Mettmann

Studienarbeit

Projekt Algorithmen

Prüfer:

Herr Thomas Seifert

Verfasser:

Emre Il

Fester Straße 22

40882 Ratingen

Matrikelnummer : 101673

Studiengang : Informatik

Spezialisierungsbereich : Angewandte

Abgabetermin:

27. November 2023

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Problemstellung

In dieser Studienarbeit geht es um ein Projekt zur Erkennung von Objekten. Dabei soll das Programm eine Echtzeiterkennung von vorher festgelegten Objekten durch eine Kamera durchführen. Das Programm soll in der Programmiersprache Python geschrieben werden, mit der Bibliothek Opencv. Auch sind Vorgaben für die zu nutzenden Objekte gestellt, wie:

- Jedes Objekt besitzt genau eine Farbe
- Es gibt mindestens 4 Objektformen.
- Es gibt mindestens 4 Objektfarben
- Pro Objektform gibt es mindestens 2 Farben
- Mindestens 6 Objekte kommen jeweils mindestens zweimal vor

Das Projekt umfasst die Implementierung von Bildverarbeitungstechniken, um Objekte in Echtzeit zu identifizieren, ihre Konturen zu extrahieren und schließlich ihre Position zu bestimmen. Auf einem DIN-A3-Blatt sind zwei Koordinaten markiert, welche die Punkte (0,0) und (1,1) darstellen sollen. Alle Objekte werden sich innerhalb der beiden Punkte befinden, und die Position der gesuchten Objekte wird sich durch die Position auf dem DIN A3 bestimmen. Dabei sind Faktoren wie die Beleuchtung, die Hintergrundumgebung und mögliche Verdeckungen der Objekte zu berücksichtigen.

Die Aufgabe der Python-Anwendung liegt darin, die Objekte, nachdem diese auf dem DIN-A3-Blatt zufällig positioniert wurden, zu identifizieren und die Position von einer ausgewählten Form und Farbe auszugeben. Bei häufiger vorkommenden Objekten sollen alle Positionen erfasst werden.

2 Beschreibung der Lösung aus Anwendersicht

Im folgenden Abschnitt wird die Sicht des Programms durch den Anwender erläutert.

Bei der Inbetriebnahme der Anwendung wird dem Anwender die Echtzeit-Aufnahme der Kamera dargestellt, woraufhin sich auch die Bedienung verwenden lässt. Dem Anwender wird in der Anwendung die Auswahl der verschiedenen gebotenen Lego-Steine dargestellt, wie z. B. 2x3 Blau, 2x3 Grün oder auch 2x6 Orange. Diese Eine Auswahl ermöglicht dem Anwender eines, durch iterierte Lego-Steine auszuwählen, um die gleichartigen Steine in der Echtzeit-Aufnahme auszuwählen und in Echtzeit zu markieren. Zudem bietet das Programm die Nutzung von vordefinierten Tasten,

die jeweils verschiedene Funktionen erfüllen. Eine wesentliche Tastatureingabe ist das Erstellen des Koordinatensystems anhand der aktuellen Aufnahme, indem um das zur Aufnahme festgehaltene Bild mit einem Koordinatensystem abgebildet wird, welches den Standort des ausgewählten Lego-Steins mit seinem X- und Y-Wert angibt. Der Start- und Endpunkt des Koordinaten-Systems werden in der Echtzeit-Aufnahme durch zwei definierte Kreise bestimmt, wobei ein gelber Kreis den Startpunkt (0,0) und ein roter Kreis den Endpunkt (1,1) angibt.

Die Anwendung bietet nicht nur die Funktion, genau definierte Lego-Steine auszuwählen, wie z. B. 2 × 3 Blau, sondern auch alle Lego-Steine, die einer Farbe angehören. Somit ist es dem Anwender auch möglich, z. B. alle blauen Lego-Steine auszuwählen, wobei irrelevant ist, ob es sich um einen 2×3 oder 2×6-Stein handelt.

3 Beschreibung von ausgewählten Aspekten aus Entwicklersicht

In diesem Abschnitt werden die technischen Schritte und Lösungswege erläutert sowie weitere Bibliotheken und Konzepte vorgestellt Für die Herausforderung, die Lego-Steine zu identifizieren, wurde in diesem Projekt das Machine-Learning-Tool Tensorflow eingesetzt. Die Einbindung in das Projekt und die Strategie zur Erkennung der einzelnen Steine werden im Verlauf des Textes erläutert.

