

Devlog - Computer Vision mit openCV.js

Mattis Thieme

November 2024

Inhaltsverzeichnis

1 Einleitung	4
1.1 Motivation	4
1.2 Zielsetzung	5
2 Erstellen der Webseite	6
2.1 openCV.js	6
2.2 Grundaufbau der Webseite	6
2.3 Grundlegende openCV Methoden	8
3 Eingriffe zur Laufzeit	11
3.1 Die Bedeutung von einfachem Testing	11
3.2 Variablen-Slider	11
3.3 Cookies	13
4 Kreiserkennung	16
4.1 Funktionsweise der HoughCircles-Methode	16
5 Template Matching	19
5.1 Überlegungen	19
5.2 Die Vorlagen	19
5.3 Funktionsweise	21
5.4 Exkurs: Mehrere Matrizen in einem Canvas anzeigen	23
5.5 Erste Ergebnisse	25
6 Anpassungen	27
6.1 Kantenerkennung	27
6.1.1 Funktionsweise der Canny-Methode	27
6.1.2 Die korrekten Parameter finden	27
6.2 Neues Problem: Mat-Types	29
6.3 Matrizen rotieren und transformieren	29
6.4 Asynchronität und Ladebalken	30
7 Ergebnisse	32
7.1 Neue MatchTemplate-Funktion	32

7.2 Finale Ergebnisse	35
8 Fazit	37
8.1 Die größten Probleme	37
8.2 Ausblick	37

1 Einleitung

Herzlich Willkommen!

Probleme der Bilderkennung und Bildverarbeitung gewinnen immer mehr an Relevanz. Kein Wunder: Mit der stetigen Verbreitung von mobilen Endgeräten haben immer mehr Systeme Zugriff auf eine (mittlerweile sehr hochwertige) Kamera. Die Möglichkeiten, die sich dadurch ergeben, sind vielfältig: über das Scannen von Dokumenten, hin zur Klassifizierung von Tieren und Pflanzen bis hin zum industriellen Einsatz in der Qualitätskontrolle.

Zusätzlich geht der Trend auch in die Richtung von Onlineservices und Webanwendungen. Programme die früher noch installiert werden mussten, laufen nun direkt im Browser des Clients - entweder als online gehostete Webseite oder lokal in Form einer Cross-Plattform-Desktop-Anwendung. Hauptgrund ist natürlich die Plattformunabhängigkeit und die damit verbundene Reduktion von Entwicklungskosten und Zeit.

Aber wie sieht es mit Bildverarbeitung und Bilderkennung im Webkontext aus? Wenn Probleme im Bereich der Bildverarbeitung gelöst werden sollen, fällt die Wahl häufig auf OpenCV. Und das nicht ohne Grund: OpenCV bietet ein riesiges Spektrum an Funktionen und Algorithmen, von einfachen Bildoperationen hin zu ausgereiften Algorithmen der Gesichtserkennung, Bildsegmentierung und Objekterkennung. Auch Maschinelles Lernen und Deep Learning sind im Funktionsumfang von OpenCV enthalten.

Jedoch wurde OpenCV nun einmal in C++ geschrieben und kann somit nicht direkt auf Clientseite implementiert werden. Aber nicht immer ist die Option sinnvoll, Aufgaben der Bildverarbeitung auf der Seite des Servers lösen zu lassen. Nicht immer steht die benötigte Rechenleistung zur Verfügung, auch die Latenz kann letztendlich je nach Anwendungsfalls ein Problem darstellen. Somit stellt sich einem die folgende Frage: Ist es möglich OpenCV auch im Browser des Clients zu lösen? Die Antwort: Ja! Mit openCV.js.

1.1 Motivation

Aber warum der Blog? Einst musste ich (wie du vielleicht auch) ein Problem der Bildverarbeitung lösen, welches zwingend im Webkontext stattfinden sollte. Die Anforderungen waren klar: die Bildverarbeitung sollte direkt im Browser des Clients stattfinden, ohne dass der Nutzer eine zusätzliche Software installieren muss.

Die Bibliothek openCV war natürlich die erste Wahl, jedoch stand ich nun genau vor diesem Problem: wie bekomme ich openCV, eine Bibliothek, die ursprünglich in C++ geschrieben ist, in meiner Webanwendung zum Laufen? Meine Lösung war natürlich 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.

Obwohl es sich zunächst als die beste Lösung angehört hat, war die Nutzung von openCV.js jedoch kein Selbstläufer. Wie du wahrscheinlich bereits mit Schrecken festgestellt hast, ist die offizielle Dokumentation nur für die C++ Schnittstelle geschrieben, und Tutorials für openCV.js sind leider rar gesät. Die JavaScript-Schnittstelle von openCV ist nun mal die neuste Änderung des mittlerweile 20 Jahre alten Projektes und somit noch nicht so ausgereift und erprobt wie die anderen Schnittstellen.

