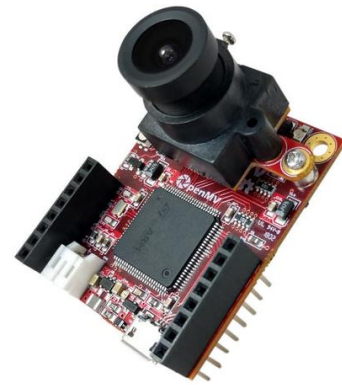


## Features

- Zur Implementation im HTLRIOT System entwickelt
- Positionserfassung von bis zu 2 Zügen in einem Bereich von 5.035m x 0.6m
- Verwendung des OpenMV-Systems
  - ARM Cortex M7 STM32H743VI - Prozessor
    - High-Level Programmierung mit MicroPython
    - Echtzeitbildverarbeitung direkt auf dem Kamerasystem mit 480MHz Systemtakt
- Integrierung der Hardware ins Gesamtsystem HTLRIOT
- Kommunikation über virtuelle serielle Schnittstelle
- Plug&Play nach erfolgreicher Einrichtung
- Einfache Erweiterung von zu detektierenden Elementen durch CIELAB-Farberkennung



**optische  
Positionserfassung  
mittels**



Leitfaden zur Hard- & Softwaremäßigen Implementierung von „Machine Vision“ in die HTLRIOT - Umgebung mit dem System:

**OpenMV Cam H7 R2  
+  
MT9M114 - Sensor**

## Inhalt

Grundfunktion – Blockschaltbild: .....	4
Hardware: .....	5
Gehäuse: .....	5
Oberleitung: .....	5
Befestigung: .....	5
Software: .....	6
Bilderfassung: .....	6
Sensorinitialisierung: .....	6
Snapshot speichern: .....	6
Objekterkennung: .....	6
Relativer Koordinatenursprung ermitteln: .....	7
Zugposition ermitteln: .....	7
Kalibrierung: .....	8
CIELAB-Threshold: .....	8
Area-Threshold: .....	9
Maßstabskalibrierung: .....	9
Offsetkalibrierung: .....	10
Übernehmen der Kalibrierung: .....	10
Referenzen: .....	11
Anhänge: .....	12
Anhang 1: Oberleitung: .....	12
Anhang 2: Befestigungsclip: .....	18
Anhang 3: Code-Implementierung .....	19

## Abbildungsverzeichnis

Abbildung 1: Grundfunktion-Blockschaltbild .....	4
Abbildung 2: Gehäuse - Deckel.....	5
Abbildung 3: Gehäuse - Korpus .....	5
Abbildung 4: Oberleitung .....	5
Abbildung 5: Befestigungsclip .....	5
Abbildung 6: Sensorinitialisierung.....	6
Abbildung 7: Snapshot speichern.....	6
Abbildung 8: Objekterkennung - find_blobs().....	6
Abbildung 9: Objekterkennung – Koordinatenursprung.....	7
Abbildung 10: Objekterkennung - Zugposition ermitteln .....	7
Abbildung 11: Schwellenwert-Editor öffnen .....	8
Abbildung 12: Schwellenwerteditor.....	8
Abbildung 13: Area-Threshold paint.net.....	9
Abbildung 14: Eintragen der Kalibrierung.....	10
Abbildung 15: Oberleitung - Gesamtaufbau .....	12
Abbildung 16: Oberleitung - Seitensteher.....	13
Abbildung 17: Oberleitung - Mittelsteher.....	14
Abbildung 18: Oberleitung - Querbalken .....	15
Abbildung 19: Oberleitung - Steher.....	16
Abbildung 20: Oberleitung - Quersteher.....	17
Abbildung 21: Befestigungsclip .....	18
Abbildung 22: Code-Implementierung Teil 1 .....	19
Abbildung 23: Code-Implementierung Teil 2 .....	20

## Grundfunktion – Blockschaltbild:

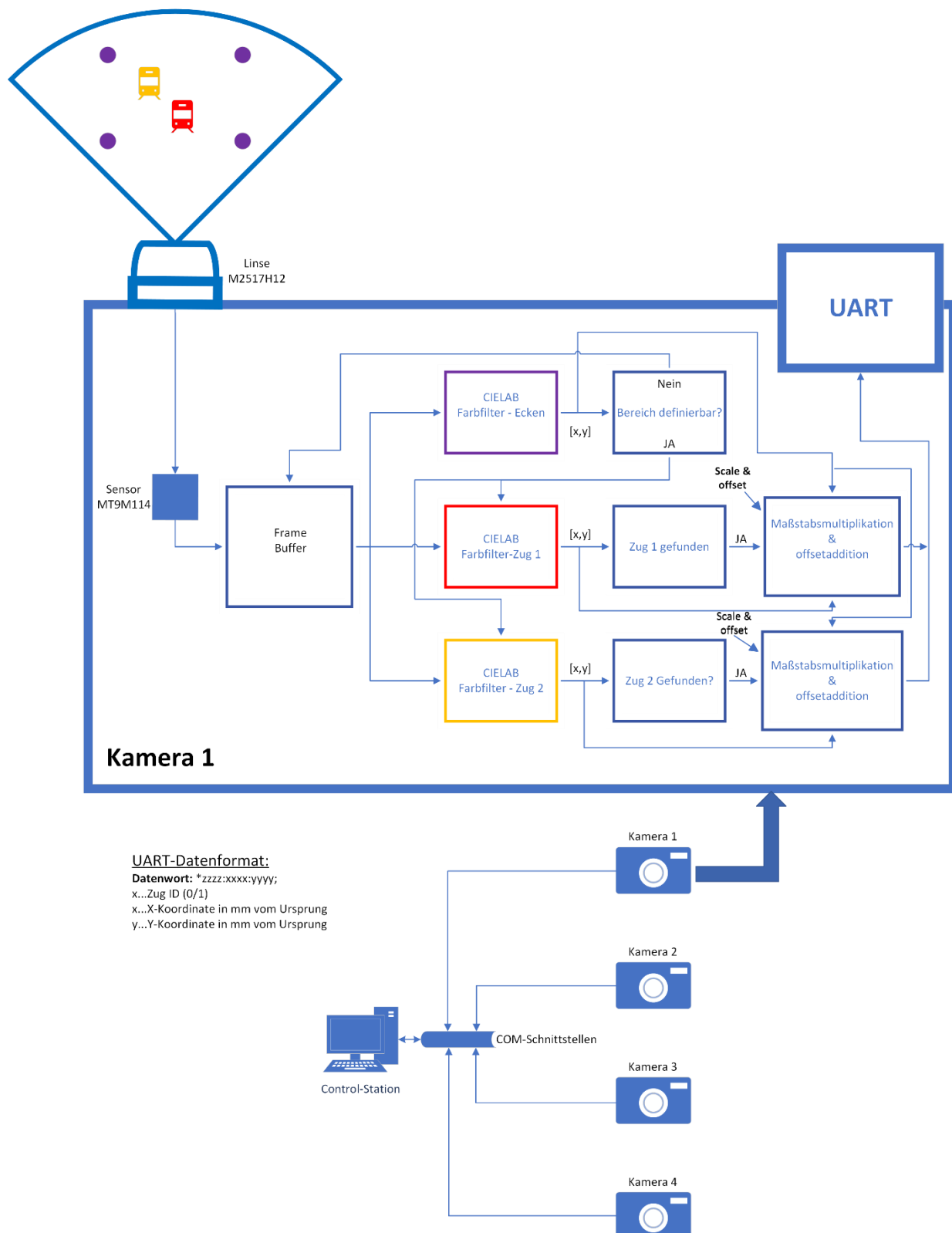


Abbildung 1: Grundfunktion-Blockschaltbild

Im Blockschaltbild ist zu erkennen wie das Bildmaterial von der Kamera verarbeitet wird und die daraus gewonnen Informationen an die Control-Station weiter gegeben werden. Jede der 4 Kameras besitzt dabei dieselbe Funktion, ist jedoch einem anderen Bereich zugeordnet. Aus diesem Grund gilt folglich überwiegend jede Erklärung für jede Kamera.