Da es sich bei Aufnahmen durch die Kamera um einzelne aufeinander folgende Bilder handelt, können alle Bildverarbeitungsfunktionen angewendet werden, und im folgenden Text werden die Aufnahmen der Kamera als Bilder bezeichnet.

Schritt 1: Bildverarbeitung

Die erste Herausforderung des Projekts liegt in der Bildverarbeitung. Genauer gesagt müssen die Legosteine und ihre Konturen durch die Kamera identifiziert werden. Für diesen Schritt wird die vorgegebene Python-Bibliothek OpenCV genutzt.

Zur Verarbeitung des Bildes werden bekannte Funktionalitäten wie die Konvertierung der Farbeinstellungen, das Ausgrauen des Bildes, Blur und Kantenerkennung eingesetzt.

Der erste Schritt ist das Ausgrauen des Bildes. Damit die Konturen besser erkannt werden können, wird versucht, die Datenmenge zur Verarbeitung der Bilder zu reduzieren und Einflüsse wie Rauschen und Helligkeiten und die Komplexität verschiedener Farben zu vermeiden. OpenCV bietet hierfür eine Funktionalität zur Konvertierung.

Eine weitere Funktionalität, welche bei der Bildverarbeitung standardmäßig eingesetzt wird, ist das Glätten oder Blurren des Bildes. Wie beim Ausgrauen werden rauschende Faktoren reduziert, und dazu kommt die Erhaltung von deutlichen und wesentlichen Strukturen im Bild wie Kanten. OpenCV bietet die GaussianBlur-Funktion zur Glättung des Bildes.

Bevor die Kontouren erfasst werden können, müssen die Kanten aus dem Bild gefunden werden. Hierfür bietet OpenCV die Funktion Canny. Canny ist dafür bekannt, präzise Kanten zu identifizieren. Der Algorithmus lokalisiert Kanten im Bild und liefert schmale, genau positionierte Kantenlinien.

Nicht zu vergessen ist, dass diese Schritte miteinander verbunden sind. Das ausgegraute Bild wird als Parameter der GaussianBlur-Funktion übergeben, das bedeutet, dass die Glättung auf das ausgegraute Bild angewendet wird. Ebenso wird das Bild, welches der Canny-Funktion gegeben wird, diese beiden Konvertierungen beinhalten. Nach diesen Verarbeitungsschritten können die Kontouren durch OpenCV endeckt werden und bieten uns somit die Möglichkeit, alle Legosteine zu markieren.

Schritt 2: Tensorflow für die Objekt-Erkennung

TensorFlow ist eine leistungsstarke Open-Source-Bibliothek für maschinelles Lernen und Deep Learning, die von Google entwickelt wurde. Sie ermöglicht Entwicklern, komplexe neuronale Netzwerke für verschiedene Anwendungen zu entwerfen, zu trainieren und bereitzustellen. Die Flexibilität von TensorFlow erstreckt sich über einfache Klassifikationsaufgaben bis hin zu komplexen Sequenzmodellen. Besonders geeignet ist TensorFlow für Deep-Learning-Anwendungen, darunter Bilderkennung, natürliche Sprachverarbeitung und mehr. Die Anwendungen von TensorFlow erstrecken sich über Bilderkennung und Sprachverarbeitung bis hin zu autonomen Fahrzeugen und medizinischer Bildgebung, wodurch es zu einem wesentlichen Werkzeug für maschinelles Lernen wird. [1]

