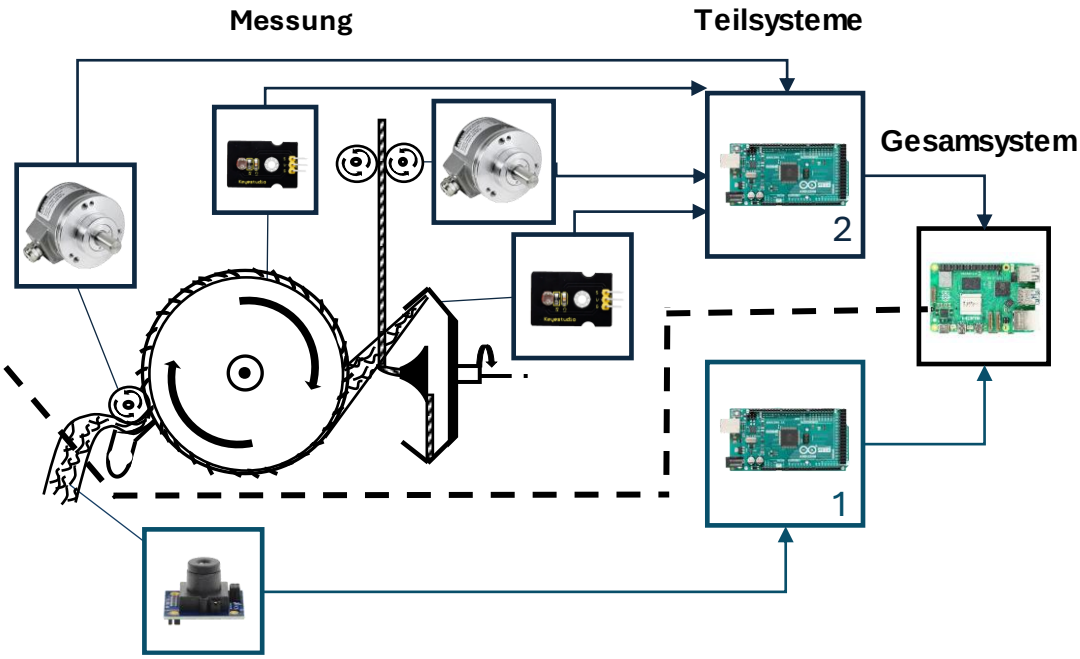


2024
Projektarbeit

Entwurf eines Sensorsystems
ANLEITUNG UND CODESTRUKTUR

Raphael Karmalker
Can-Louis Yilmaz

Inhaltsverzeichnis

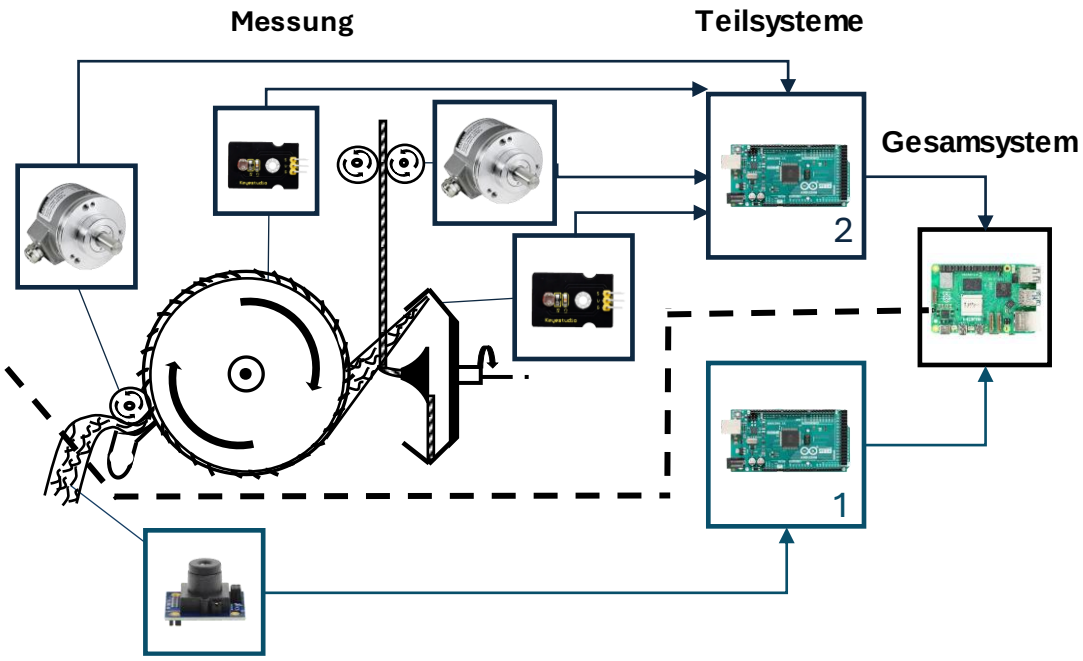
1 Einleitung & Systemstruktur

1.1 Vorwort & Download

Bei konkreten Fragen zu dem Code gerne an raphaelkarmalker@gmail.com wenden.

Der gesamte Code des Sensorsystems befindet sich unter https://github.com/Raphael-Karmalker/SensorSystem.git. Er wurde für den Arduino Microcontroller verfasst und in C++ geschrieben. Für den Raspberry Pi 5 entsprechend in Python.

1.2 Systemstruktur



Der Code besteht aus dem Arduino C++ Code für Teilsystem 1 (2) und Teilsystem 2 (3) sowie Python Code für den Raspberry Pi 5 (4). Im Folgenden wird die grobe Codestruktur der einzelnen Systeme erklärt und wichtige Algorithmen werden erläutert.

2 Teilsystem 1 – Liniensensor

Auf Teilsystem 1 läuft ausschließlich der Liniensensor. In diesem Abschnitt werden die Funktionsmöglichkeiten Continious() und OneShot() erläutert:

2.1 Dateistruktur

LiniensensorSystem.ino -> Ausführbare Datei auf dem Arduino

LiniensensorSystem.h -> Header-Datei, Definition von Hilfsvariablen und Hilfsfunktionen

LineScan.cpp -> Alle Funktionen und Klasse des Liniensensors

LineScan.h -> Header-Datei, Definition von Hilfsvariablen und Hilfsfunktionen

2.2 Continious() und OneShot() - Betriebsmöglichkeit

Tab. 2.1: Definition und Beschreibung relevanter Signale des Liniensensors [DS1]

Signal	Definition	Beschreibung
A0	Analog0	analoger Pixeleingang von Sensor: 0 – Vdd