## Hardware:

### Gehäuse:

Das Gehäuse der Kamera soll die Funktion besitzen die Kamera vor mechanischen Belastungen zu schützen und sich gut dafür eignen modulare Befestigungen daran anzubringen. Ein Gehäuse, das diese Anforderungen erfüllt existiert bereits in der 3D-Bibliothek „Thingiverse“ und wird so verwendet.

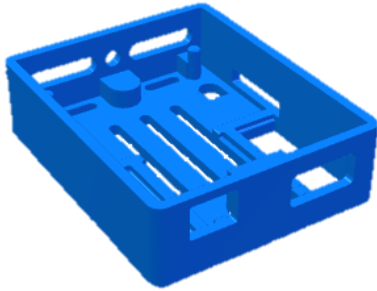


Abbildung 3: Gehäuse - Korpus



Abbildung 2: Gehäuse - Deckel

### Oberleitung:

Die Funktion der Oberleitung besteht darin eine Befestigungsmöglichkeit für Kamera und Licht darzustellen. Eine mögliche Ausführung einer Oberleitung sähe wie folgt aus:



Abbildung 4: Oberleitung

Detaillierte technische Zeichnungen dazu befinden sich unter „[Anhang 1: Oberleitung](#)“.

### Befestigung:

Zur Integration in das HTLR IOT System wird zusätzlich zum Gehäuse der Kamera ein Clip verwendet, um diese an der "Oberleitung" einfach und modular anbringen zu können.

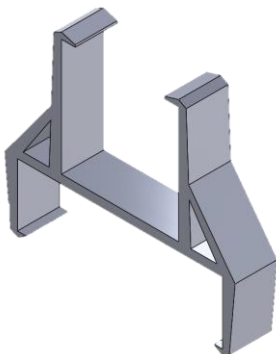


Abbildung 5: Befestigungsclip

Detaillierte technische Zeichnungen dazu befinden sich unter „[Anhang 1: Oberleitung](#)“.

## Software:

Die OpenMV-Kameras werden hauptsächlich über die speziell für diese Hardware entwickelte IDE mit MicroPython programmiert. Die IDE beinhaltet wichtige Bibliotheksfunktionen explizit für die OpenMV-Kameras erstellt. Die offizielle Dokumentation ist unter [Referenzen](#) vorzufinden. Im Folgenden werden die einzelnen Software-Abschnitte erläutert. Ein funktionsfähiger Code ist unter [Referenzen](#) verlinkt.

## Bilderfassung:

### Sensorinitialisierung:

Bei der Sensorinitialisierung ist in dieser Anwendung hauptsächlich darauf zu achten, hohe Bild- und Farbaufösungen zu erreichen. Mögliche Bildkorrekturen wie horizontale Spiegelung oder vertikale Drehung stellen den optionalen Teil der Initialisierung dar. Eine bewährte Konfiguration für den MT9M114 Bildsensor ist:

```
#Initialisierung des Sensors:
sensor.reset() #Rücksetzen der vorhandenen Einstellungen
sensor.set_pixformat(sensor.RGB565) #Farbformat für Pixel wählen, hier:
    RGB256 --> 256-bit Rot Grün Blau Zahl
sensor.set_framesize(sensor.QVGA) #Captureframe Auflösung festlegen, hier:
    QVGA 320x240 (QVGA = Quarter VGA ==> ein Viertel der VGA Auflösung)
sensor.set_vflip(False) # Optionale Bildkorrektur durch
    vertikale Ausrichtung
sensor.set_hmirror(False) # Optionale Bildkorrektur durch horizontale
    Spiegelung
sensor.skip_frames(time = 2000) # Frames überspringen um fehlerhafte
    Frames nach Sensorinitialisierung zu verhindern
```

Abbildung 6: Sensorinitialisierung

### Snapshot speichern:

Zum Speichern eines Snapshots muss nach der Initialisierung lediglich die entsprechende Funktion, auf eine Variable referenziert, aufgerufen werden. Besondere Vorsicht gilt es hierbei aber vor allem auf die Laufzeitspeicherverwendung zu legen, da die große Datenmenge besonders bei mehreren gespeicherten Bildern schnell zu Fehlern führt. Implementierung:

```
img = sensor.snapshot() #Zwischenspeichern des aktuellen Bilds
```

Abbildung 7: Snapshot speichern

## Objekterkennung:

Die Objekterkennung erfolgt über die Funktion `find_blobs()` welche das Bild auf markante Farbpunkte untersucht. Der Farb- und Flächenschwellwert muss dieser Funktion übergeben werden. Der Farbschwellwert ist mit dem CIELAB-Farbraum beschrieben und beinhaltet für die L, a und b Farbkoordinaten jeweils Minimal- und Maximalwert. CIELAB ist ein Farbraum der alle Farben in Geräteunabhängiger Form enthält und dadurch erlaubt eine verlustfreie Konvertierung von Farbinformationen durchzuführen. Die Farbkoordinate a beschreibt die Farb-Art und Farbintensität zwischen Grün und Rot und b zwischen Blau und Gelb jeweils mit einem Wert von -128 bis 127. Die L Farbkoordinate beschreibt die Helligkeit mit Werten von 0 bis 100. Der Flächenschwellwert beschreibt die minimale Größe die eine Farbfläche aufweisen muss um als erkannt zu gelten. Implementierung:

```
blobsCorner = img.find_blobs([thresholdCorner], area_threshold=25,
    merge=True) #Suchen nach Eck-Markierungen
```

Abbildung 8: Objekterkennung - find\_blobs()

Die Definierung von Thresholds wird in [CIELAB-Threshold](#) erläutert. Mit `merge=True` werden nahe aneinander liegende Farbflächen zusammengeführt.

### Relativer Koordinatenursprung ermitteln:

Um nach der Objekterkennung der bereichsdefinierenden Punkte, den Koordinatenursprung zur anschließenden Positionsbestimmung der Loks zu ermitteln, wurde eine Funktion erstellt. Diese sortiert alle Punkte zunächst absteigend abhängig von ihrer Y-Bildkoordinate. Als nächster Schritt wird die X-Bildkoordinate der ersten zwei Elemente des Arrays verglichen und jenes Element zurückgegeben welches die kleinere X-Bildkoordinate besitzt. Die Implementierung erfolgt so:

```
#Ermitteln der Bildkoordinate des relativen Koordinatenursprungs
def getOrigin(blobs):
    blobs.sort(key=lambda x: x.cy(), reverse=True)

    if (blobs[0].cx() < blobs[1].cx()):
        return blobs[0]
    else:
        return blobs[1]
```

