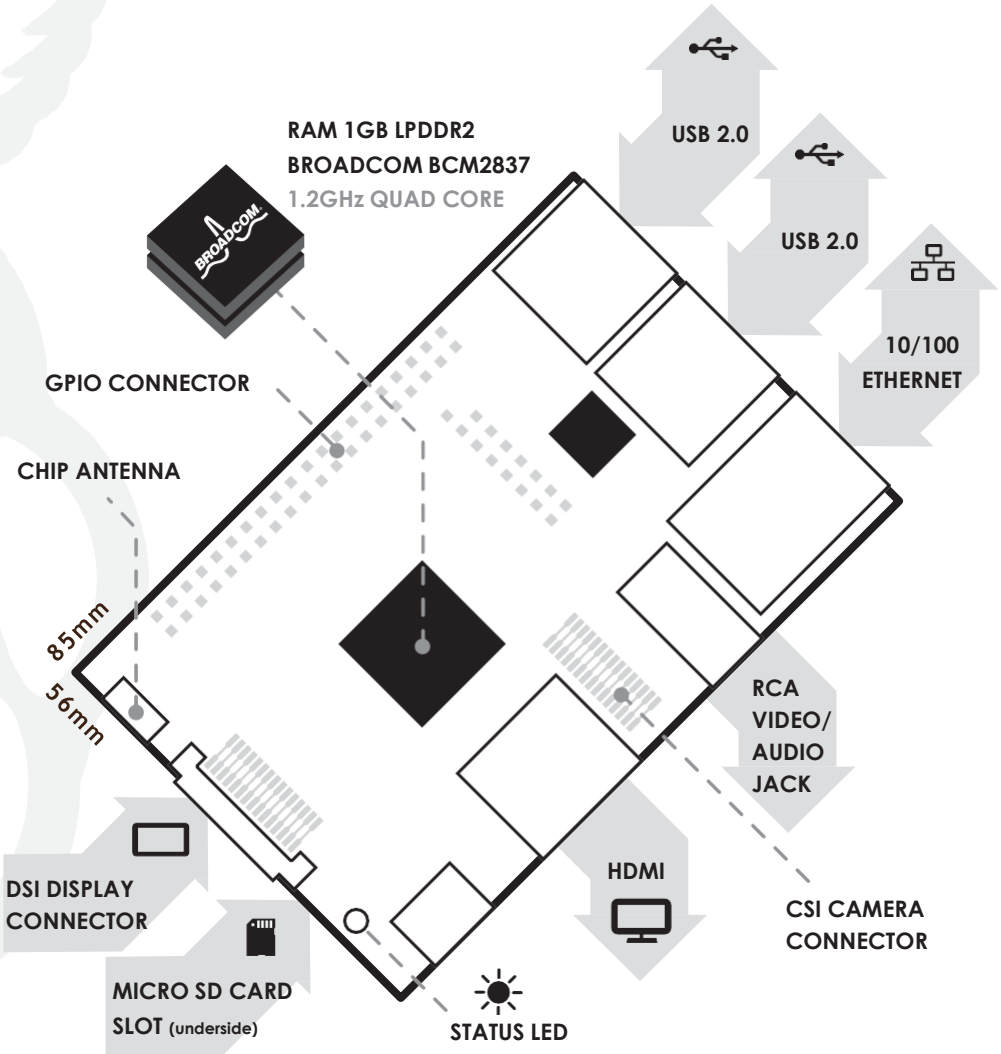
Key Applications

- Low cost PC/tablet/laptop
- Media centre
- Industrial/Home automation
- Print server
- Web camera
- Wireless access point
- IoT applications
- Robotics
- Server/cloud server
- Security monitoring
- Gaming
- Environmental sensing/monitoring (e.g. weather station)