Si	Start Integration	digitaler Ausgang zu Sensor: startet Scan/Belichtung
Clk	Clock/Takt	digitaler Ausgang zum Sensor: übernimmt das SI-Signal und taktet die Pixel aus

Für den Betrieb des Liniensensors sind folgende Signale erforderlich: A0, Si und Clk (Tab. 2.1).

Der Beginn einer Funktionsperiode wird durch das Senden eines Si-Impulses von dem Microcontroller an den Sensor signalisiert. Zusätzlich wird ein einzelner Clk-Impuls gesendet, um das Si-Signal zu registrieren. Die Zeitspanne zwischen zwei Si-Impulsen bzw. zwei Funktionsperioden wird als Belichtungszeit bezeichnet. Während dieser Zeitspanne wird die eintreffende Lichtintensität integriert, wodurch am Ende der entsprechende Helligkeitswert entsteht. Je länger die Belichtungszeit, desto höher ist der resultierende Helligkeitswert. Die Integration der empfangenen Lichtintensität wird im Folgenden als „Integration Scan" definiert.

Erfasste Werte aus Scan n können erst in der folgenden Funktionsperiode P_{n+1} ausgelesen werden. Um die Werte auszulesen, werden z Taktimpulse (Clk) an den Sensor gesendet, wobei z der Anzahl der Photorezeptoren entspricht. Jeder Taktimpuls verschiebt die Ladung von einem Pixel zum nächsten. Die Spannung des A0-Ausgangs wird hierbei proportional zu der im Pixel gesammelten Lichtmenge angepasst. Zwischen

jedem Impuls misst der Mikrocontroller die Spannungswerte des A0-Ausgangs. Anschließend erfolgt die Umwandlung dieser Werte in entsprechende Helligkeitswerte. Das Auslesen der Messwerte wird im weiteren Verlauf als „Clock out Scan“ bezeichnet.