Genau aus diesen Gründen habe ich mich für das Schreiben dieses Devblogs entschieden. Ich möchte

meine Erfahrungen und Erkenntnisse teilen, um anderen Entwicklern zu helfen, die vor dem gleichen Problem stehen. Ich möchte zeigen, dass es gar nicht so kompliziert ist, openCV.js in einer Webanwendung zu nutzen und wie mächtig die Bibliothek tatsächlich ist. Ja es gibt ein paar Eigenheiten und Macken, aber genau diese werde ich erläutern, damit du nicht die gleichen Fehler machst wie sie ich einst gemacht habe.

1.2 Zielsetzung

Kommen wir nun zu den eigentlichen Fragen: Was möchte ich erreichen? Was möchte ich entwickeln? Wie bereits oben erwähnt, möchte ich eine Webanwendung erstellen, welche durch Nutzung von openCV.js Probleme der Bildverarbeitung und Bilderkennung lösen kann. Das konkrete Problem sollte dabei sowohl häufig genutzte Methoden der Bildverarbeitung abdecken, als auch leicht verständlich und reproduzierbar sein. Ich möchte, dass du die Anwendung einfach nachbauen und selber ausprobieren kannst, um die Funktionsweise von openCV.js direkt durch eigene Erfahrungen zu erlernen.

Nach einigen Überlegungen bin ich auf die Idee gekommen, eine Webanwendung zu entwickeln, welche Münzen erkennen und deren Wert bestimmen kann. Warum ausgerechnet Münzen? Zum einen ist es ein angemessen komplexes Problem, welches gleich mehrere unterschiedliche Aspekte der Bildverarbeitung abdeckt. Zum anderen ist es eine Problemstellung, welche sich leicht auf einem anderen System reproduzieren lässt. Alles was du brauchst, sind lediglich eine Webkamera und ein paar Münzen.

Interessiert? Dann lass uns anfangen! Im meinem nächsten Beitrag werde ich direkt die grundlegende Webseitenstruktur und die Einbindung von openCV.js erläutern.

Bis dahin!

2 Erstellen der Webseite

Bevor wir mit der Implementierung irgendwelcher Bildverarbeitungsalgorithmen beginnen können, müssen wir zunächst eine Webseite mit den notwendigen Funktionalitäten erstellen.

2.1 openCV.js

Als aller erstes braucht es natürlich eine aktuelle Version von openCV.js. Diese kann direkt von der offiziellen openCV-Webseite heruntergeladen werden:

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

Es handelt sich hierbei um eine mittels Emscripten kompilierte Version von OpenCV, welche in JavaScript in Form einer WebAssembly ausgeführt werden kann. Das Einbinden erfolgt wie mit einer gewöhnlichen JavaScript-Datei. Da der Programmcode in Bytecode zur Verfügung steht, erfolgt die Ausführung deutlich schneller da die Kompilierung zur Laufzeit entfällt. Dies führt jedoch auch zu dem Problem, dass IDE-Funktionalitäten wie Auto vervollständigung oder Syntax-Highlighting leider nicht verfügbar sind.

2.2 Grundaufbau der Webseite

Nun stellt sich die Frage: Welche Funktionen soll unsere Webseite haben? Zum einen brauchen wir natürlich Zugriff auf die Kamera des Nutzers, um den Videostream zu erhalten. Zum anderen benötigen wir eine Möglichkeit, das Ergebnis der Bildverarbeitung anzuzeigen.

Mithilfe eines Video-Elementes können wir den Kamerastream von Webkameras anzeigen und auf diesen zugreifen. Für die Anzeige der Bildverarbeitungsergebnisse habe ich mich für ein Canvas-Element entschieden. Warum? - In OpenCV gibt es viele Möglichkeiten, Zeichenvorgänge auf Matrizen anzuwenden. Diese Matrizen können dann auf ein Canvas-Element gezeichnet werden (mehr dazu weiter unten). Mein Plan ist somit alle relevanten Ergebnisse direkt auf den zu verarbeitenden Frame zu zeichnen und diesen dann auf dem Canvas-Element anzuzeigen.