Abbildung 9: Objekterkennung – Koordinatenursprung

### Zugposition ermitteln:

Nach der erfolgreichen Definierung eines Arbeitsbereiches und der Ermittlung des relativen Koordinatenursprungs wird, wie in [Objekterkennung](#) erläutert, nach Zügen gesucht. Kann ein Zug gefunden werden, erfolgt seine Koordinatenbestimmung mit der Berechnung der Differenz von Ursprung und Zugposition, Multiplikation des Ergebnisses mit dem Maßstabsfaktor zur Umwandlung in Millimeter und Addition bzw. Subtraktion der berechneten Koordinate mit dem Offset jeweils für den X- und Y-Wert. Nach der erfolgreichen Berechnung des Ergebnisses wird überprüft, ob sich die Koordinaten im definierten Arbeitsbereich befinden und wenn dies der Fall ist mit dem definierten Syntax über die `print()`-Methode an die Control-Station geschickt. Die Implementierung sieht wie folgt aus:

```
outputX = xPixelToMMScale*(blobsTrain2[0].cx() - blobOrigin.cx()) + xOffset
# Berechnen der relativen X-Koordinate in mm + Kompensation
outputY = yPixelToMMScale*(blobOrigin.cy() - blobsTrain2[0].cy()) + yOffset
# Berechnen der relativen Y-Koordinate in mm + Kompensation
if (outputX > 0 and outputY > 0): #Ausgabe der Zug-Koordinaten via UART
    print("*1:" + str(int(outputX)) + ":" + str(int(outputY)) + ";")
```

Abbildung 10: Objekterkennung - Zugposition ermitteln

## Kalibrierung:

### CIELAB-Threshold:

Da es nicht ausgeschlossen ist, dass sich z.B. Lichtverhältnisse im überwachten Bereich ändern und dadurch die Anhaltspunkte auf den Objekten für die Kamera in anderen Farben erscheinen, ist es umso wichtiger eine gute Kalibrierung vorzunehmen. Die OpenMV-IDE bietet dafür ein Tool wodurch sich die Bestimmung der Schwellwerte sehr einfach gestaltet.

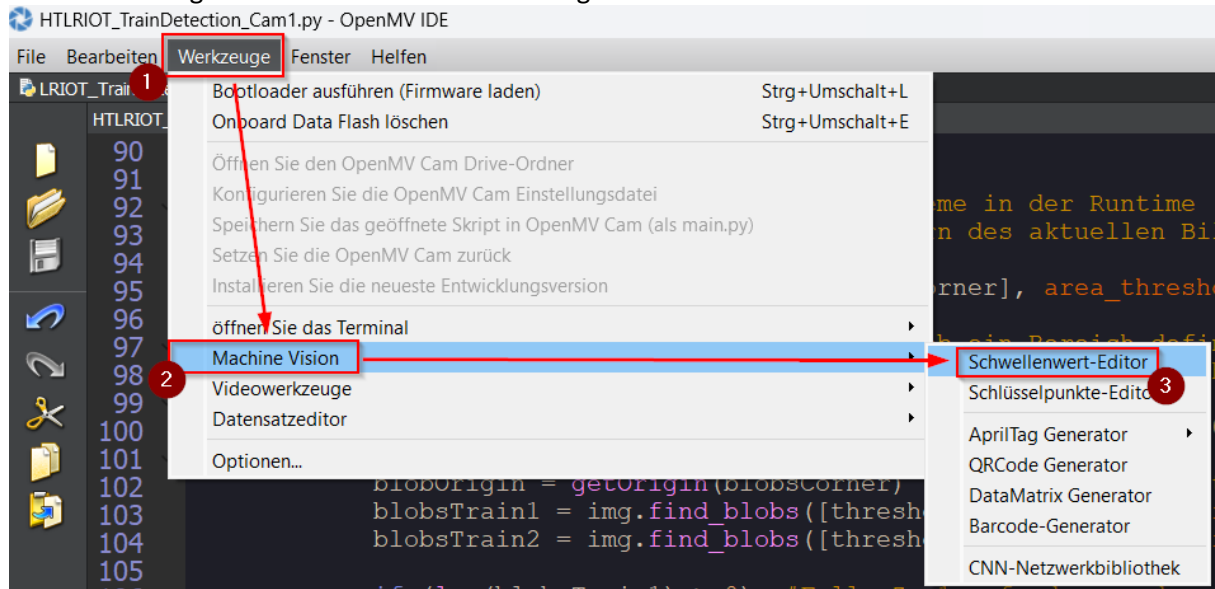


Abbildung 11: Schwellenwert-Editor öffnen

Nach dem Öffnen des Schwellenwert-Editors kann entweder ein gespeichertes Bild oder der aktuelle Snapshot einer angeschlossenen Kamera ausgewählt werden.

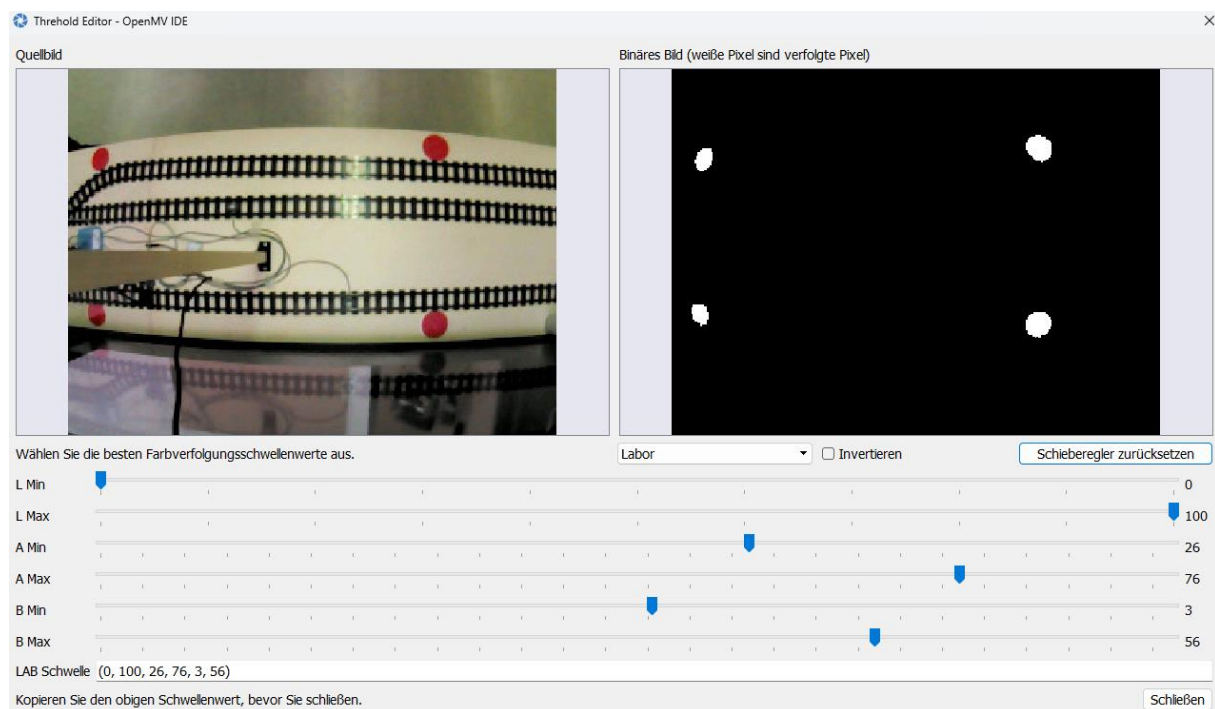


Abbildung 12: Schwellenwerteditor

Im Schwellenwerteditor wird der Schwellenwert nun so lange angepasst bis nur noch die gewünschten Farbbereiche weiß zu sehen sind. Es empfiehlt sich Farbbereiche anderer Objekte auch im Bild zu haben, um zu verhindern das die Schwellenwerte zu groß definiert und dadurch ungewollt andere Objekte mitberücksichtigt werden. (In Abbildung 12 nicht der Fall)



### Area-Threshold:

Der Area-Threshold kann durch unterschiedliche Arten festgestellt werden. Generell kann aber gesagt werden, lieber einen kleineren als zu großen Wert zu wählen. Ein ungefährer Richtwert lässt sich durch markieren einer Fläche im aktuellen Echtzeitbild oder durch verschiedene Bildbearbeitungsprogramme (hier paint.net) bestimmen.