Tab. 2.2: Beschreibung verschiedener Scanarten des Liniensensors

Scan	Beschreibung
Integration Scan	Integrieren der empfangenen Lichtintensität
Clock out Scan	Auslesen der Messwerte
Garbage Clock out Scan	Leeren der Messwerte für den nächsten Scan

Tab. 2.2 fasst die unterschiedlichen Scanarten zusammen. Im Anwendungsfall des Sensors lässt sich zwischen zwei verschiedenen Betriebsmodi unterscheiden. Es wird zwischen dem kontinuierlichen Modus (2.2.1) und One Shot Modus (2.2.2) differenziert. [DS1]

2.2.1 Kontinuierlicher Modus

In dem kontinuierlichen Modus erfolgen die Si-Impulse und Funktionsperioden in einem regelmäßigen zeitlichen Abstand. Die Zeit zwischen zwei Si-Impulsen entspricht der Zeit, in welcher der Sensor einen Scan absolviert. Außerdem definiert sie die Belichtungszeit. In Funktionsperiode P_n findet der erste Integration Scan statt. In Periode P_{n+1} erfolgt der Clock out Scan. Simultan wird der nächste Integration Scan durchgeführt. Der Integration Scan aus Periode P_{n+1} wird in Periode P_{n+2} ausgelesen (Clock out Scan). Dies ist ein kontinuierlicher Prozess. (Abb. 2.2.1)

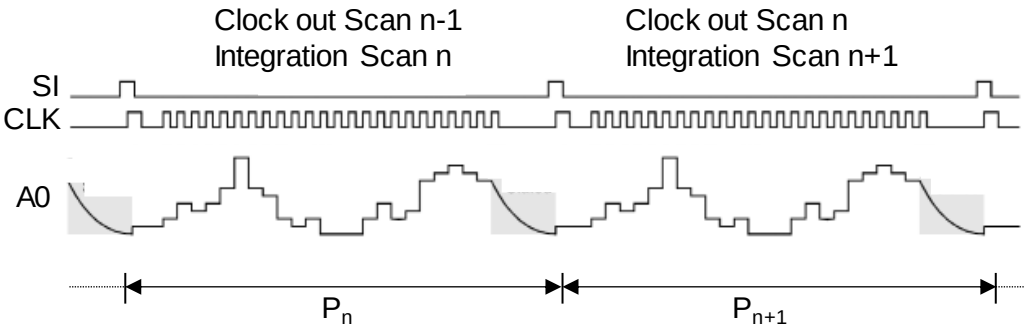


Abb. 2.2.1: Funktionsperioden im kontinuierlichen Modus [DS1]

2.2.2 One Shot Modus

Der OneShot Modus ist kein kontinuierlicher Prozess. In diesem Modus bilden zwei Funktionsperioden eine isolierte Messperiode.

Die erste Funktionsperiode P_n dient als Integration Scan. Zusätzlich wird ein Garbage Clock out Scan durchgeführt, um zuvor irrelevant gesammelte Lichtdaten zu leeren. Die Länge dieser Periode definiert die Belichtungszeit. In der darauffolgenden Periode P_{n+1} erfolgt der Clock out Scan. Bis zur nächsten Funktionsperiode kann eine unbestimmte Zeit vergehen, da die Belichtungszeit von Periode P_{n+1} keinen direkten Einfluss auf den nächsten Clock out Scan hat. (Abb. 2.2.2)

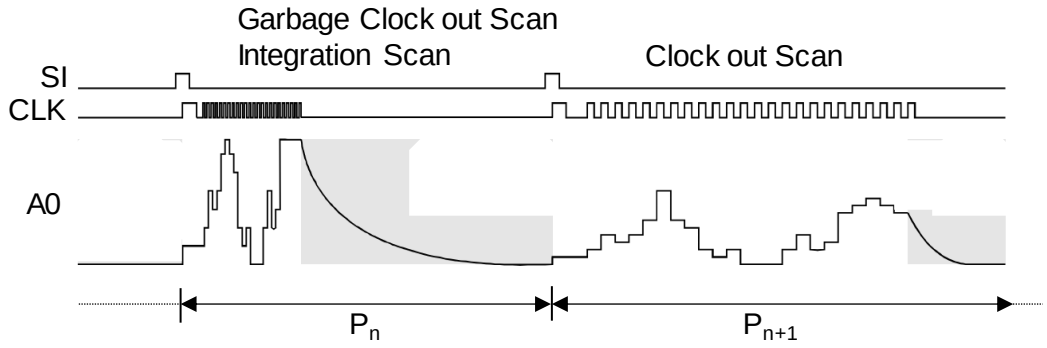


Abb. 2.2.2: Funktionsperioden im One Shot Modus [DS1]