Nach einigen Tests habe ich mich für folgende Struktur entschieden:

```
1 <!DOCTYPE html>
2 <html lang="de">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Video Player mit Canvas Overlay</title>
7   <link rel="stylesheet" href="style.css">
8   <script src="lib/opencv_4.10.0.js"></script>
9 </head>
10 <body>
11 <div id="mainContainer">
12   <div id="textContainer">
13     <h1>CoinFinder</h1>
14     <h3>Aktueller Wert: <span id="value">0</span></h3>
15   </div>
16   <div class="container" id="videoContainer">
17     <video id="video" width="720" height="540" autoplay muted loop></video>
18     <canvas id="outputCanvas" width="720" height="540"></canvas>
19   </div>
20 </div>
21 </body>
22 </html>
```

Damit haben wir ein Video-Element, welches den Kamerastream anzeigt und ein Canvas-Element, um die Bildverarbeitungsergebnisse anzuzeigen. In der nachfolgenden CSS-Datei sorge ich dafür, dass beide Elemente stets übereinander liegen und die gleiche Größe haben. So sieht der Nutzer immer das aktuelle Kamerabild und die Bildverarbeitungsergebnisse direkt darüber. Zusätzlich bekommt der Canvas die Eigenschaft `image-rendering: pixelated;`, damit auch kleine Matrizen originalgetreu dargestellt werden.

Die CSS-Datei sieht wie folgt aus:

```
1  *{
2      box-sizing: border-box;
3  }
4
5  html, body {
6      background-color: #18204d;
7      color: #ffffff;
8      font-family: Arial, Helvetica, sans-serif;
9  }
10
11 h3{
12     margin: 0 0 0 1em;
13 }
14
15 #mainContainer{
16     position: relative;
17     width: 97vw;
18     height: 97dvh;
19     display: flex;
20     flex-direction: column;
21 }
22
23 #textContainer{
24     display: flex;
25     align-items: center;
26 }
27
28 #textContainer > *{
29     margin: 0 1em 0 1em;
30 }
31
32 button{
33     background-color: #ffa300;
34     color: #18204d;
35     border: none;
36     padding: 0.5em 1em;
37     font-size: 1em;
38     cursor: pointer;
39 }
40
41 #videoContainer{
42     position: relative;
43     height: 100%;
44     width: 100%;
45     flex-grow: 1;
46     align-items: center;
47     justify-content: center;
48     display: flex;
49 }
50
51
52 #video, #outputCanvas {
53     height: 100%;
54     width: 100%;
```

```

55     left: 0;
56     top: 0;
57     position: absolute;
58     aspect-ratio: inherit;
59     object-fit: contain;
60   }
61
62 #outputCanvas{
63   image-rendering: pixelated;
64 }
65
66 #mainContainer{
67   border: 0.5em solid #ffa300;
68 }
69
70 #videoContainer{
71   border: 0.5em solid blue;
72 }
```

Als nächstes braucht es nun grundlegende Methoden, um auf die Kamera des Nutzers zuzugreifen und den Videostream auf dem Video-Element anzulegen.

2.3 Grundlegende openCV Methoden

Um auf die Kamera des Nutzers zuzugreifen, kann man unter anderem ein openCV-VideoCapture-Objekt verwenden. Dieses Objekt kann entweder direkt auf ein Video-Element zugreifen oder auf eine Datei. Da wir in unserem Fall den Kamerastream verwenden wollen, greifen wir direkt auf das Video-Element zu. Für den Zugriff auf die Webkamera benötigen wir die getUserMedia()-Methode des MediaDevices-Interfaces. Um sicherzustellen, dass openCV vollständig geladen ist, warten wir zusätzlich auf das "load"-Event des Fensters und initialisieren erst dann die Kamera:

```

1 window.addEventListener("load", function () {
2   video = document.getElementById('video');
3   outputCanvas = document.getElementById('outputCanvas');
4   videoContainer = document.getElementById('videoContainer');
5
6   // get the camera
7   navigator.mediaDevices.getUserMedia({
8     video: true,
9     audio: false
10 }).then(stream => {
11   video.srcObject = stream;
12   video.onloadedmetadata = () => {
13     video.play()
14
15     //print camera stats
16     console.log("Camera resolution: " + video.videoWidth + "x" + video.
17                 videoHeight);
18     console.log("Camera frame rate: " + stream.getVideoTracks()[0].
19                 getSettings().frameRate+ " fps");
20
21     //initialize the inputMat-Matrix
22     inputMat = new cv.Mat(video.height, video.width, cv.CV_8UC4);
23     guiMat = new cv.Mat(video.height, video.width, cv.CV_8UC4);
24     videoCapture = new cv.VideoCapture(video);
25
26     //check if everything is loaded
27     cameraLoaded = true;
28     console.log("Camera loaded");
29     CheckIfLoadingFinished();
```

```

28         };
29     }).catch(error => {
30         console.error('Error accessing the camera: ', error);
31     });
32 };
33 }

```

Nachdem die Kamera initialisiert ist, kann die `read()`-Methode des `VideoCapture`-Objekts verwendet werden, um den aktuellen Frame des Videostreams in eine openCV-Matrix zu konvertieren:

```

1 let inputMat;
2 let guiMat;
3
4 videoCapture.read(inputMat);

```

Für die nachfolgenden Funktionen werde ich immer wieder die Variablen `inputMat` und `guiMat` verwenden. `inputMat` enthält den aktuellen Frame des Videostreams, während `guiMat` eine Kopie des Frames ist, auf welchem die Bildverarbeitungsergebnisse gezeichnet werden.