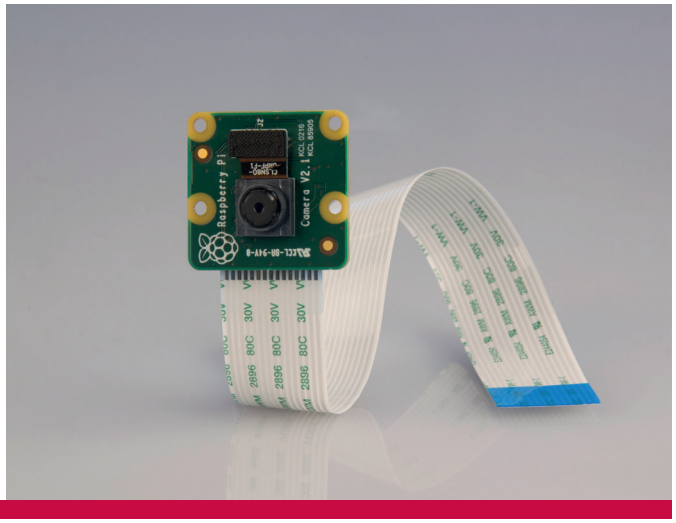
Raspberry Pi

Raspberry Pi 3 Model B

Product Name	**Raspberry Pi 3**
Product Description	The Raspberry Pi 3 Model B is the third generation Raspberry Pi. This powerful credit-card sized single board computer can be used for many applications and supersedes the original Raspberry Pi Model B+ and Raspberry Pi 2 Model B. Whilst maintaining the popular board format the Raspberry Pi 3 Model B brings you a more powerful processer, 10x faster than the first generation Raspberry Pi. Additionally it adds wireless LAN & Bluetooth connectivity making it the ideal solution for powerful connected designs.
RS Part Number	**896-8660**



Raspberry Pi



Camera Module

Product Name	**Raspberry Pi Camera Module**
Product Description	High Definition camera module compatible with all Raspberry Pi models. Provides high sensitivity, low crosstalk and low noise image capture in an ultra small and lightweight design. The camera module connects to the Raspberry Pi board via the CSI connector designed specifically for interfacing to cameras. The CSI bus is capable of extremely high data rates, and it exclusively carries pixel data to the processor.
RS Part Numer	**913-2664**

Specifications

Image Sensor	Sony IMX 219 PQ CMOS image sensor in a fixed-focus module.
Resolution	8-megapixel
Still picture resolution	3280 x 2464
Max image transfer rate	1080p: 30fps (encode and decode) 720p: 60fps
Connection to Raspberry Pi	15-pin ribbon cable, to the dedicated 15-pin MIPI Camera Serial Interface (CSI-2).
Image control functions	Automatic exposure control Automatic white balance Automatic band filter Automatic 50/60 Hz luminance detection Automatic black level calibration
Temp range	Operating: -20° to 60° Stable image: -20° to 60°
Lens size	1/4"
Dimensions	23.86 x 25 x 9mm
Weight	3g