2.3 Automatischer Faserschwellenwert

```
automateTresholdCalculation();
```

$$\gamma_{\text{Hintergrund,ges}} = \frac{\sum_{i=0}^{\kappa} \gamma_{\text{Hintergrund,i}}}{\kappa} \qquad \text{Gl. 2.1}$$

$$\gamma_{\text{Faserband,ges}} = \frac{\sum_{i=0}^{\varrho} \gamma_{\text{Faserband,i}}}{\varrho} \qquad \text{Gl. 2.2}$$

$$\vartheta = \frac{\gamma_{\text{Hintergrund,ges}} + \gamma_{\text{Faserband,ges}}}{2} \qquad \text{Gl. 2.3}$$

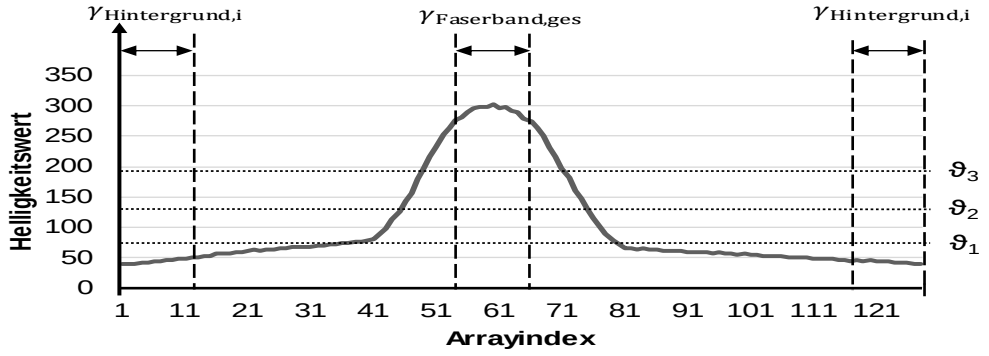


Abb. 2.3: Prinzip der Berechnung eines automatisierten Schwellenwerts ϑ

2.4 Ausreißer einschränken

```
limitExceptions();
```

Es besteht immer die Möglichkeit, dass Lichtreflektionen oder andere störende Einflüsse in der Umgebung auftreten, wodurch ein Pixel außerhalb des eigentlichen Faserbands einen Helligkeitswert über dem Schwellenwert erfasst. Um dies zu verhindern, wird ein weiteres Prüfkriterium eingeführt. Es soll überprüft werden, an welcher Stelle die Dicke des Faserbandes anfängt und wo sie aufhört. Jeder Pixel außerhalb des Faserbandbereichs wird anschließend unabhängig seines Helligkeitswerts als Umgebung klassifiziert. Der Anfang und das Ende des Faserbandes werden durch die ersten oder letzten κ unmittelbar aufeinanderfolgenden Pixel bestimmt, die als Faserband klassifiziert sind. Der erste dieser Pixel markiert den Anfang, der letzte das Ende des Faserbandes.

Die Anzahl der Pixel, welche das Faserband klassifizieren ist eine relative Zielgröße. Eine Umrechnung in die tatsächliche Breite ($width$) des Faserbandes ist über Multiplikation der als Faserband klassifizierten Pixel $n_{\text{Faserband}}$ und der projizierten Länge am Faserband L_{pixel}, die ein Pixel repräsentiert, möglich.