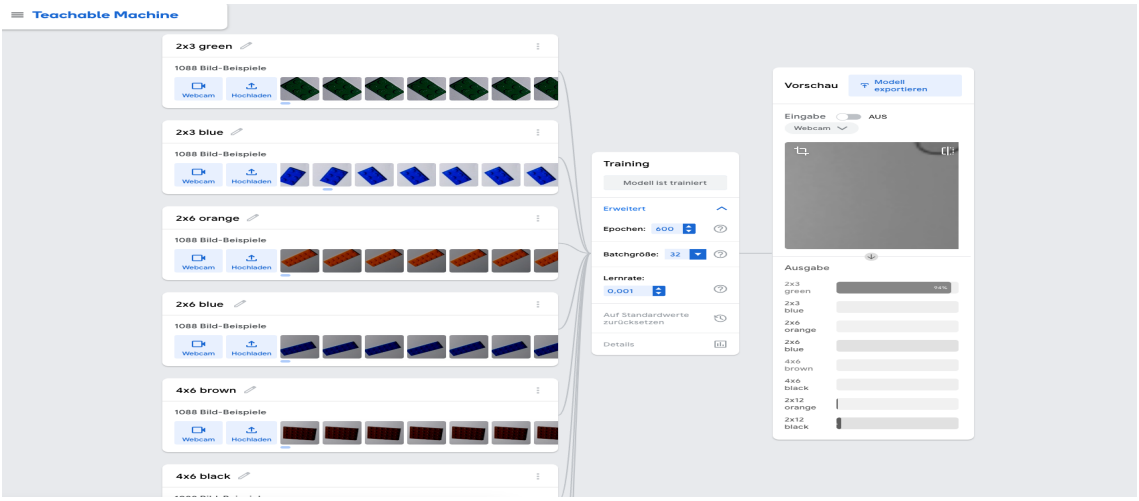
Für die Erkennung meiner spezifisch ausgewählten Objekte wird ein trainiertes Modell benötigt, wobei es sich um ein Tensorflow-Savermodell handeln kann oder um ein Keras-Modell. In meinem Fall entschied ich mich für das Kerasmodell. Keras,

[1] vgl Hope, Tom, Yehezkel S. Resheff und Itay Lieder (2017), S. 1

als benutzerfreundliche High-Level-Neural-Network-Bibliothek, ist in TensorFlow integriert und ermöglicht eine schnelle Entwicklung von Prototypen für neuronale Netzwerke. Mit einer einfachen und intuitiven API gestaltet sich die Erstellung von Modellen als effizient und leicht verständlich. Keras bietet Zugriff auf vorab trainierte Modelle für verschiedene Aufgaben wie Bilderkennung und Sprachverarbeitung, was die Entwicklung und Anpassung von Modellen vereinfacht. [2]

Damit ein Modell trainiert werden kann, müssen Bilder der verschiedenen Objekte erstellt werden. Für das Trainieren des Modells wurde die Webseite Teachable Machine von Google eingesetzt. Teachable Machine vereinfacht das Erstellen von Machine-Learning-Modellen und bietet eine interaktive Seite, um verschiedene Klassen zu erstellen und die Kamera zu nutzen, um schneller eine große Anzahl von Bildern von den Objekten machen zu können. Eine Klasse repräsentiert hierbei ein bestimmtes Objekt und durch die erstellten Klassen kann das Modell trainiert werden. Mit Python können diese Modelle eingebunden werden und bieten die Möglichkeit, Objekte aus einem Bild zu erkennen. [3]

Abbildung 1: Teachable machine



Ein Problem bei der Arbeit mit Tensorflow-Objekterkennung ist, dass die Funktion zur Erkennung von Objekten nicht mehrere Objekte gleichzeitig erkennen kann. Die Aufgabe erfordert jedoch, dass jedes Objekt erkannt wird und die Koordinaten am Ende ausgegeben werden. Damit die Aufgabe erfüllt werden kann, war es erforderlich, die bereits erkannten Konturen zu nutzen, um die Objekte aus dem Bild auszuschneiden.

Schritt 3: Koordinaten-System Aufbau

[2] vgl *Keras* (2023)
[3] vgl *Teachable Machine 2.0 makes AI easier for everyone* (2019)

Im folgenden Schritt wird erläutert, wie im vorliegenden Projekt ein Koordinatensystem aufgebaut wird, um die Lego-Steine zu lokalisieren und darzustellen. Zur Initialisierung des Koordinatensystems muss ein Start- und Endpunkt definiert werden, um die X-Achse und die Y-Achse darauf abbilden zu können. Hierzu werden Kreise verwendet. Um die beiden Punkte abzubilden. In dem Projekt definiert der gelbe Kreis den Punkt (0, 0), welcher der Anfang des Koordinatensystems ist, und der rote Kreis den Punkt (1, 1), welcher den Endpunkt darstellen soll. Hiermit sind die Grundeigenschaften bestimmt, und nun kann es zur Erstellung des Koordinatensystems kommen.