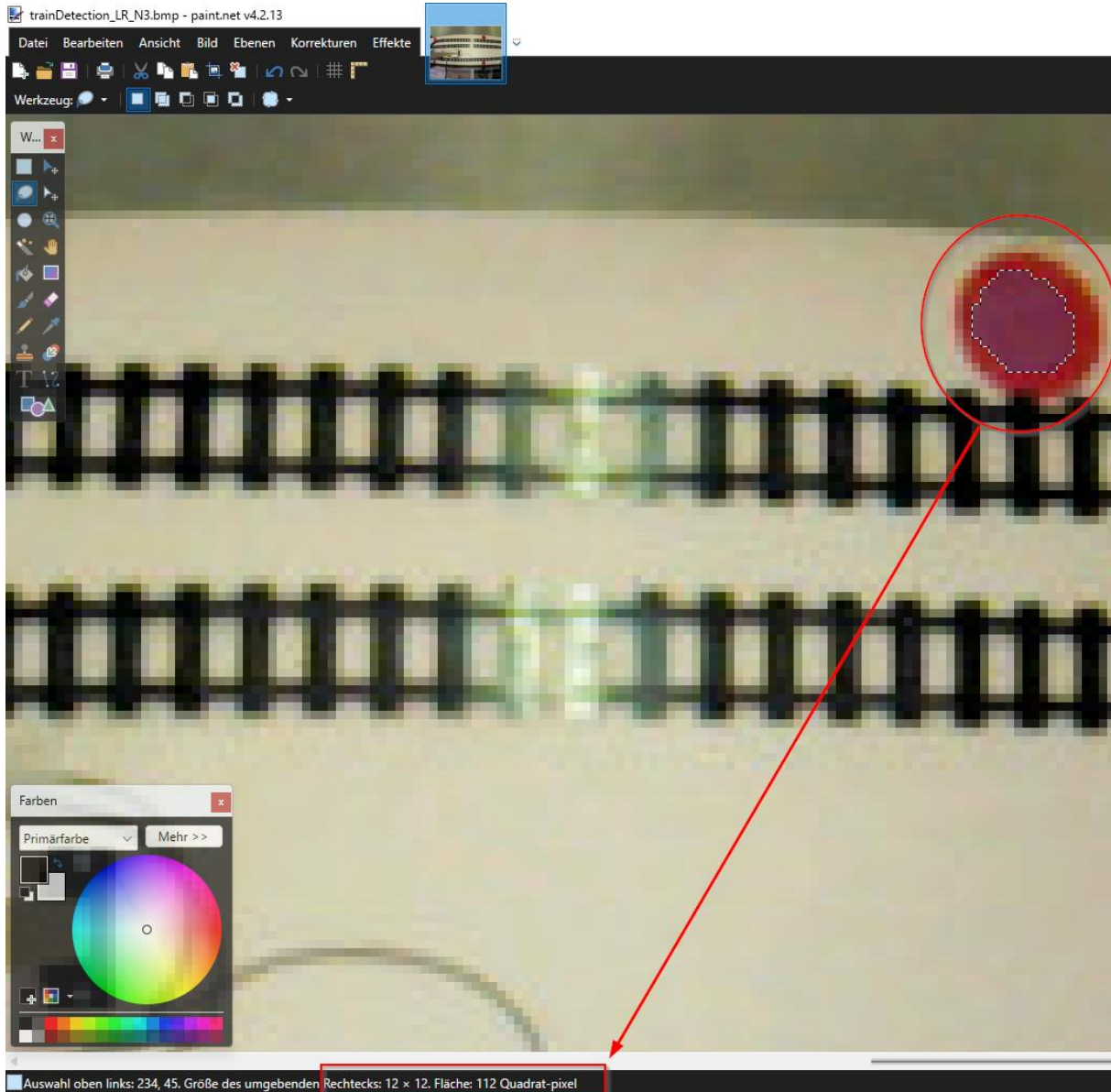


Abbildung 13: Area-Threshold paint.net

### Maßstabskalibrierung:

Die Methoden zur Maßstabskalibrierung sind vielfältig, eine bewährte Methode ist aber die Differenz von zwei Positionen jeweils in Pixel und in Millimetern zu ermitteln und durch die Division den Maßstab zu berechnen. Achtung: Der Maßstab der X-Achse kann, aufgrund des nicht quadratischen Bildformats, vom Maßstab der Y-Achse abweichen und muss deshalb separat bestimmt werden.

$$scale = \frac{\Delta mm}{\Delta Pixel}$$

### Offsetkalibrierung:

Zur Offsetkalibrierung wird die Differenz zwischen dem berechneten Istwert, nach der Maßstabskalibrierung, und dem gemessenen Sollwert ermittelt.

$$offset = Position_{Soll} - Position_{Ist}$$

### Übernehmen der Kalibrierung:

Um die Kalibrierung auf der Kamera zu übernehmen, müssen die Werte lediglich an den bestimmten Stellen der dazugehörigen Variablen im, auf der Kamera unter *main.py* gespeicherten, Code eingetragen werden. Nach einem Neustart werden die Änderungen sofort wirksam.

```
#Definieren der Farbbereiche zur Erkennung der Markierungen
thresholdCorner = (0, 100, 19, 71, -7, 28)
thresholdTrain1 = (0, 63, -25, 48, -47, -16)
thresholdTrain2 = (45, 75, -23, 5, 40, 80)
#Definieren der Massstabs- und Offsetkalibrierung
xPixelToMMScale = 5.03;
yPixelToMMScale = 5;
xOffset = 3318;
yOffset = 60;
```

Abbildung 14: Eintragen der Kalibrierung

## Referenzen:

- GitHub-Repository  
(<https://github.com/Skh4rf/HTLRIOT-OpenMV>)
- OpenMV Dokumentation  
(<https://docs.openmv.io>)
- Open MV Cam H7 R2  
(<https://openmv.io/collections/products/products/openmv-cam-h7-r2>)
- Gehäuse  
(<https://www.thingiverse.com/thing:4770533>)
- Oberleitung  
([https://github.com/Skh4rf/HTLRIOT-OpenMV/blob/main/doc/ref/HTLRIOT\\_ZugDemo\\_Haltegeruest.pdf](https://github.com/Skh4rf/HTLRIOT-OpenMV/blob/main/doc/ref/HTLRIOT_ZugDemo_Haltegeruest.pdf))
- Befestigungsclip  
([https://github.com/Skh4rf/HTLRIOT-OpenMV/blob/main/doc/ref/HTLRIOT\\_OpenMV\\_Befestigungsclip.pdf](https://github.com/Skh4rf/HTLRIOT-OpenMV/blob/main/doc/ref/HTLRIOT_OpenMV_Befestigungsclip.pdf))
- Sensor MT9M114  
(<https://github.com/Skh4rf/HTLRIOT-OpenMV/blob/main/doc/ref/MT9M114-D.PDF>)
- Mikrocontroller STM32H743VI  
(<https://github.com/Skh4rf/HTLRIOT-OpenMV/blob/main/doc/ref/stm32h743vi.pdf>)
- Software  
(<https://github.com/Skh4rf/HTLRIOT-OpenMV/tree/main/sw>)
- Hardware (Solidwork-Files, STL-Files, etc.)  
(<https://github.com/Skh4rf/HTLRIOT-OpenMV/tree/main/hw>)