$$width = n_{Faserband} \cdot L_{pixel} \qquad \text{Gl. 2.4}$$

Die Ermittlung der Länge des Faserbandes ist über eine manuelle Kalibrierung möglich. Dazu wird ein Referenzobjekt mit der bekannten Länge L_{ref} verwendet. Es erfolgt eine Messung, wie viele Pixel des Liniensensors der bekannten Länge des Referenzobjekts entsprechen. Diese Größe entspricht $n_{pixelAboveϑ}$. Über folgenden Zusammenhang (Gl. 2.5) lässt sich auf die Länge des Faserbandes schließen, die ein Pixel repräsentiert.

$$L_{pixel} = L_{ref} \cdot n_{pixelAboveϑ} \qquad \text{Gl. 2.5}$$

Für eine präzisere Kalibrierung sollten mehrere Referenzobjekte mit unterschiedlichen Referenzlängen L_{ref} verwendet werden. Daraus wird der Durchschnittswert für die gesuchte Variable L_{pixel} berechnet.

3 Teilsystem 2

3.1 Dateistruktur

SensorSystem.ino-> Ausführbare Datei auf dem Arduino

SensorSystem.h -> Header-Datei, Definition von Hilfsvariablen und Hilfsfunktionen

LineScan.cpp -> Alle Funktionen und Klasse des Liniensensors

LineScan.h -> Header-Datei, Definition von Hilfsvariablen und Hilfsfunktionen

Lightsensor.cpp -> Alle Funktionen und Klasse des Lichtsensors

Lightsensor.h -> Header-Datei, Definition von Hilfsvariablen und Hilfsfunktionen

Encoder.cpp -> Alle Funktionen und Klasse des Drehgebers

Encoder.h -> Header-Datei, Definition von Hilfsvariablen und Hilfsfunktionen

3.2 Zyklischer Prozess

Die grundlegende Idee des Zyklischen Prozesses ist, dass in jeder Messperiode genau ein Messwert gesendet und verarbeitet wird. Eine Messperiode beinhaltet hierbei die Messung aller Sensorwerte, aber die Verarbeitung und Sendung von nur einem. Bei vier Messstellen werden nach vier Messperioden alle Werte gesendet. Bezieht man zusätzlich den Statuscode ein, beträgt ein Zyklus 5 Messperioden. (Abb. 3.1)