Nun brauchen wir jedoch noch eine Methode, um eine openCV-Matrix (in meinem Fall `guiMat`) auf einem Canvas ausgeben zu können. Dafür können wir einfach die `cv.imshow()`-Methode verwenden, welche eine Matrix auf ein Canvas-Element zeichnet. In unserem Fall verwenden wir das `outputCanvas`-Element, welches über dem Video-Element liegt. Die Funktion zum Anzeigen einer openCV-Matrix sieht wie folgt aus:

```

1 function ShowMatrix(src, canvas){
2     cv.imshow(canvas, src);
3 }

```

Hinweis zur Speicherfreigabe: openCV.js verwaltet den Speicher nicht automatisch, wie es bei JavaScript üblich ist. Das bedeutet, dass wir selbst dafür verantwortlich sind, den Speicher freizugeben, sobald wir ihn nicht mehr benötigen. Dies betrifft hauptsächlich Objekte vom Typ `cv.Mat` und `cv.MatVector`, welche wir mit der `delete()`-Methode freigeben können. Tun wir dies nicht, verbraucht die Anwendung mit jedem neuen Aufruf von `new cv.Mat()` oder `cv.imread()` mehr Speicher, bis irgendwann der Browser abstürzt. Sollte dein Programm nach einigen Sekunden oder Minuten aufhören zu funktionieren, könnte dies ein Hinweis auf ein Speicherleck sein. Schaue in diesem Fall in die Konsole nach einer entsprechenden Fehlermeldung.

Zu guter Letzt benötigen wir noch eine Hauptschleife, bei der die Bilddaten aus dem Video-Element extrahiert, die zukünftigen Bildverarbeitungsalgorithmen ausgeführt und das Ergebnis auf dem Canvas-Element angezeigt werden. Dafür können wir entweder die `requestAnimationFrame()`-Methode verwenden, welche die Ausführung der Methode automatisch der Bildschirmwiederholrate anpasst, oder wir benutzen `setTimeout()` um selber die Zeit zwischen der wiederholten Ausführung festzulegen. Ich habe zusätzlich einen Bool "loopActive" hinzugefügt, um den Loop bei Bedarf auch nur einmalig ausführen zu können. Die Funktion sieht schließlich so aus:

```

1 let waitingForAnimationFrame = false;
2 function mainLoop() {
3     if(!loadingFinished){
4         console.warn("Can't start the loop because something is not loaded yet");
5         return;
6     }
7
8     waitingForAnimationFrame = false;
9
10    console.log("--- loop started");
11
12    videoCapture.read(inputMat);
13    videoCapture.read(guiMat);

```

```
14
15 //do something with the matrix
16
17 ShowMatrix(guiMat, outputCanvas);
18
19 if(loopActive){
20     waitingForAnimationFrame = true;
21     requestAnimationFrame(mainLoop);
22 }
23
24 console.log("--- loop ended");
25 }
```

Nun haben wir das Grundgerüst für unsere Webseite erstellt. Als nächstes können wir uns an die Implementierung der ersten Bildverarbeitungsalgorithmen wie der Kreiserkennung machen. Mehr dazu gibt es im nächsten Eintrag. Bis bald!

3 Eingriffe zur Laufzeit

Halt! Einfach drauf los zu programmieren ist nicht immer die beste Idee. Aus Erfahrung weiß ich, dass insbesondere die Arbeit mit OpenCV in viel Trial and Error resultieren kann.

Bevor ich anfange die ersten Bildverarbeitungsalgorithmen zu implementieren, habe ich daher einige Vorkehrungen getroffen, um den Entwicklungsprozess zu beschleunigen und zu vereinfachen. In diesem Abschnitt möchte ich näher auf diese Vorkehrungen eingehen, und zudem verdeutlichen warum es bei der Arbeit mit OpenCV so wichtig ist, einfache Testing-Möglichkeiten zu haben.

3.1 Die Bedeutung von einfachem Testing

Meine nächste Aufgabe, die Implementierung der Kreiserkennung, erfordert eine Menge an Feintuning und Anpassungen. Jeder Parameter muss sorgfältig gewählt werden, um ein optimales Ergebnis zu erzielen. Zudem kann im Vorhinein nicht ermittelt werden, welche Parameter die besten sind. Es ist also ein iterativer Prozess, bei dem ich die Parameter anpasse, das Ergebnis betrachte und dann erneut die Parameter anpasse.