Anhang 3: Quellcode

```python
from picamera.array import PiRGBArray
from picamera import PiCamera
import time


# initialize camera ; set resolution ; get the camera array
camera = PiCamera()
camera.resolution = (640, 480)
cap = PiRGBArray(camera)


# camera need to warm up
time.sleep(0.1)

# get image from camera and set to BGR for OpenCV
camera.capture(cap, format="bgr")
image = cap.array

# display the image in window and wait till any key is pressed
cv2.imshow("Image", image)
cv2.waitKey(0)
cv2.imwrite("Einzelbild.png",image)
```

```python
import numpy as np
import cv2
import time

captime = 180 #capture time

cap = cv2.VideoCapture(0)#capture video from camera '0' (PiCamera)
fourcc = cv2.VideoWriter_fourcc('m', 'p', '4', 'v')#set videoformat
out = cv2.VideoWriter('videoaktuell4.avi',fourcc, 20.0, (640,480))#write video in file

starttime = time.time() #set starttime to actual time

while(int(time.time() - starttime) < captime ):
    ret, frame = cap.read()
    if ret==True:
        frame = cv2.flip(frame,0)

        # write the flipped frame
        out.write(frame)
        #show frame by frame
        cv2.imshow('frame',frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    else:
          break
#release all
cap.release()
out.release()
cv2.destroyAllWindows()
```

```
import sys
import time
import cv2
import numpy as np
from operator import add
from matplotlib import pyplot as plt
from Tkinter import *
import tkFileDialog
import tkMessageBox


#-------------- declaration------------------

cv2.namedWindow('image') # create window

modus=0
areal =1
filename=""

#------------function----------------------

def testareal():
    global areal
    if areal ==1:
        ButtonAreal.configure(bg = "#01DF01")
        areal =2
    else:
        ButtonAreal.configure(bg = "#BDBDBD")
        areal =1


#function for changing mode after button clicked
def mode(value):
    global modus
    modus=value
    if value ==1:
        ButtonMode2.configure(bg = "#DF013A")
        ButtonMode3.configure(bg = "#DF013A")
        ButtonMode1.configure(bg = "#01DF01")
        ButtonGetStart.configure(bg = "#DF013A")
        ButtonGetvideofile.configure(bg = "#BDBDBD")
        ButtonGetPicturefile.configure(bg = "#BDBDBD")
    if value ==2:
        ButtonMode1.configure(bg = "#DF013A")
        ButtonMode3.configure(bg = "#DF013A")
        ButtonMode2.configure(bg = "#01DF01")
        ButtonGetStart.configure(bg = "#DF013A")
        ButtonGetvideofile.configure(bg = "#BDBDBD")
        ButtonGetPicturefile.configure(bg = "#BDBDBD")
    if value ==3:
        ButtonMode1.configure(bg = "#DF013A")
        ButtonMode2.configure(bg = "#DF013A")
        ButtonMode3.configure(bg = "#01DF01")
        ButtonGetStart.configure(bg = "#01DF01")
        ButtonGetvideofile.configure(bg = "#DF013A")
        ButtonGetPicturefile.configure(bg = "#DF013A")

#function for getting videofile after button clicked
def getvideofile():
    global modus
    if modus !=2:
        tkMessageBox.showerror("Error", "Zunaechst den Modus Video waehlen!")
    else:
        global filename