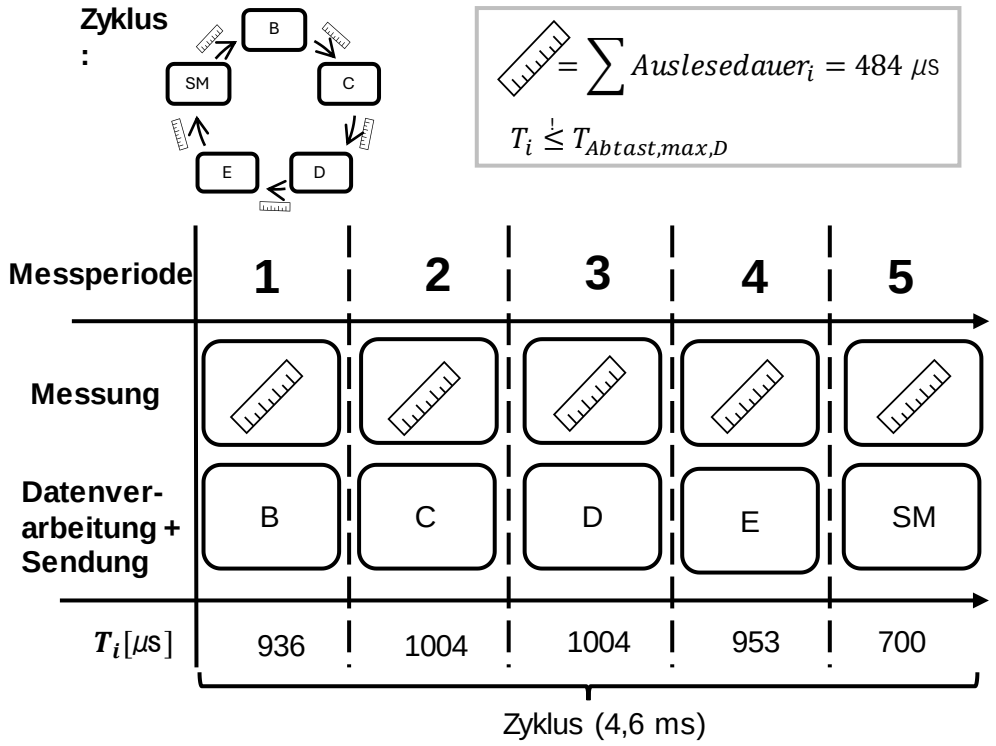


Abb. 3.1: Zusammenfassung des Zyklischen Prozesses

3.3 Elektrisches Funktionsprinzip - Drehgeber

Tab. 3.1: Beschreibung relevanter Signale eines Drehgebers

Signal	Beschreibung
A	Impuls bei Rotation der Welle
B	Impuls bei Rotation der Welle, um 90 Grad phasenverschoben zu A
Z	Impuls bei einer vollständigen Umdrehung der Welle
invA	Invertiertes Signal zu A
invB	Invertiertes Signal zu B
invZ	Invertiertes Signal zu Z

Der Drehgeber verfügt über sechs verschiedene Ausgangssignale: A, B, Z, invA, invB, invZ. Ein gesendetes Signal nimmt entweder den Spannungszustand „High“(5V) oder „Low“(0V) an. Ein Impuls ist eine elektrische Signalform, die durch einen kurzzeitigen

Wechsel des Spannungszustands von einem niedrigen (Low) Pegel zu einem hohen (High) Pegel und wieder zurück zu einem niedrigen Pegel gekennzeichnet ist. In Tab. 3.1 werden alle Ausgangssignale ihrer Funktion zugeordnet. Abb. 3.2 zeigt die Visualisierung der SignaleFür die Auflösung z des Drehgebermodells gilt z = 1024. [DS2]

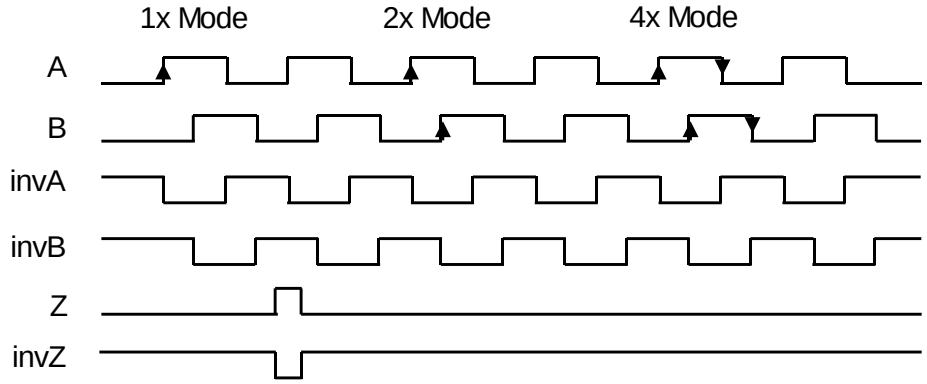


Abb. 3.2: Visualisierung relevanter Signale eines Drehgebers [DS2]

$$rpm = \frac{i \cdot 60}{z \cdot t_{interval}} \qquad \text{Gl. 3.1}$$

Mit i = Anzahl gemessener Impulse in Zeitintervall $t_{interval}$ bei Auflösung z.

3.4 Initiale Kalibrierung Schwellenwert Lichtsensor

```
Calibration1();

Calibration2();
```

Diese Funktionen ermöglichen eine initiale Schwellenwertsanpassung an die Umgebungsverhältnisse.

$$\vartheta = \frac{\gamma_{Schwarz} + \gamma_{\text{Weiß}}}{2} \qquad \text{Gl. 3.2}$$

Min/Max Klassifikation - Calibration1();

Für die Ermittlung von $\gamma_{Schwarz}$ und $\gamma_{\text{Weiß}}$ erfolgt eine Klassifikation des Zustandes ohne konkreten Schwellenwert. Hierfür werden in einem Zeitintervall t_{mess} Helligkeitsmessungen des Zustandes getätigt. Nach Ablauf des Zeitintervalls liegen alle Werte h_{mess} vor. Der niedrigste Messwert h_{lowest} wird als γ_{schwarz} definiert. Analog wird der höchste Messwert $h_{highest}$ als $\gamma_{\text{Weiß}}$ definiert.