Nun stelle man sich vor, man müsste für jede kleine Änderung die Webseite neu laden, die Kamera neu ausrichten und die Münzen neu platzieren. Dies ist der Grund, weshalb ich Eingriffe zur Laufzeit implementieren möchte. Diese Eingriffe sollen es mir ermöglichen, die Parameter der Kreiserkennung und weiterer openCV Funktionen direkt zur Laufzeit zu verändern, ohne die Webseite neu laden zu müssen, und somit schnell die besten Parameter zu finden.

3.2 Variablen-Slider

Meine Idee ist es, einen HTML-Slider erstellen zu können, mit dem ich schnell und einfach den Wert einer beliebigen JS-Variable verändern kann. Diese Slider sollen sowohl den aktuellen Wert der Variable anzeigen als auch bei Interaktion des Benutzers ihre zugewiesene Variable verändern. Der Wert des HTML-Elementes soll somit direkt mit der Variable synchronisiert werden.

Für die Slider habe ich die Bibliothek [noUISlider](#) verwendet. Über die data-Attribute von den HTML-Elementen kann unkompliziert der Wertebereich und die Schrittweite des Sliders festgelegt werden.

Nun müsste aber standardmäßig ein Event-Listener für jeden Slider erstellt werden, der bei Veränderung den Wert einer Variable ändert. Da ich jedoch nicht für jede Variable einen eigenen Event-Listener erstellen möchte, habe ich mich für einen anderen Ansatz entschieden. Stattdessen soll man direkt im HTML-Element des Sliders angeben können, welche Variable durch diesen Slider verändert werden soll.

Hierfür benötigen wir zunächst ein eigenes data-Attribute, in welchem der Name der zu verändernden Variable angegeben wird. Zum Zweiten benötigen wir eine Funktion, die alle Slider-Elemente durchgeht und für jedes Element mithilfe des data-Attributs den Event-Listener für die angegebene Variable erstellt. Hierfür braucht es eine Möglichkeit den String-Namen der Variable in eine Referenz auf die Variable umzuwandeln. Dies kann mithilfe der eval()-Funktion erreicht werden.