Aus technischer Sicht beginnt die Erstellung des Koordinatensystems mit der Lokalisierung der beiden Punkte. Sobald die beiden Punkte vom Programm gefunden wurden, wird herausgesucht, bei welchem der beiden Punkte es sich um den gelben und bei welchem es sich um den roten Punkt handelt. Nachdem die Farb-Erkennung erledigt ist, werden die Mittelpunkte der beiden Kreise errechnet und alles im Gelben-Zentrum und Roten-Zentrum gespeichert.

Zuletzt wird das Koordinaten-System aufgebaut, indem durch die beiden identifizierten Punkte die Längen der X-Achse und der Y-Achse berechnet werden.

Schritt 4: Koordinaten der Lego-Steine berechnen

Nachdem im vorherigen Schritt 3 das Koordinatensystem aufgebaut wurde, mithilfe der zwei farbigen Kreise und der Berechnung der X-Achse und Y-Achse, wird nun auf die Berechnung der Lego-Steine eingegangen.

In diesem Schritt werden die Kontouren aus dem ersten Schritt verwendet, um daraus den Mittelpunkt des Lego-Steins zu errechnen. Darauf folgend werden die Kontouren des Lego-Steins zur Findung der Koordinaten im Koordinatensystem verwendet und normalisiert.

4 Fazit

Im Fazit wird auf die Probleme eingegangen, die im Laufe des Projekts aufgetreten sind und welche Resultate am Ende herausgekommen sind, im Vergleich zu den Anfangserwartungen.

Als Erstes wird sich den auftretenden Problemen gewidmet, die bei der Entwicklung der Objekterkennung aufgetreten sind und wie sie das Projekt beeinflusst haben. Ein wesentliches Problem in dem Projekt war das Training des Machine-Learning-Modells, da auch nach längerem Training und Erweiterung der Datensets eine große

Ungenauigkeit bei der Erkennung der Echtzeit-Aufnahmen aufgetreten ist. Dies ist auf die Komplexität des Trainings der Machine-Learning-Modelle zurückzuführen, da es für das anständige Training des Modells zur genaueren Erkennung der Lego-Steine eine sehr große Menge an Trainingsdaten und Trainingszeit benötigt wird. Selbst nach einer großen Bild-Sammlung aus der Nähe, mit einer Anzahl an Bildern im vierstelligen Bereich für jeden Lego-Stein, hatte das Programm eine hohe Fehlerquote beim Identifizierung des richtigen Lego-Steins. Kleine Faktoren, die bereits einen wesentlichen Unterschied beim Training des Modells ausmachen, wären z. B. falsche Lichteinfälle.

Zu den guten Resultaten bei der Entwicklung des Programms gehört, dass die Lokalisierung der Lego-Steine im Koordinaten-System gute Resultate erbracht hat. Hierzu gehören die Erkennung der Kontouren der aufgenommenen Lego-Steine und die Abbildung des Koordinaten-Systems durch die Erkennung der beiden Punkte zum Start und Ende des Koordinatensystems. Der Mittelpunkt der beiden Punkte und die Abbildung des Koordinaten-Systems daraus sowie die Berechnung des Mittelpunkts des Lego-Steins verliefen erfolgreich.