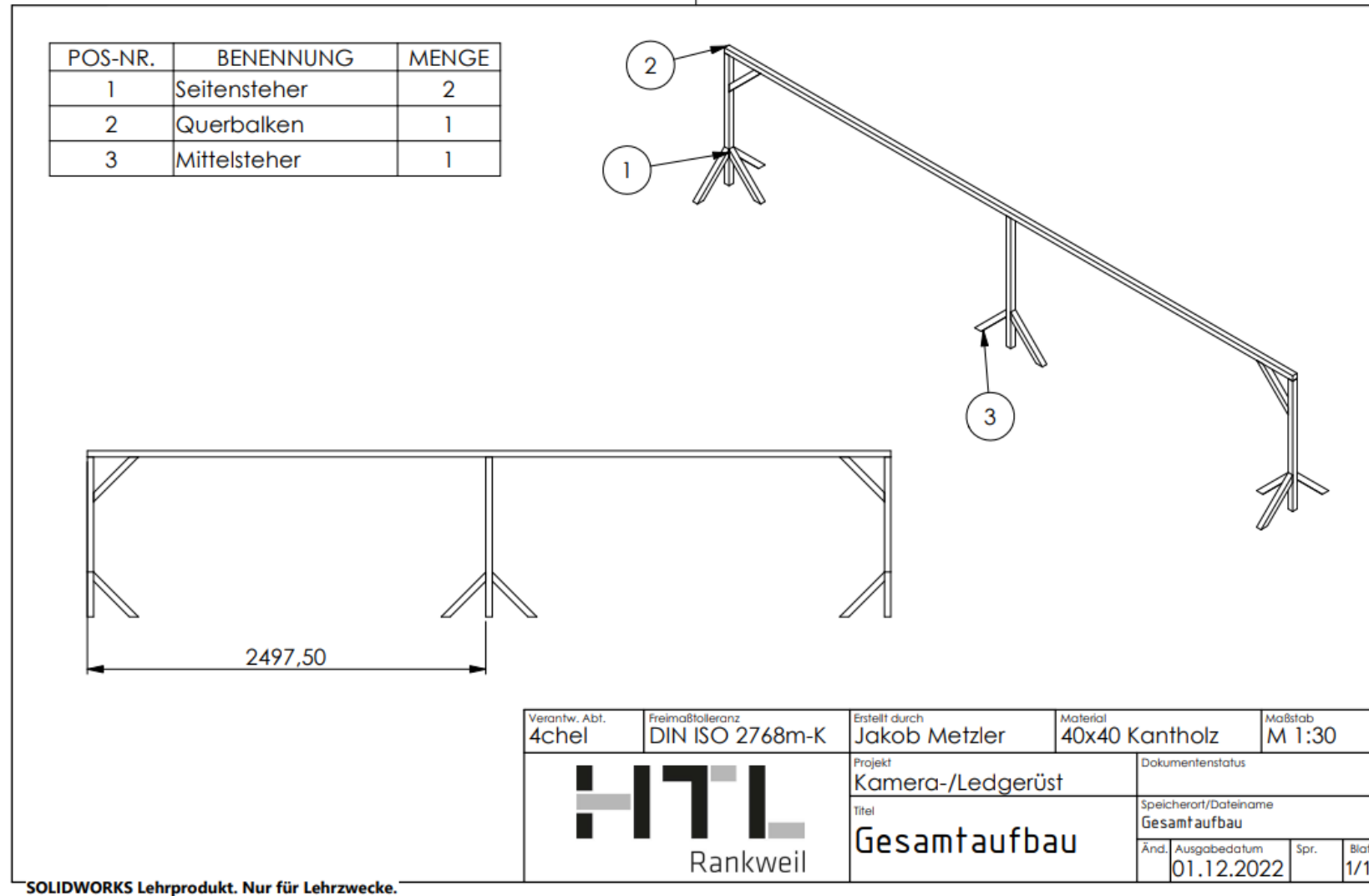
**Anhänge:****Anhang 1: Oberleitung:**

Abbildung 15: Oberleitung - Gesamtaufbau

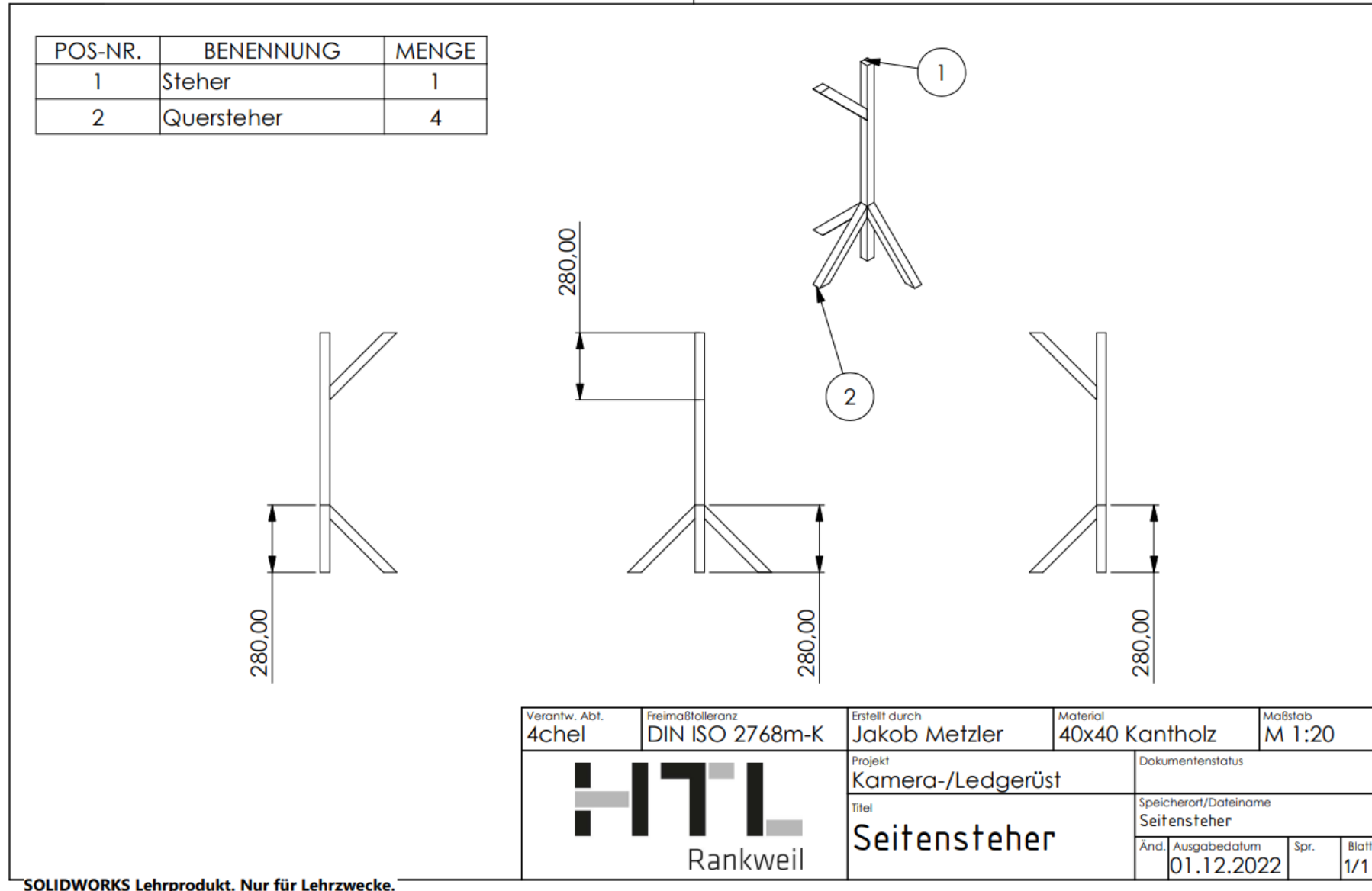
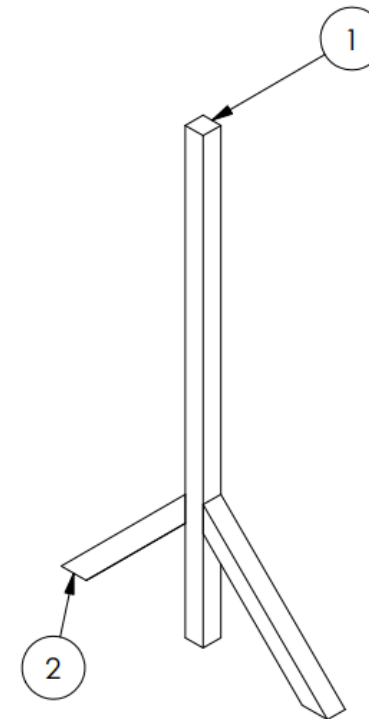
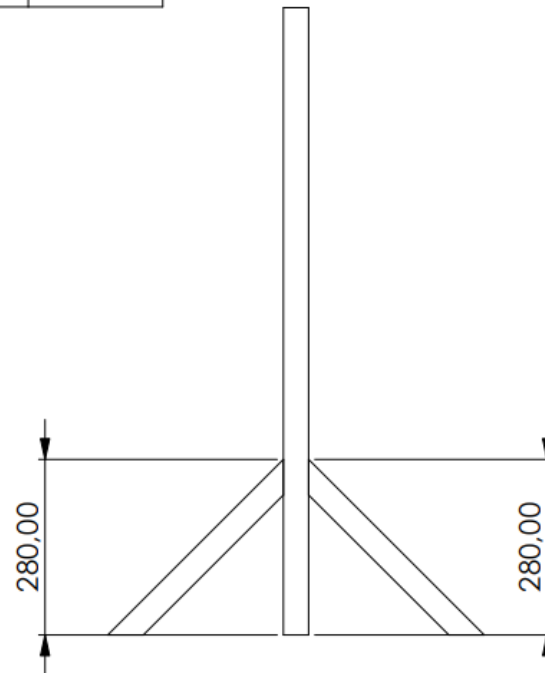