Die Event-Listener Funktion sieht dann wie folgt aus:

```
1 function InitSliders(){
2     // Alle Slider-Container selektieren
3     const sliderContainers = document.querySelectorAll('.sliderContainer');
4
5     sliderContainers.forEach(container => {
6         const sliderElement = container.querySelector('.slider');
```

```

7      const valueElement = container.querySelector('.sliderValue');
8
9      const min = parseFloat(container.dataset.min);
10     const max = parseFloat(container.dataset.max);
11     const step = parseFloat(container.dataset.step);
12     const var1 = container.dataset.var1;
13     const var2 = container.dataset.var2;
14     const isRange = container.dataset.range === "true"; // Überprüft, ob
15         Bereichsmodus aktiv ist
16         //console.log("Data for slider: min: " + min + " max: " + max + " step: " +
17             step + " var1: " + var1 + " var2: " + var2 + " isRange: " + isRange);
18
19         // Startwerte auslesen
20         let startValues = [];
21         if (isRange) {
22             // Bereichsmodus: Startwerte aus den <span>-Elementen lesen
23             const spanValues = valueElement.querySelectorAll('span');
24             startValues = Array.from(spanValues).map(span => parseFloat(span.
25                textContent));
26             if (startValues.length !== 2) {
27                 // Fallback: Standardwerte in der Mitte des Bereichs
28                 startValues = [min + (max - min) / 3, max - (max - min) / 3];
29             }
30         } else {
31             // Einzelregler: Einzelwert auslesen
32             startValues = [parseFloat(valueElement.textContent) || (min + max) /
33                 2];
34         }
35
36         // Slider erstellen
37         noUiSlider.create(sliderElement, {
38             start: startValues,
39             range: {
40                 'min': min,
41                 'max': max
42             },
43             step: step,
44             connect: isRange ? true : [true, false] // Verbindet die Regler bei
45                 Range-Modus
46         });
47
48         SetSliderValueFromCookie(container.id);
49
50         // Update-Event
51         sliderElement.noUiSlider.on('update', (values, handle) => {
52             if (isRange) {
53                 // Bereichsmodus: Werte in <span>-Elementen aktualisieren
54                 if(step >= 1){
55                     valueElement.textContent = values.map(value => Math.round(value
56                         )).join(' - ');
57                 }else{
58                     valueElement.textContent = values.map(value => value).join(' -
59                         ');
60                 }
61
62                 //TODO: Beide Variablen aktualisieren
63                 if(var1 !== undefined && var2 !== undefined){
64                     UpdateVariable(var1, values, 0);
65                     UpdateVariable(var2, values, 1);
66                 }else{
67                     console.log("var1 or var2 is undefined. Slider will not change
68                         any variables");
69                 }
70             }
71         });
72
73     }
74
75     // Setzt die Werte in den Inputfeldern
76     function SetSliderValueFromCookie(id) {
77         const cookie = document.cookie;
78         const cookieArr = cookie.split(';');
79         const cookieObj = {};
80         for (let i = 0; i < cookieArr.length; i++) {
81             const cookiePair = cookieArr[i].trim();
82             const cookieKey = cookiePair.split('=')[0];
83             const cookieValue = cookiePair.split('=')[1];
84             cookieObj[cookieKey] = cookieValue;
85         }
86         const sliderElement = document.getElementById(id);
87         const valueElement = sliderElement.querySelector('.sliderValue');
88         const var1 = sliderElement.dataset.var1;
89         const var2 = sliderElement.dataset.var2;
90         const step = sliderElement.dataset.step;
91         const min = sliderElement.dataset.min;
92         const max = sliderElement.dataset.max;
93         const isRange = sliderElement.dataset.range === "true";
94
95         if (isRange) {
96             const startValues = cookieObj[var1] ? cookieObj[var1] : cookieObj[var2];
97             const startValuesArr = startValues.split(' - ');
98             const startValue1 = startValuesArr[0];
99             const startValue2 = startValuesArr[1];
100            const startValue1Obj = {
101                value: startValue1,
102                handle: 0
103            };
104            const startValue2Obj = {
105                value: startValue2,
106                handle: 1
107            };
108            const values = [startValue1Obj, startValue2Obj];
109            const handle = 1;
110            const stepObj = {
111                value: step,
112                handle: 1
113            };
114            const connectObj = {
115                value: true,
116                handle: 1
117            };
118            const rangeObj = {
119                min: min,
120                max: max
121            };
122            const sliderConfig = {
123                start: values,
124                range: rangeObj,
125                step: stepObj,
126                connect: connectObj
127            };
128            const slider = noUiSlider.create(sliderElement, sliderConfig);
129            slider.setOption('range', rangeObj);
130            slider.setOption('step', stepObj);
131            slider.setOption('connect', connectObj);
132            slider.setOption('values', values);
133            slider.setOption('handle', handle);
134            slider.setOption('start', values);
135        } else {
136            const startValueObj = {
137                value: cookieObj[var1],
138                handle: 0
139            };
140            const sliderConfig = {
141                start: startValueObj
142            };
143            const slider = noUiSlider.create(sliderElement, sliderConfig);
144            slider.setOption('start', startValueObj);
145        }
146    }
147
148    // Aktualisiert eine Variable
149    function UpdateVariable(variable, values, handle) {
150        const variableElement = document.getElementById(variable);
151        const valueElement = variableElement.querySelector('.sliderValue');
152        const value = values[handle];
153        const valueString = value.toString();
154        valueElement.textContent = valueString;
155        variableElement.dataset.value = valueString;
156    }
157
158    // Loggt eine Fehlermeldung
159    function logError(message) {
160        console.error(`Error: ${message}`);
161    }
162
```

```

63
64     } else {
65         // Einzelregler: Textinhalt aktualisieren
66         if(step >= 1){
67             valueElement.textContent = Math.round(values[handle]);
68         }else{
69             valueElement.textContent = values[handle];
70         }
71
72         if(var1 !== undefined){
73             // Variable aktualisieren
74             UpdateVariable(var1, values, handle);
75         }else{
76             console.log("var1 is undefined. Slider will not change any
77                         variables");
78         }
79     }
80
81     // Cookie setzen
82     SetSliderCookie(container.id, values);
83 });
84 );
85
86 console.log("Sliders initialized");
87 }
```

Die Funktion `UpdateVariable()` wandelt schließlich den String-Namen der Variable in eine Referenz um und ändert den Wert der Variable. Hierfür wird die `eval()`-Funktion verwendet, die den String als JavaScript-Code interpretiert.

```

1 function UpdateVariable(varName, values, handle){
2     //check if variable exists
3     if (eval('typeof ' + varName) === 'undefined') {
4         console.warn('Variable ' + varName + ' does not exist');
5         return;
6     }
7
8     //update variable
9     if(eval('typeof ' + varName) === 'number'){
10        eval(varName + ' = ' + parseFloat(values[handle]));
11    }else if(eval('typeof ' + varName) === 'string'){
12        eval(varName + ' = ' + values[handle]);
13    }
14 }
```

Wie du vielleicht gesehen hast, gibt es auch noch die Funktionen `SetSliderValueFromCookie()` und `SetSliderCookie()`. Diese Funktionen dienen dazu, die Werte der Slider in Cookies zu speichern und beim Laden der Webseite wiederherzustellen. So behalten die Slider ihre eingestellten Werte auch nach einem Neuladen der Webseite.