Anhang

Anhangsverzeichnis

Anhang 1 main.py

```
   import cv2 as cv
import tensorflow as tf
import numpy as np
from coordinate_system import CoordinateSystem
from detector import Detector
from color_palatte import ColorPalette as cp

labels = []
legend = []

# Create a legend which shows all options
def create_legend():
    global text
    legend.append("s show coordinate system")
    legend.append("a show all legos")
    legend.append("b show 2x4 blue lego")
    legend.append("l show 4x6 black lego")
    legend.append("p disable all options")
    legend.append("1 show all blue legos")
    legend.append("2 show all orange legos")

# loading the keras model and the labels
def load_model():
    global labels
    model_path ="data/main_model/keras_model.h5"
    model = tf.keras.models.load_model(model_path)
    lable_path ="data/main_model/labels.txt"
    with open(lable_path, "r") as file:
        labels = file.read().strip().split('\n')


    return model

def print_labels(frame):
    global legend
    text_start_position = (frame.shape[1] - 300, 50)
    for i, text in enumerate(legend):
```

```
        # Calculate position for the text
        text_position = (text_start_position[0], text_start_position[1] + i * 5
        cv.putText(frame, text, text_position, cv.FONT_HERSHEY_SIMPLEX, 0.5, (0
        cv.imshow("Main Frame", frame)


def run():
    global labels
    global legend

    cs = CoordinateSystem()
    dt = Detector()
    model = load_model()

    create_legend()

    # set all options first disabled
    lego_object = []
    blueLego = False
    findAll = False
    find2x6Blue = False
    find4x6Black = False
    orangeLego = False

    cap = cv.VideoCapture(1)
    cap.set(3, 1280)
    """cap.set(cv.CAP_PROP_AUTOFOCUS, 0) """

    if not cap.isOpened():
        print("Cannot open camera")
        exit(1)

    while True:
        ret, frame = cap.read()
        cv.imshow("Main Frame", frame)
        print_labels(frame)

        if not ret:
            print("No frame available")
```

```
        break

    cs.create(frame)

    # actove options
    if blueLego:
        find_legos_by_color(dt, cs, frame,model, cp.BLUE)
    if orangeLego:
        find_legos_by_color(dt, cs, frame,model, cp.ORANGE)
    if findAll:
        locate_all_legos(dt,cs,frame,model)
    if find2x6Blue:
        locate_specfic_Legos(dt,cs,frame,model, cp.BLUE, 3)
    if find4x6Black:
        locate_specfic_Legos(dt,cs,frame,model, cp.BLACK, 5)


    key = cv.waitKey(5)
    if key == ord("q"):
        break
    elif key == ord("s"):
        cs.buildFigure(frame)
    elif key == ord('p'):
        blueLego = False
        findAll = False
        find2x6Blue = False
        find4x6Black = False
        orangeLego = False
    elif key == ord('1'):
        blueLego = True
    elif key == ord('2'):
        orangeLego = True
    elif key == ord('a'):
        findAll = True
    elif key == ord('b'):
        find2x6Blue = True
    elif key == ord('l'):
        find4x6Black = True
```

```
    cap.release()
    cv.destroyAllWindows()


# detects legos by color
def find_legos_by_color(dt, cs, frame, model, color):
    legos = []
    legos = dt.detect_by_color(frame, color)

    locate(legos, frame, model, dt, cs)

# detects all contours in the frame
def locate_all_legos(dt,cs,frame,model):
    legos = []
    legos = dt.detect_contours(frame)

    locate(legos, frame, model, dt, cs)

# detects specific contours in the frame
def locate_specfic_Legos(dt, cs, frame, model, color, number):
    global labels
    legos = []
    legos = dt.detect_by_color(frame, color)

    locate_specific(legos, frame, model, dt, cs, labels[number])

# locate the position and label of the specific lego
def locate_specific(legos, frame, model, dt, cs, label):
     for lego in legos:
        if lego is not None:
            x, y, w, h = cv.boundingRect(lego)
            min_size_threshold = 90

            # validating for None values and too small values
            if all(coord is not None for coord in (x, y, w, h)) and w >= min_si
                class_label, confidence = dt.identfiy_object(frame, model, labe
                if class_label == label:
                    text = class_label
                    lego_coords = cs.calculate(lego)
```

```
                    lego_x = None
                    lego_y = None

                    if lego_coords is not None:
                        lego_x, lego_y = lego_coords
                    if lego_x is not None and lego_y is not None:
                        text = f'{class_label} Lego Center: ({lego_x:.2f}, {leg
                    cv.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
                    cv.putText(frame, text, (x, y - 10), cv.FONT_HERSHEY_SIMPLE
                    cv.imshow("Main Frame", frame)

# used to get the positon | universal function for other functions
def locate(legos, frame, model, dt, cs):
    for lego in legos:
        if lego is not None:
            x, y, w, h = cv.boundingRect(lego)
            min_size_threshold = 90

            if all(coord is not None for coord in (x, y, w, h)) and w >= min_si
                class_label, confidence = dt.identfiy_object(frame, model, labe
                text = class_label
                lego_coords = cs.calculate(lego)
                lego_x = None
                lego_y = None

                if lego_coords is not None:
                    lego_x, lego_y = lego_coords
                if lego_x is not None and lego_y is not None:
                    text = f'{class_label} Lego Center: ({lego_x:.2f}, {lego_y
                cv.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
                cv.putText(frame, text, (x, y - 10), cv.FONT_HERSHEY_SIMPLEX, 0
                cv.imshow("Main Frame", frame)


if __name__ == '__main__':
    run()
```