Eine solche Ermittlung von γ stellt die Gefahr, dass „Ausreißer" das Ergebnis unerwünscht beeinflussen. Um dies zu vermeiden und eine zuverlässigere Zustandsklassifizierung zu ermöglichen lässt sich der folgende Ansatz umsetzen.

Vereinfachte Signifikanz Klassifikation – Calibration2();

Da die Rechenkapazität des Sensorsystems begrenzt ist wird im Folgenden ein vereinfachter Algorithmus der Signifikanz Klassifikation vorgestellt.

Statt der Suche nach $h_{signifikant}$ mittels einer Iteration durch alle Messwerte, wird $h_{signifikant}$ in der Mitte der sortierten Werte definiert. (Abb. 3.3)

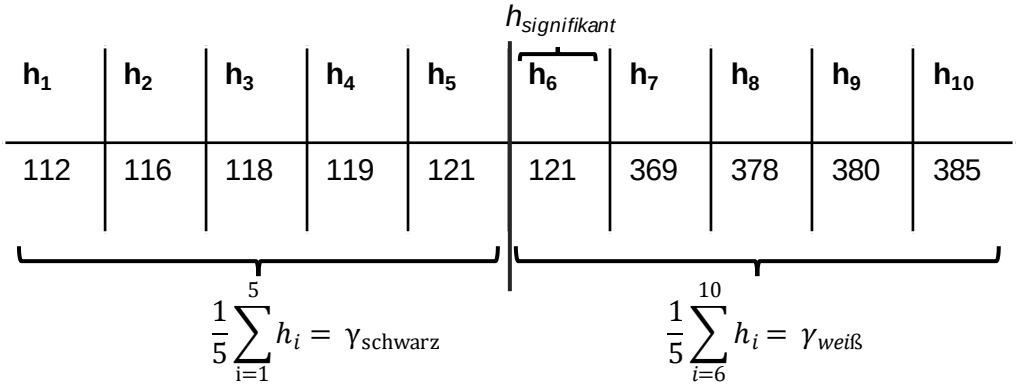


Abb. 3.3: Berechnung von $\gamma_{Schwarz}$ und $\gamma_{\text{Weiß}}$ mittels einer vereinfachten Signifikanz Klassifikation

3.5 Adaptiver Schwellenwert

Das Ziel der adaptiven Schwellenwertberechnung besteht darin, den Schwellenwert in Echtzeit an die gegebenen Lichtverhältnisse anzupassen.

Voraussetzung ist ein erster berechneter Schwellenwert mittels einer der dargestellten Klassifzierungsmethoden. Im Folgenden wird der Algorithmus des adaptiven Schwellenwerts anhand eines Auszugs des Microcontrollercodes (C++) erklärt.

Während des Auslesens der Messwerte h_{mess} zur Erkennung einer Zustandsänderung (1) werden diese zunächst gesammelt (2). Im Falle der Identifizierung einer Zustandsänderung (4) wird der Mittelwert der Messwerte zwischen der aktuellen (Z_i) und der vor-

herigen Zustandsänderung Z_{i-1} berechnet (5). Dieser Vorgang wird für jede Zustandsperiode Z durchgeführt. Zwei aufeinanderfolgende Messungen entsprechen dabei zwei unterschiedlichen Zuständen. Aus den Durchschnittswerten der Messungen Z_i und Z_{i-1} resultieren die Werte $\gamma_{Schwarz}$ und $\gamma_{\text{Weiß}}$.

Gemäß Gleichung 3.2 lässt sich der aktuelle Schwellenwert ϑ_{actual} berechnen (6). Um die Berechnungen vor Ausreißern zu schützen, wird für den neuen Schwellenwert ϑ_i der Durchschnitt des aktuellen Schwellenwerte ϑ_{actual} und des vorherigen Schwellenwertes ϑ_{i-1} gebildet (7). Dieser iterative Prozess erfolgt nach jeder Zustandsänderung und stellt sicher, dass der Schwellenwert kontinuierlich an die aktuellen Lichtverhältnisse angepasst wird.