Abbildung 16: Oberleitung - Seitensteher

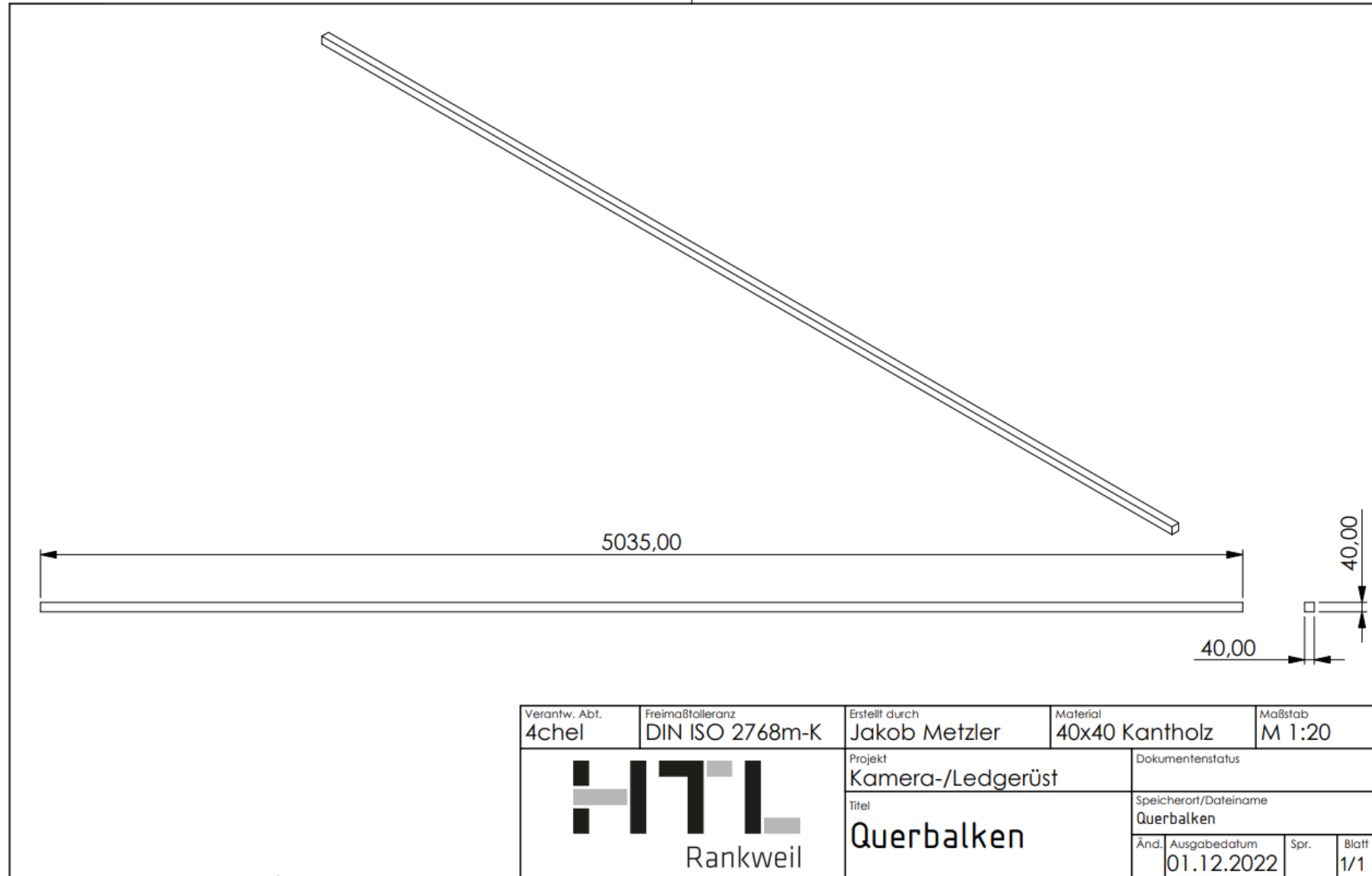
POS-NR.	BENENNUNG	MENGE
1	Steher	1
2	Quersteher	2



Verantw. Abt. 4chel	Freimaßtoleranz DIN ISO 2768m-K	Erstellt durch Jakob Metzler	Material 40x40 Kantholz	Maßstab M 1:10
 Rankweil		Projekt Kamera-/Ledgerüst	Dokumentenstatus	
		Titel Mittelsteher	Speicherort/Dateiname Mittelsteher	
			Änd.	Ausgabedatum 01.12.2022