3.3 Cookies

Cookies sind kleine Textdateien, die auf dem Computer des Benutzers gespeichert werden. Wie du sicherlich bereits weißt, dienen sie unter anderem dem Zweck, Informationen Session-übergreifend zu speichern. In meinem Fall möchte ich die Werte der Slider speichern, um sie beim Neuladen der Webseite wiederherzustellen.

Jedoch ist das Speichern von Werten in Cookies nicht ganz trivial: Pro Cookie kann nur ein einziger

String mit einer maximalen Länge von 4096 Bytes gespeichert werden. Da die Werte der Slider jedoch Arrays von Zahlen sind, müssen diese erst in einen String umgewandelt werden. Hierfür habe ich mich für die `JSON.stringify()`-Funktion entschieden, die ein JavaScript-Objekt in einen JSON-String umwandelt. Die Funktion zum schreiben des Cookies sieht dann wie folgt aus:

```

1 function SetSliderCookie(sliderId, value) {
2     // Holen des bestehenden Cookie-Werts
3     let sliderValues = GetSliderCookie();
4
5     // Setze den Wert des Sliders im Cookie
6     sliderValues[sliderId] = value;
7
8     // Setze den Cookie mit den neuen Werten
9     const expires = new Date();
10    expires.setDate(expires.getDate() + 7); // Cookie läuft in 7 Tagen ab
11    document.cookie = `sliderValues=${JSON.stringify(sliderValues)}; expires=${expires.toUTCString()}; path=/`;
12 }

```

Zunächst erstelle ich ein Objekt `sliderValues` aus dem Cookie, falls es bereits existiert. Anschließend füge ich den neuen Wert des Sliders hinzu und speichere das Objekt als JSON-String im Cookie. Hierfür muss zudem ein Ablaufdatum für den Cookie festgelegt werden, in meinem Fall läuft der Cookie nach 7 Tagen ab. Der Cookie sieht dann wie folgt aus:

```

1 {"slider1":["5100.00"], "slider2":["3900.00"], "slider3":["7.00"], "slider4":["3.00"],
  "sliderValue3":["230.00"], "sliderValue4":["129.00"]}

```

Um die Daten aus dem Cookie wiederherzustellen, muss in der Funktion `GetSliderCookie()` der Cookie ausgelesen und in ein JavaScript-Objekt umgewandelt werden. Eine saubere Lösung, welche selbst beim Vorhandensein von weiteren Cookies funktioniert, könnte wie folgt aussehen:

```

1 function GetSliderCookie(){
2     const value = `; ${document.cookie}`;
3     const parts = value.split(`; sliderValues=`);
4     if (parts.length === 2) {
5         const cookieValue = parts.pop().split(',').shift();
6         return JSON.parse(cookieValue); // Parsen des JSON-Strings
7     }
8     return {};// Falls der Cookie nicht existiert, ein leeres Objekt zurückgeben
9 }

```

Lass mich diese Funktion kurz erklären: Zunächst wird der Cookie-Wert aus dem `document.cookie`-String extrahiert. Die „cookie“ Eigenschaft des Objekts „`document`“ enthält alle Cookies, die auf der aktuellen Seite gesetzt wurden. Dem String wird ein Semikolon und ein Leerzeichen vorangestellt, um sicherzustellen, dass der Cookie-Wert korrekt extrahiert wird, selbst wenn noch andere Cookies vorhanden sind. Nun wird über `value.split('; sliderValues=')` der String zweigeteilt. Der erste Teil, welcher alles vor dem Cookie-Wert enthält, wird per `pop()` verworfen. Der zweite Teil, welcher den Cookie-Wert inklusive möglicher weiterer Cookies enthält, muss noch weiter verarbeitet werden. Hierfür wird der String per `split(',')` wieder in ein Array geteilt und mit `shift()` das erste Element, welches den Cookie-Wert enthält, extrahiert. Nun erst können wir sicher sein, dass mögliche weitere Cookies vor und nach unserem Cookie-Wert entfernt wurden. Der Slider-Cookie-Wert wird schließlich per `JSON.parse()` in ein JavaScript-Objekt umgewandelt und zurückgegeben.