Anhang 2 detector.py

```
    import cv2 as cv
import numpy as np

class Detector(object):

    # detect all contours of the frame
    def detect_contours(self, frame):
        objects = []

        gray_frame = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
        blur_frame = cv.GaussianBlur(gray_frame, (9,9), 0)
        edges = cv.Canny(blur_frame,100, 200)

        # validating for better results
        kernel = np.ones((5, 5), np.uint8)
        dilated_edges = cv.dilate(edges, kernel, iterations=1)
        eroded_edges = cv.erode(dilated_edges, kernel, iterations=1)

        contours, hierarchy = cv.findContours(image=eroded_edges, mode=cv.RETR_
        #cv.drawContours(frame, contours, contourIdx=-1, color=(0, 255, 0), thi

        # append all contours to an object array
        for contour in contours:
            min_contour_area = 40
            if cv.contourArea(contour) > min_contour_area:
                x = y = w = h = None
                x, y, w, h = cv.boundingRect(contour)
                if all(coord is not None for coord in (x, y, w, h)):
                    objects.append(contour)

        if objects != []:
            return objects

    # detects by specific color
    def detect_by_color(self, frame, color):
        objects = []
```

```
        # process image to get the contours
        hsv_image = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
        mask = cv.inRange(hsv_image, color['lower'], color['upper'])
        color_contours, _ = cv.findContours(mask, cv.RETR_EXTERNAL, cv.CHAIN_AP

        # validate the contours
        for contour in color_contours:
            min_contour_area = 40
            if cv.contourArea(contour) > min_contour_area:
                x = y = w = h = None
                x, y, w, h = cv.boundingRect(contour)
                if all(coord is not None for coord in (x, y, w, h)):
                    objects.append(contour)

        if objects != []:
            return objects

    # using the model to identify the object
    def identfiy_object(self, frame, model, labels, object):
        roi = frame
        confidence_threshold = 0.7

        # cutting the lego out of the full image to identify it
        x, y, w, h = cv.boundingRect(object)
        roi = frame[y:y+h, x:x+w]

        if roi is not None and roi.any():
            # process it so it can be used in the model.predic
            input_image = cv.resize(roi, (224, 224))
            input_image = input_image / 255.0

            # identify the object
            predictions = model.predict(np.expand_dims(input_image, axis=0))[0]
            class_label = None
            confidence = None

            for i, confidence in enumerate(predictions):
                if confidence > confidence_threshold:
                    class_label = labels[i]
```

```
                    text = f"{class_label}: {confidence:.2f}"

        if class_label != None and confidence != None:
            return (class_label, confidence)
        return ('','')
```