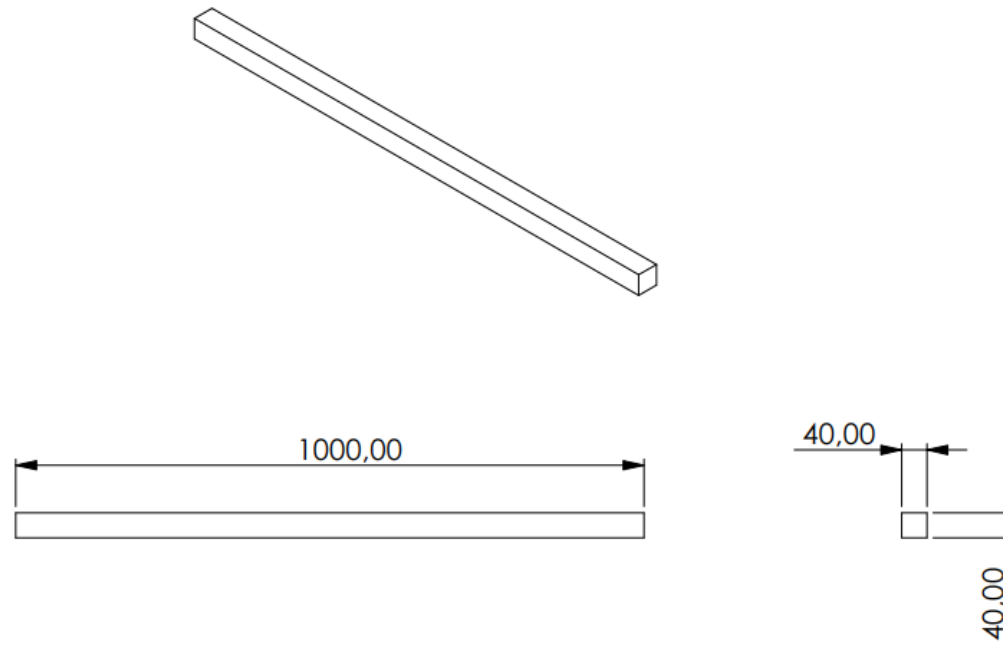
SOLIDWORKS Lehrprodukt. Nur für Lehrzwecke.


Abbildung 17: Oberleitung - Mittelsteher



SOLIDWORKS Lehrprodukt. Nur für Lehrzwecke.

Abbildung 18: Oberleitung - Querbalken

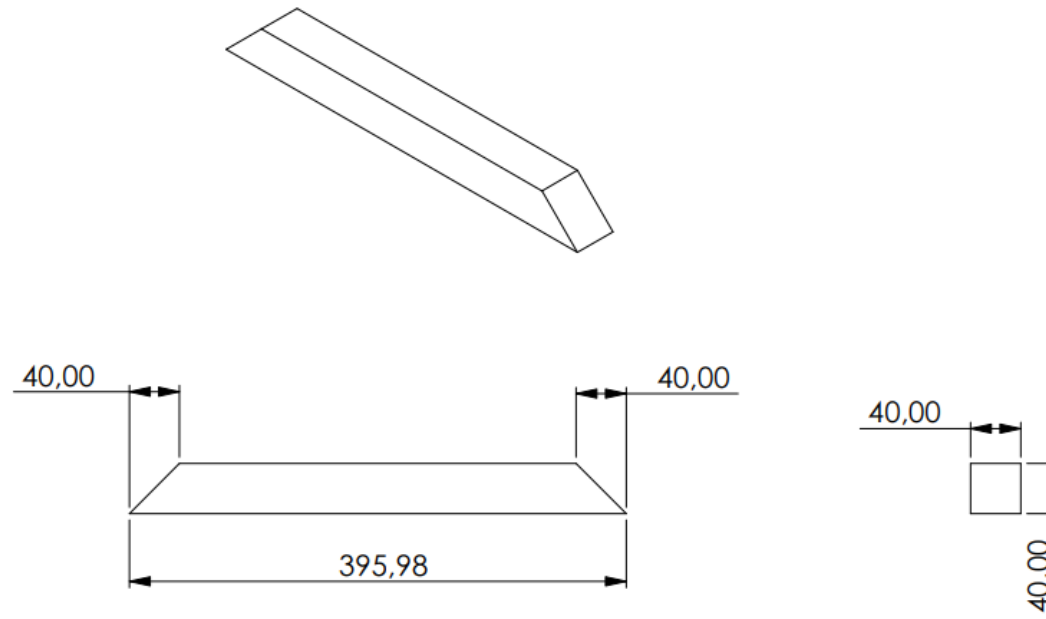



Verantw. Abt. 4chel	Freimaßtoleranz DIN ISO 2768m-K	Erstellt durch Jakob Metzler	Material 40x40 Kantholz	Maßstab M 1:10
 Rankweil		Projekt Kamera-/Ledgerüst	Dokumentenstatus	
		Titel Steher	Speicherort/Dateiname Steher	
			Änd.	Ausgabedatum 01.12.2022
			Spr.	Blatt 1/1

SOLIDWORKS Lehrprodukt. Nur für Lehrzwecke.

Abbildung 19: Oberleitung - Steher





Verantw. Abt. 4chel	Freimaßtoleranz DIN ISO 2768m-K	Erstellt durch Jakob Metzler	Material 40x40 Kantholz	Maßstab M 1:5
 Rankweil		Projekt Kamera-/Ledgerüst	Dokumentenstatus	
		Titel Quersteher	Speicherort/Dateiname Quersteher	
		Änd.	Ausgabedatum 01.12.2022	Spr. Blatt 1/1

SOLIDWORKS Lehrprodukt. Nur für Lehrzwecke.