        filename = tkFileDialog.askopenfilename(title = "Select file",filetypes =
        (("video files","*.avi"),("all files","*.*")))
        ButtonGetvideofile.configure(bg = "#01DF01")
        ButtonGetPicturefile.configure(bg = "#DF013A")
        ButtonGetStart.configure(bg = "#01DF01")

#function for getting picture file after button clicked
def getpicturefile():
    global modus
    if modus !=1:
```

```
        tkMessageBox.showerror("Error", "Zunaechst den Modus Einzelbild waehlen!")
    else:
        global filename
        filename = tkFileDialog.askopenfilename(title = "Select file",filetypes =
        (("png files","*.png"),("all files","*.*")))
        ButtonGetvideofile.configure(bg = "#DF013A")
        ButtonGetPicturefile.configure(bg = "#01DF01")
        ButtonGetStart.configure(bg = "#01DF01")

#function for start the algorithm and close gui after button clicked
def start():
    global modus
    global filename

    if ((modus ==1 or modus ==2) and filename == "") or modus == 0:
        tkMessageBox.showerror("Error", "Zunaechst den Modus und Datei waehlen!")
    else:
        gui.destroy()
#--------------------GUI--------------------
#initialize gui and framecontour
gui = Tk()
gui.title("Automatische Zaehlung von Insekten")
gui.minsize (width = 600, height =200)
gui.config(background = "#737373")#backgroundcolour
leftFrame = Frame(gui, width=200, height = 300)
leftFrame.grid(row=0, column=0, padx=15, pady=3)
leftFrame.pack()

#simple textlabel
TextMode = Label(leftFrame, text="Optional: 2 Zonen Modus")
TextMode.grid(row=0, column=0, padx=10, pady=3)

#Button for set 2 seperate Areal
ButtonAreal=Button(leftFrame, text="2 Zonen Testareal", bg="#BDBDBD", width=15,
command= testareal)
ButtonAreal.grid(row=1, column=0, padx=10, pady=3)

Space = Label(leftFrame, text="") #just a empty textfield for a space
Space.grid(row=2, column=0, padx=10, pady=3)

TextMode = Label(leftFrame, text="1. Modus waehlen")
TextMode.grid(row=4, column=0, padx=10, pady=3)

#Buttons for choosing Mode
ButtonMode1=Button(leftFrame, text="Einzelbild", bg="#BDBDBD", width=15,
command= lambda *args:mode(1))
ButtonMode1.grid(row=5, column=0, padx=10, pady=3)
ButtonMode2=Button(leftFrame, text="Video", bg="#BDBDBD", width=15,
command=lambda *args:mode(2))
ButtonMode2.grid(row=6, column=0, padx=10, pady=3)
ButtonMode3=Button(leftFrame, text="Live-Video", bg="#BDBDBD", width=15,
command=lambda *args:mode(3))
ButtonMode3.grid(row=7, column=0, padx=10, pady=3)

Space = Label(leftFrame, text="")
Space.grid(row=8, column=0, padx=10, pady=3)

TextMode1 = Label(leftFrame, text="2. Datei waehlen")
TextMode1.grid(row=9, column=0, padx=10, pady=3)

# Buttons for getting video or picture file


ButtonGetPicturefile=Button(leftFrame, text="Einzelbild auswaehlen", bg="#BDBDBD",
width=15, command=getpicturefile)
ButtonGetPicturefile.grid(row=10, column=0, padx=10, pady=3)

ButtonGetvideofile=Button(leftFrame, text="Video auswaehlen", bg="#BDBDBD", width=15,
command=getvideofile)
ButtonGetvideofile.grid(row=11, column=0, padx=10, pady=3)

Space = Label(leftFrame, text="")
Space.grid(row=12, column=0, padx=10, pady=3)

