Devlog - Computer Vision mit openCV.js

Mattis Thieme

November 2024

Inhaltsverzeichnis

1 Einleitung und Motivation

(Einleitung und Motivation)

In diesem Devlog möchte ich einen Überblick über die geläufigen Methoden der Kreiserkennung und Kreisklassifizierung aufzeigen, von einfachen und leicht verständlichen Verfahren wie der Geometrie- oder Farberkennung, bis hin zu den mächtigen Werkzeugen der Musteranalyse und Objekterkennung.

Alle Ansätze werde ich dabei stets an einem Beispiel demonstrieren. Damit Du leicht alles auf deinem eigenen Rechner selber ausprobieren kannst, testen wir alle Methoden an Hand von Euromünzen. Warum ausgerechnet Euromünzen? Sie eignen sich perfekt: sie sind alle rund, haben klar festgelegte Farben und Größen und fast jeder sollte welche zur Hand haben. Im Laufe des Devlogs werden wir somit eine Webanwendung entwickeln, welche auf eine angeschlossene Webkamera zugreift, die Bilddaten nach Münzen durchsucht, und jede gefundene Münze ihren 'Wert' zuteilt. Am Ende wird dann anschließend der Gesamtbetrag aller Münzen im Bild angezeigt.

2 Wahl der Technologien

Wenn Probleme im Bereich der Bildverarbeitung gelöst werden sollen, fällt die Wahl häufig auf OpenCV. Und das nicht ohne Grund: OpenCV bietet ein rießiges Spektrum an Funktionen und Algorithmen, von einfachen Bildoperationen hin zu ausgereiften Algorithmen der Gesichtserkennung, Bildsegmentierung und Objekterkennung. Auch Maschinelles Lernen und Deep Learning sind in OpenCV integriert.

Die Wahl der Bibliothek wäre somit schnell getroffen, wenn wir nicht noch ein weiteres Kriterium hätten: die Webanwendung. Da OpenCV jedoch ursprünglich in C++ geschrieben ist, ist das primäre Interface, mit welchem auf die Funktionaltiätenzugegriffen wird, auch in C++ verfasst. Es gibt zwar mit Java und Python auch noch weitere alternative Schnittstellen, jedoch soll unsere Webanwendung, wie bereits oben erwähnt, nicht auf einem Server laufen, sondern direkt im Browser des Clients. Die Lösung: openCV.js.

Als relativ neuer Bestandteil des openCV-Projektes, ist openCV.js eine JavaScript-Portierung der OpenCV-Bibliothek. Sie ermöglicht es, OpenCV-Funktionen direkt im Browser auszuführen, ohne dass der Nutzer eine zusätzliche Software installieren muss. Somit können wir die volle Bandbreite der OpenCV-Funktionen nutzen, ohne auf die Vorteile einer Webanwendung verzichten zu müssen.

3 Integration in Webanwendung

3.1 openCV.js

Als aller erstes brauchen wir natürlich eine aktuelle Version von openCV.js. Diese können wir direkt von der offiziellen openCV-Webseite herunterladen:

https://docs.opencv.org/4.10.0/opencv.js

Speichere die Datei in deinem Projektverzeichnis und binde sie in deiner HTML-Datei wie eine normale JavaScript-Datei ein:

```
1 <script src="opencv.js"></script>
```

3.2 Grundaufbau der Webseite

Für unsere Webanwendung benötigen wir zunächst eine simple HTML-Struktur mit mindestens zwei Elementen: einem Video-Element für den Kamerastream und einem Canvas-Element, auf welchem wir die Bildverarbeitungsergebnisse anzeigen können. Beide Elemente sollten idealerweise übereinander liegen und die selbe Größe haben.