Abbildung 20: Oberleitung - Quersteher

## Anhang 2: Befestigungsclip:

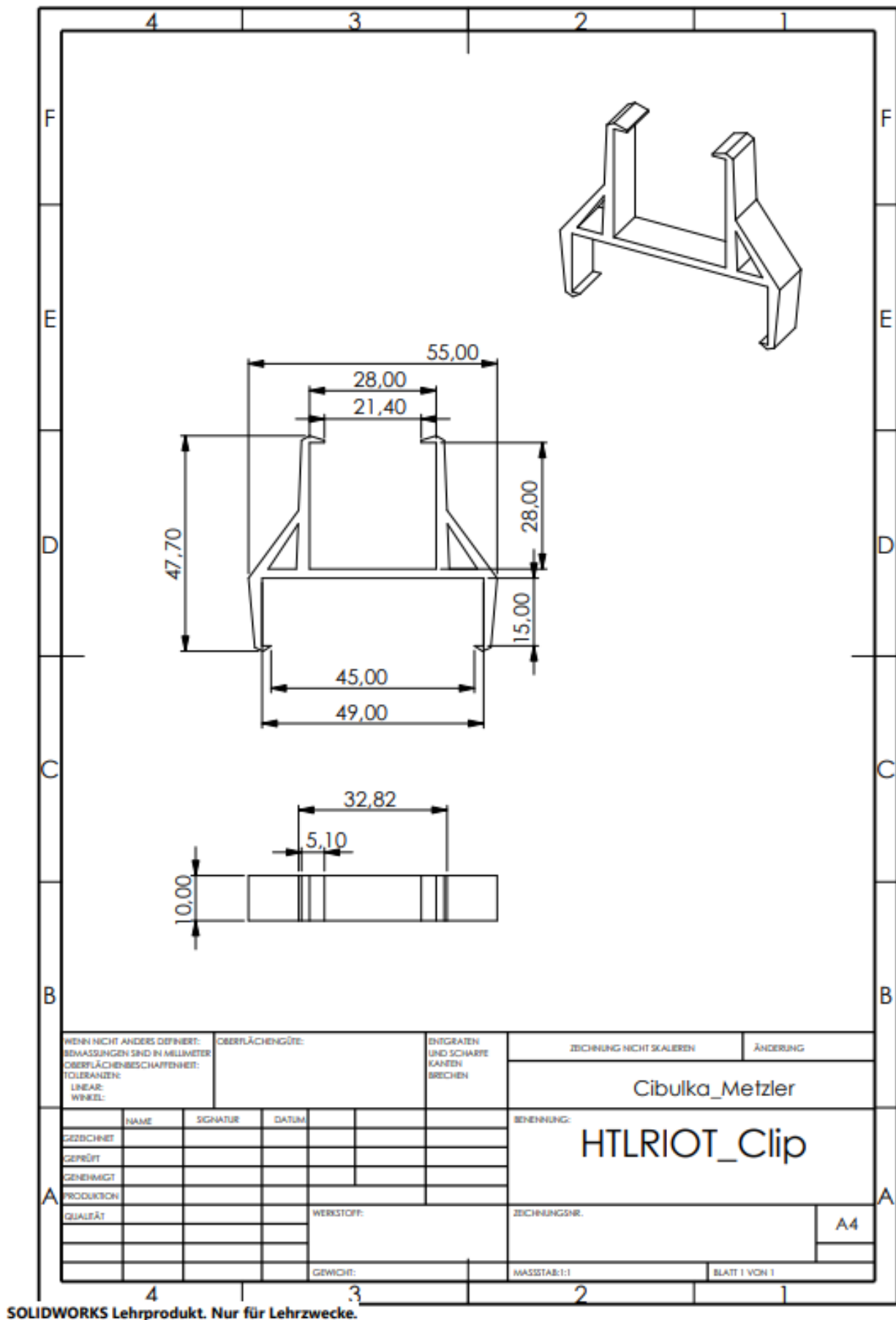


Abbildung 21: Befestigungsclip

## Anhang 3: Code-Implementierung

#Auhor: Jakob Metzler, Otto Cibulka

```
#Importieren der benötigten µPython-Bibliotheken:
import pyb, sensor, image, time
from pyb import UART

#Initialisierung des Sensors:
sensor.reset() #Rücksetzen der vorhandenen Einstellungen
sensor.set_pixformat(sensor.RGB565) #Farbformat für Pixel wählen, hier:
    RGB256 --> 256-bit Rot Grün Blau Zahl
sensor.set_framesize(sensor.QVGA) #Captureframe Auflösung festlegen, hier:
    QVGA 320x240 (QVGA = Quarter VGA ==> ein viertel der VGA Auflösung)
sensor.set_vflip(False) # Optionale Bildkorrektur durch vertikale
    Ausrichtung
sensor.set_hmirror(False) # Optionale Bildkorrektur durch horizontale
    Spiegelung
sensor.skip_frames(time = 2000) # Frames überspringen um fehlerhafte Frames
    nach Sensorinitialisierung zu verhindern

#Definieren von Objekten der Onboard-RGB-Led
red_led = pyb.LED(1)
green_led = pyb.LED(2)
blue_led = pyb.LED(3)

#Ermitteln der Bildkoordinate des relativen Koordinatenursprungs
def getOrigin(blobs):
    blobs.sort(key=lambda x: x.cy(), reverse=True)

    if (blobs[0].cx() < blobs[1].cx()):
        return blobs[0]
    else:
        return blobs[1]

#RGB Led Rot setzen
def setLedRed():
    red_led.on()
    green_led.off()
    blue_led.off()

#RGB Led Grün setzen
def setLedGreen():
    red_led.off()
    green_led.on()
    blue_led.off()

#RGB Led Blau setzen
def setLedBlue():
    red_led.off()
    green_led.off()
    blue_led.on()

#Arbeitsschleife
while(True):
    time.sleep(0.01)
    #Definieren der Farbbereiche zur Erkennung der Markierungen
    thresholdCorner = (0, 100, 19, 71, -7, 28)
    thresholdTrain1 = (0, 63, -25, 48, -47, -16)
    thresholdTrain2 = (45, 75, -23, 5, 40, 80)
    #Definieren der Massstabs- und Offsetkalibrierung
    xPixelToMMScale = 5.03;
    yPixelToMMScale = 5;
    xOffset = 3318;
    yOffset = 60;
```

Abbildung 22: Code-Implementierung Teil 1

```

try: # Abfangen möglicher Frame-Buffer Probleme in der Runtime
    img = sensor.snapshot() #Zwischenspeichern des aktuellen Bilds

    blobsCorner = img.find_blobs([thresholdCorner], area_threshold=25,
                                  merge=True) #Suchen nach Eck-Markierungen

    if (len(blobsCorner) == 4): #Überprüfen ob ein Bereich definiert
        werden kann
        setLedGreen() #LED grün setzen um Erfolg bei der
        Bereichdefinierung anzuzeigen
        for b in blobsCorner:
            img.draw_circle(b.x(), b.y(), 10, color = (255, 0, 0))

        blobOrigin = getOrigin(blobsCorner) #Ermitteln des relativen
        Koordinaten Ursprungs (links unten)
        blobsTrain1 = img.find_blobs([thresholdTrain1],
                                      area_threshold=25, merge=True) #Suchen nach Zug1 im
        definierten Bildbereich
        blobsTrain2 = img.find_blobs([thresholdTrain2],
                                      area_threshold=20, merge=True) #Suchen nach ZUg2 im
        definierten Bildbereich

        if (len(blobsTrain1) > 0): #Falls Zug1 gefunden wurde
            setLedBlue() #Indikator-Led blau setzen um gefundenen Zug
            anzuzeigen
            img.draw_circle(blobsTrain1[0].x(), blobsTrain1[0].y(), 10,
                            color = (0,255,0))
            outputX = xPixelToMMScale*(blobsTrain1[0].cx()-
                                      blobOrigin.cx()) + xOffset # Berechnen der relativen
            X-Koordinate in mm + Kompensation
            outputY = yPixelToMMScale*(blobOrigin.cy()-
                                      blobsTrain1[0].cy()) + yOffset # Berechnen der
            relativen Y-Koordiante in mm + Kompensation
            if (outputX > 0 and outputY > 0): #Ausgabe der Zug
            Koordinaten via USB
                print("*0:" + str(int(outputX)) + ":" +
                      str(int(outputY)) + ";")

        if (len(blobsTrain2) > 0): #Falls Zug2 gefunden wurde
            setLedBlue() #Indikator-Led blau setzen um gefundenen Zug
            anzuzeigen
            img.draw_circle(blobsTrain2[0].x(), blobsTrain2[0].y(), 10,
                            color = (0,0,255))
            outputX = xPixelToMMScale*(blobsTrain2[0].cx()-
                                      blobOrigin.cx()) + xOffset # Berechnen der relativen
            X-Koordinate in mm + Kompensation
            outputY = yPixelToMMScale*(blobOrigin.cy()-
                                      blobsTrain2[0].cy()) + yOffset # Berechnen der
            relativen Y-Koordinate in mm + Kompensation
            if (outputX > 0 and outputY > 0): #Ausgabe der Zug-
            Koordinaten via UART
                print("*1:" + str(int(outputX)) + ":" +
                      str(int(outputY)) + ";")

    else:
        setLedRed() #Indikator-Led rot setzen falls kein definierbarer
        Bereich gefunden wurde
        time.sleep(0.1)
except:
    time.sleep(0.1) #Eine Sekunde warten um im falle eines Runtime
    Frame-Errors die Capture wieder neu laden zu lassen

```

Abbildung 23: Code-Implementierung Teil 2