```

```
TextMode2 = Label(leftFrame, text="3. Auswertung Starten")
TextMode2.grid(row=13, column=0, padx=10, pady=3)

# Button for start algorithm and close gui
ButtonGetStart=Button(leftFrame, text="Start", bg="#DF013A", width=15, command=start)
ButtonGetStart.grid(row=14, column=0, padx=10, pady=3)

Space = Label(leftFrame, text="")
Space.grid(row=15, column=0, padx=10, pady=3)


Hinweis = Label(leftFrame, text="Beende Algortihmus/Video mit der Taste'Q' ")
Hinweis.grid(row=15, column=0, padx=10, pady=3)


gui.mainloop()



#--------------------Graph paramters--------------------

graphcountleft=[0]
graphcountright=[0]
graphcountareal=[0]
graphxaxis=[0]

#----------------single picture--------------
if modus == 1:

    framergb = cv2.imread(filename,1) # read picture from file in colour
    framegray = cv2.imread(filename,0)# read picture from file in gray

#--------------------video-------------------
if modus == 2:

    cap = cv2.VideoCapture(filename) # read video from file

#--------------------LIVE--------------------
if modus == 3:

    cap = cv2.VideoCapture(0) # read video live from camera

        fourcc = cv2.VideoWriter_fourcc('m', 'p', '4', 'v')
        out = cv2.VideoWriter('liveVideoLANG.avi',fourcc, 20.0, (640,480))


#-----------algorithm for single picture-----



if modus == 1:
    cmax=60 # maximum contour size
    cmin=5 # minimum contour size
    #filtering by bilateral filter
    blur=cv2.bilateralFilter(framegray,3,75,75)

    #edges by canny algorithm
    framefiltered = cv2.Canny(blur,40,120)

    #contour detection by findContours
    framecont, contours, hierarchy = cv2.findContours(framefiltered, cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)


    if areal ==2 :#if the areal is seperated
        height, width = framegray.shape
        countleft = 0
        countright =0
        for c in contours:

            if cv2.contourArea(c) < cmax and cv2.contourArea(c) > cmin:
            #Only if Area of Object is between cmin and cmax
                (x, y, w, h) = cv2.boundingRect(c)
                if x <= width/2:#if object is on the left side of the border
```

```
                    countleft = countleft +1
                    cv2.rectangle(framergb, (x-7, y-7), (x + w+7, y + h+7), (255, 0, 0), 1)

                if x >  width/2: #if object is on the right side of the border
                    countright = countright+1
                    cv2.rectangle(framergb, (x-7, y-7), (x + w+7, y + h+7), (0, 0, 255), 1)


        font = cv2.FONT_HERSHEY_SIMPLEX
        cv2.putText(framergb,str(countleft),(10,height-30), font, 4,
        (255,255,255),2,cv2.LINE_AA)
        cv2.putText(framergb,str(countright),(width-150,height-30), font, 4,
        (255,255,255),2,cv2.LINE_AA)
        cv2.line(framergb, (width/2, 0), (width/2, height), (0,255,0), 2)
    else: # if the areal is not seperated
        countglobal=0
        for c in contours:

            if cv2.contourArea(c) < cmax and cv2.contourArea(c) > cmin:
            #Only if Area of Object is between cmin and cmax
                (x, y, w, h) = cv2.boundingRect(c)
                countglobal = countglobal +1
                cv2.rectangle(framergb, (x-7, y-7), (x + w+7, y + h+7), (255, 0, 0), 1)


        font = cv2.FONT_HERSHEY_SIMPLEX
        cv2.putText(framergb,str(countglobal),(10,460), font, 4,
        (255,255,255),2,cv2.LINE_AA)

    cv2.imshow('image',framergb) #Show Video in Window
    cv2.waitKey(0)



#----------algorithm for video or live------


if modus == 2 or modus == 3:

    def nothing(x):
        pass
    # create trackbars for size change
    cv2.createTrackbar('ContourSizeMin','image',0,100,nothing)
    cv2.createTrackbar('ContourSizeMax','image',0,200,nothing)

    cmin=0
    cmax=0


    framecount=0# counter for grafik evaluation

    while(True):

        # capture frame
        ret,frame = cap.read()

        if frame == None: # if the captured frame is empty the algorithm stopped
            break
        frame = cv2.flip(frame,0)
        time.sleep(0.1)
        framegray=cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) #put frame from BGR to gray
        framergb = cv2.cvtColor(frame,cv2.COLOR_BGR2RGB) #put frame from BGR to RGB

        framecount=framecount+1 #framecounter for evaluation graph

        #filtering by bilateral filter
        blur=cv2.bilateralFilter(framegray,3,75,75)

        #edges by canny algorithm

        #-------------Contours-------------------------
        def canny(image, var):
        # automatic canny algorithm with including
        #the median from the grayscale of the video
```

```
                    median = np.median(image)
                    lower = int(max(0, (1.0 - var) * median))
                    upper = int(min(255, (1.0 + var) * median))
                    filtered = cv2.Canny(image, lower, upper)
                    return filtered


                canny_var=0.22
                # variable for canny algorithm, should be adapt to the captured video
                framefiltered =canny(blur,canny_var)

        #contour detection by findContours
        framecont, contours, hierarchy = cv2.findContours(framefiltered,
        cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        #Count and create bounding boxes
        countleft = 0
        countright =0

        if areal ==2 :
            height, width, channel = framergb.shape
            # frame shape for putting area in two zones
            countleft = 0
            countright =0
            for c in contours:

                if cv2.contourArea(c) < cmax and cv2.contourArea(c) > cmin:
                #Only if Area of Object is between cmin and cmax
                    (x, y, w, h) = cv2.boundingRect(c) # get the positon from contour
                    if x <= width/2:#if object is on the left side of the border
                        countleft = countleft +1
                        cv2.rectangle(framergb, (x-7, y-7), (x + w+7, y + h+7),
                         (255, 0, 0), 1)

                    if x >  width/2: #if object is on the right side of the border
                        countright = countright+1
                        cv2.rectangle(framergb, (x-7, y-7), (x + w+7, y + h+7)
                        , (0, 0, 255), 1)


            font = cv2.FONT_HERSHEY_SIMPLEX
            cv2.putText(framergb,str(countleft),(10,height-30), font, 4,
            (255,255,255),2,cv2.LINE_AA)
            cv2.putText(framergb,str(countright),(width-150,height-30), font, 4,
            (255,255,255),2,cv2.LINE_AA)
            cv2.line(framergb, (width/2, 0), (width/2, height), (0,255,0), 2)

            # put actual count value in graphcounter lists
            graphcountleft.append(countleft)
            graphcountright.append(countright)

        else:
            countglobal=0
            for c in contours:

                if cv2.contourArea(c) < cmax and cv2.contourArea(c) > cmin:
                #Only if Area of Object is between cmin and cmax
                    (x, y, w, h) = cv2.boundingRect(c)
                    countglobal = countglobal +1
                    cv2.rectangle(framergb, (x-7, y-7), (x + w+7, y + h+7),
                    (255, 0, 0), 1)


            font = cv2.FONT_HERSHEY_SIMPLEX
            cv2.putText(framergb,str(countglobal),(10,460), font, 4,(255,255,255),2,
            cv2.LINE_AA)
            # put actual count value in graphcounter lists
            graphcountareal.append(countglobal)


        cv2.imshow('image',framergb) #Show Video in Window

        # Get Trackbar Values
        cmin=cv2.getTrackbarPos('ContourSizeMin','image')
```

```
        cmax=cv2.getTrackbarPos('ContourSizeMax','image')

        # put actual count of frames in x axis lists
        graphxaxis.append(framecount)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break


    cap.release()
    cv2.destroyWindow('image')

#----------------- Evaluation ---------------

    n=20 # Frames per sec
    # build the average of all counter values over 20 frames per secs
    if areal == 2:
        averagesleft = [sum(graphcountleft[i:i+n])//n for i in range
        (0,len(graphcountleft),n)]
        averagesright = [sum(graphcountright[i:i+n])//n for i in range
        (0,len(graphcountright),n)]
        sumobject=list( map(add, averagesleft, averagesright) )

        #plots

        plt.figure(1)
        plt.subplot(311)
        plt.title("Left Area")
        plt.plot(averagesleft,drawstyle='steps-pre')
        plt.xlabel("time in s")
        plt.ylabel("Number of Insects")
        plt.xticks(np.arange(0,len(sumobject)+1,10))
        plt.yticks(range(min(averagesleft),max(averagesright)+1))

        plt.subplot(312)
        plt.title("Right Area")
        plt.plot(averagesright,drawstyle='steps-pre')
        plt.xlabel("time in s")
        plt.ylabel("Number of Insects")
        plt.xticks(np.arange(0,len(sumobject)+1,10))
        plt.yticks(range(min(averagesright),max(averagesright)+1))

        plt.subplot(313)
        plt.title("Complete Area")
        plt.plot(sumobject,drawstyle='steps-pre')
        plt.xlabel("time in s")
        plt.ylabel("Number of Insects")
        plt.xticks(np.arange(0,len(sumobject)+1,10))
        plt.yticks(range(min(sumobject),max(sumobject)+1))

        plt.figure(2)
        plt.plot(sumobject,drawstyle='steps-pre',label='sum')
        plt.plot(averagesright,drawstyle='steps-pre',label='right')
        plt.plot(averagesleft,drawstyle='steps-pre',label='left')
        plt.xticks(np.arange(0,len(sumobject)+1,10))
        plt.yticks(range(min(sumobject),max(sumobject)+1))
        plt.xlabel("time in s")
        plt.ylabel("Number of Insects")
    else:
        sumobject = [sum(graphcountareal[i:i+n])//n for i in range
        (0,len(graphcountareal),n)]

        plt.figure(1)
        plt.plot(sumobject,drawstyle='steps-pre',label='sum')
        plt.xticks(np.arange(0,len(sumobject)+1,10))
        plt.yticks(range(min(sumobject),max(sumobject)+1))
        plt.xlabel("time in s")
        plt.ylabel("Number of Insects")

    plt.legend()
    plt.show()
    cv2.destroyAllWindows()#close all windows
```

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Dieses Blatt, mit der folgenden Erklärung, ist nach Fertigstellung der Abschlussarbeit durch den Studierenden auszufüllen und jeweils mit Originalunterschrift als letztes Blatt in das Prüfungsexemplar der Abschlussarbeit einzubinden.
Eine unrichtig abgegebene Erklärung kann -auch nachträglich- zur Ungültigkeit des Studienabschlusses führen.

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: Mac Allister

Vorname: Danny

dass ich die vorliegende Bachelorarbeit bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Konzeption und Realisierung eines Prototyps zur automatisierten Zählung von Insekten mittels Bilderkennungssoftware

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

- die folgende Aussage ist bei Gruppenarbeiten auszufüllen und entfällt bei Einzelarbeiten -

Die Kennzeichnung der von mir erstellten und verantworteten Teile der -bitte auswählen- ist erfolgt durch:

Hamburg	08.03.19	
Ort	Datum	Unterschrift im Original