Alogrithmus zur Berechnung des adaptiven Schwellenwerts:

```
(1)  currentStateWhite = (lichtwert > threshold);
(2)   sum += lichtwert;
(3)   anzahlMesswerte +=1;
(4)   if (currentStateWhite != lastStateWhite){
(5)     Avg = sum/ anzahlMesswerte;
(6)     actualThreshold = (Avg+LastAvg)/2;
(7)     threshold = (threshold + actualThreshold)/2;
(8)     anzahlMesswerte = 0;
(9)     Sum = 0;
(10)    LastAvg = Avg;
(11)    lastStateWhite = currentStateWhite;}
```

4 Gesamtsystem & Tools

In diesem System werden die beiden Teilsysteme zusammengeführt.

4.1 Dateistruktur

Gesamtsystem.py -> Funktionsdatei die Teilsysteme zu Gesamtsystem verbindet und eine csv-Datei mit Messwerten erstellt

4.2 Liniensensor Evaluationstool

Das Tool kann verwendet werden, um Livedaten des Sensors zu visualisieren. (Abb.4.1)

Hierfür muss in Teilsystem 1 auf dem Arduino die Funktion `UseSim();` verwendet werden. Das PyTool muss nur ausgeführt werden.

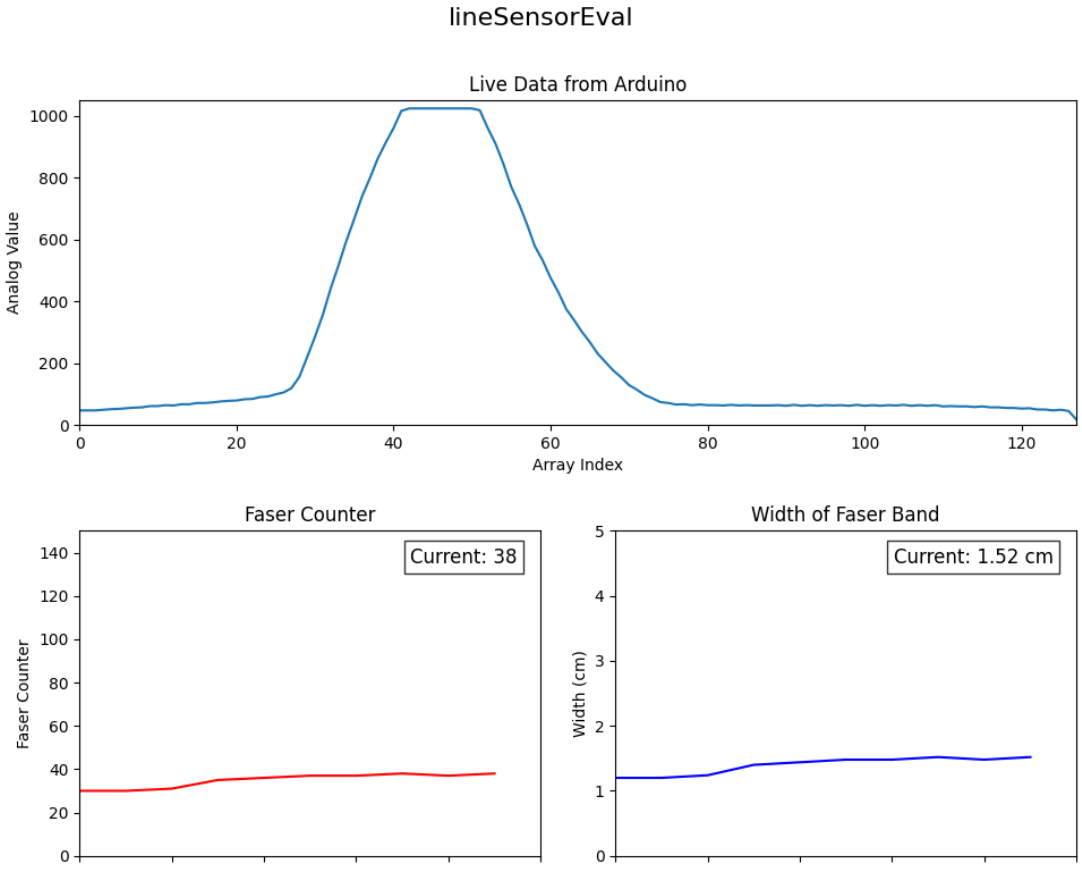


Abb. 4.1: Live Evaluation Tool des Liniensensors