Anhang 3 coordinatesystem.py

```
    import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

class CoordinateSystem(object):

    yellow_circle = None
    yellow_center = None

    red_circle = None
    red_center = None

    x_length = None
    y_length = None

    # finding the circles to create an coordinate system
    def create(self, frame):
        gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
        gray_blurred = cv.GaussianBlur(gray, (9, 9), 2)

        # identify the circles
        circles = cv.HoughCircles(
            gray_blurred,
            cv.HOUGH_GRADIENT,
            dp=1,
            minDist=50,
            param1=50,
            param2=30,
            minRadius=10,
```

```
            maxRadius=100
        )

        red_lower = np.array([160, 100, 100])
        red_upper = np.array([180, 255, 255])

        yellow_lower = np.array([20, 100, 100])
        yellow_upper = np.array([30, 255, 255])

        if circles is not None:

            circles = np.round(circles[0, :]).astype("int")

            for (x, y, r) in circles:

                roi = frame[y - r:y + r, x - r:x + r]
                if roi is not None:
                    hsv_image = cv.cvtColor(roi, cv.COLOR_BGR2HSV)

                    yellow_mask = cv.inRange(hsv_image, yellow_lower, yellow_up
                    red_mask = cv.inRange(hsv_image, red_lower, red_upper)

                    # if the circle is red than its the 1/1 and when yellow the
                    if np.any(yellow_mask > 0):
                        self.yellow_circle = (x,y,r)
                    if np.any(red_mask > 0):
                        self.red_circle = (x,y,r)

                if self.yellow_circle is not None and self.red_circle is not No
                    self.yellow_center = self.yellow_circle[:2]
                    self.red_center = self.red_circle[:2]
                    self.x_length = self.red_center[0] - self.yellow_center[0]
                    self.y_length = self.yellow_center[1] - self.red_center[1]

    # using the coordinates from the circles and the lego to calculate the posi
    def calculate(self, countour):

        x, y, w, h = cv.boundingRect(countour)
        if all(coord is not None for coord in (x, y, w, h,self.yellow_center[0]
```

```
            lego_contour = countour

            # finding the center of the lego
            M = cv.moments(lego_contour)
            if M["m00"] != 0:
                lego_center_x = int(M["m10"] / M["m00"])
                lego_center_y = int(M["m01"] / M["m00"])
                lego_center = (lego_center_x, lego_center_y)

                # convert the frame coordinates to the new coordinate system be
                normalized_lego_x = (lego_center_x - self.yellow_center[0]) / s
                normalized_lego_y = (self.yellow_center[1] - lego_center_y) / s

                normalized_lego_x = max(0, min(normalized_lego_x, 1))
                normalized_lego_y = max(0, min(normalized_lego_y, 1))

                if normalized_lego_x != 0 and normalized_lego_y !=0:
                    return (normalized_lego_x, normalized_lego_y)

    # Building a figure to check the values of the calculated position
    def buildFigure(self, frame):
        fig, ax = plt.subplots()

        ax.imshow(cv.cvtColor(frame, cv.COLOR_BGR2RGB))
        ax.scatter(*self.yellow_center, color='yellow', label='Yellow Circle')
        ax.scatter(*self.red_center, color='red', label='Red Circle')

        ax.plot([self.red_center[0], self.yellow_center[0]], [self.yellow_cente
                color='blue', linestyle='--', label='X Line')
        ax.plot([self.yellow_center[0], self.yellow_center[0]], [self.yellow_ce
                color='green', linestyle='--', label='Y Line')

        ax.set_aspect('equal', adjustable='box')

        x_ticks = np.linspace(self.yellow_center[0], self.red_center[0], 5)
        y_ticks = np.linspace(self.yellow_center[1], self.red_center[1], 5)

        ax.set_xticks(x_ticks)
        ax.set_yticks(y_ticks)
```

```
ax.set_xticklabels([f'{(val - self.yellow_center[0]) / (self.red_center
ax.set_yticklabels([f'{(val - self.yellow_center[1]) / (self.red_center

ax.set_xlabel('X Coordinate')
ax.set_ylabel('Y Coordinate')

ax.legend()

plt.grid(True)
plt.show()
```

Anhang 4 Tools und Hilfsprogramme

tensorflow==2.13.1: https://www.tensorflow.org/ Ein Machine learning/deep learning Tool zu erkennung der lego Steine

Teachable machine: https://teachablemachine.withgoogle.com/train Zum trainieren des Keras Model

ChatGPT: https://chat.openai.com/ Hilfsprogramm für Fragestellungen und Problemen

Python 3.8.3
keras==2.13.1
matplotlib==3.7.4
numpy==1.24.3
opencv-contrib-python==4.8.1.78
urllib3==2.1.0
vscode 1.81.0

Logitunes
Für die Einstellung der Kamera

Zotero
Für die Wissenschaftlichen Quellen

Overleaf: https://de.overleaf.com/
Online Latex editor

Quellenverzeichnis

Monographien

Hope, Tom, Yehezkel S. Resheff und Itay Lieder (9. Aug. 2017). *Learning TensorFlow: A Guide to Building Deep Learning Systems*. Google-Books-ID: sUowDwAAQBAJ. Ö'Reilly Media, Inc." 242 S. ISBN: 978-1-4919-7848-1.

Internetquellen

Keras (2023). *Keras: The high-level API for TensorFlow | TensorFlow Core*. TensorFlow. URL: `https://www.tensorflow.org/guide/keras` (besucht am 27. Nov. 2023).

Teachable Machine 2.0 makes AI easier for everyone (7. Nov. 2019). Google. URL: `https://blog.google/technology/ai/teachable-machine/` (besucht am 27. Nov. 2023).

Ehrenwörtliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Studienarbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mettmann, 27. November 2023

Emre Il