Die html-Datei könnte also wie folgt aussehen:

```
    <!DOCTYPE html>
<html lang="de">
<script src="lib/opencv_4.10.0.js"></script>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Video Player mit Canvas Overlay</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
<div id="mainContainer">
    <div class="container" id="textContainer">
        <h1>CoinFinder</h1>
        <p>Aktueller Wert: <span id="value">0</span>€</p>
    </div>
    <div class="container" id="videoContainer">
        <video id="video" width="720" height="540" autoplay muted loop></video>
        <canvas id="canvas" width="720" height="540"></canvas>
    </div>
</div>

<script src="script.js"></script>
</body>
</html>
```

Und die dazugehörige CSS-Datei:

```
    *{
    box-sizing: border-box;
}

html, body {
    background-color: #18204d;
    color: #ffffff;
}

#mainContainer{
    position: relative;
    width: 97vw;
    height: 97dvh;
    display: flex;
    flex-direction: column;
}

#videoContainer{
    position: relative;
    height: 100%;
    width: 100%;

    flex-grow: 1;

    align-items: center;
    justify-content: center;
    display: flex;
```

```
28
29  }
30
31  #video, #canvas {
32      height: 100%;
33      width: 100%;
34      left: 0;
35      top: 0;
36      position: absolute;
37      aspect-ratio: inherit;
38      object-fit: contain;
39  }
40
41  #mainContainer{
42      border: 0.5em solid #ffa300;
43  }
44
45  #videoContainer{
46      border: 0.5em solid blue;
47  }
```

3.3 Grundlegende openCV Methoden

Codebeispiel:

```
1  if(!result.wasPinDetected || (coreStartPoint === undefined || coreEndPoint ===
       undefined)){
2      if(debug){
3          console.log("Kern nicht gefunden");
4      }
5      amountOfUnknown++;
6  } else { //Wenn ein Kern gefunden wurde:
7      //Kern zeichnen und Eckpunkte ermitteln
8      result = DrawCore(guiMat, coreStartPoint, coreEndPoint, 1);
9
10     //Eckpunkte des Rechteckes speichern
11     let leftPoint = result.leftPoint;
12     let rightPoint = result.rightPoint;
13     let topPoint = result.topPoint;
14     let bottomPoint = result.bottomPoint;
15
16     //den Bildbereich definieren, in dem sich der Zylinderkern befindet
17     let pinScanArea = new cv.Rect(leftPoint.x, topPoint.y, Math.abs(rightPoint.x -
           leftPoint.x), Math.abs(bottomPoint.y - topPoint.y));
18     //scanArea in der Ausgabe-Matrix einzeichnen
19     if (debug) {
20         cv.rectangle(guiMat, new cv.Point(pinScanArea.x, pinScanArea.y), new cv.
               Point(pinScanArea.x + pinScanArea.width, pinScanArea.y + pinScanArea.
               height), new cv.Scalar(255, 255, 255, 255), 5);
21     }
22
23     result = currentCore.CheckPins(inputMat, guiMat, pinScanArea); //Stifte prüfen
24 }
```

4 Kreiserkennung

4.1 Circle Hough Transform

4.2 Parameter und Einstellungen

5 Kreis-Klassifizierung

5.1 Geometrie

5.2 Farbanalyse

5.2.1 Durchschnitt und Helligkeit

5.2.2 Histogrammvergleich

5.2.3 Farbverlauf-Analyse

5.3 Musteranalyse

5.4 Objekterkennung

5.5 Machinelles Lernen

6 Limitationen

7 Fazit

Die Möglichkeit von openCV.js, OpenCV-Funktionen direkt im Browser des Clients auszuführen, eröffnet eine Vielzahl von neuen Anwendungsmöglichkeiten. Rechenintensive Bildverarbeitungsalgorithmen müssen nun nicht mehr auf einem Server laufen, wodurch Kapazitäten frei werden und für andere Aufgaben genutzt werden können. Immer mehr Onlinebesuche finden auf mobilen Geräten statt - die Verwendung der eingebauten Kamera für Bildverarbeitungsaufgaben könnte somit in Zukunft eine wichtige Rolle spielen. (Beispiele einfügen?)