Zu guter Letzt braucht es noch eine Funktion, welche beim Start der Webseite die Werte der Slider aus dem Cookie wiederherstellt. Diese Funktion sieht wie folgt aus:

```

1 function SetSliderValueFromCookie(containerID){
2     const sliderValues = GetSliderCookie();
3     const sliderContainer = document.getElementById(containerID);
4

```

```

5   if (!sliderContainer) {
6     console.error('Slider container not found');
7     return;
8   }
9
10  const slider = sliderContainer.querySelector('.slider');
11  const sliderValueElement = sliderContainer.querySelector('.sliderValue');
12
13  if (!slider || !sliderValueElement) {
14    console.error('Slider or value element not found');
15    return;
16  }
17
18  const savedValue = sliderValues[containerID];
19
20  if (savedValue !== undefined) {
21    // Den Slider auf den gespeicherten Wert setzen
22    slider.noUiSlider.set(savedValue);
23
24    // Anzeige des aktuellen Werts aktualisieren
25    sliderValueElement.textContent = Array.isArray(savedValue) ? savedValue.
26      join(' - ') : savedValue;
27  } else{
28    console.log("No saved value for " + containerID + " found in cookie");
29  }

```

Nun habe ich endlich eine solide Möglichkeit geschaffen, schnell und unkompliziert Variablen zur Laufzeit zu verändern und diese Änderungen sogar über einen Neustart der Webseite hinweg zu speichern. Dies wird mir hoffentlich eine Menge Zeit und Nerven sparen, wenn ich nun im nächsten Blogbeitrag mit der Implementierung der Kreiserkennung beginne.

4 Kreiserkennung

Die Kreiserkennung ist fertig! Und es war tatsächlich einfacher als gedacht. Die größte Herausforderung ist es gewesen, die optimalen Parameter für die Kreiserkennung zu finden. Aber mit meinen neuen Slidern war das auch kein Problem. Mehr dazu weiter unten.

4.1 Funktionsweise der HoughCircles-Methode

Um Kreise in einem Bild zu erkennen, gibt es in OpenCV.js praktischerweise bereits eine fertige Methode, die dies für uns übernimmt. Diese Methode heißt `cv.HoughCircles` und verwendet die Hough-Transformation, um Kreise in einem Bild zu finden. Die Methode hat folgende Signatur:

```
1 cv.HoughCircles(image, circles, method, dp, minDist, param1, param2, minRadius,  
    maxRadius);
```

Gehen wir fürs Erste die einzelnen Parameter durch:

image - Dies ist das Bild, in dem die Kreise gefunden werden sollen. Es muss eine openCV-Matrix sein und sollte als Graustufenbild vorliegen.

circles - In diese Matrix werden die gefundenen Kreise gespeichert. Ja - die Methode schreibt die Kreise direkt in eine openCV-Matrix. In OpenCV können Matrizen nicht nur für die Speicherung von Bildern verwendet werden, sondern auch für die Speicherung von Vektoren. In diesem Fall wird in der Matrix „circles“ in jeder Zeile ein Kreis gespeichert. Ein Kreis ist dabei ein Vektor der Form $[x, y, r]$, wobei (x, y) die Koordinaten des Mittelpunkts des Kreises und r der Radius des Kreises sind.

Weiter unten zeige ich eine Möglichkeit, wie die einzelnen Kreise in dieser Matrix in Objekte umgewandelt werden können. Dies macht die weitere Verarbeitung der Kreise deutlich einfacher.

method - Dies ist eine Konstante, die angibt, welche Methode zur Kreiserkennung verwendet werden soll. In openCV.js gibt es aktuell nur eine Methode, die verwendet werden kann, nämlich `cv.HOUGH_GRADIENT`.

dp - Hier wird es leider etwas komplizierter. Der Parameter `dp` ist ein inverser Skalierungsfaktor, der die Genauigkeit der Kreiserkennung beeinflusst. Ein Wert von 1 bedeutet, dass die Auflösung des Eingabebildes verwendet wird. Ein Wert von 2 bedeutet, dass das Eingabebild halb so groß wie das Originalbild ist. Je kleiner der Wert, desto genauer ist die Kreiserkennung, aber auch desto langsamer.

minDist - Dieser Parameter gibt den minimalen Abstand zwischen den Mittelpunkten der gefundenen Kreise an. Dies ist sehr nützlich um die Anzahl an falschen Positiven zu reduzieren.

param1 - Dieser Parameter ist spezifisch für die Methode `cv.HOUGH_GRADIENT`. Es handelt sich hier um den oberen Schwellwert für die Kantenerkennung, welche intern vor dem Suchen nach Kreisen durchgeführt wird. Der untere Schwellwert ist stets die Hälfte dieses Wertes.

param2 - Dieser Parameter ist ebenfalls spezifisch für die Methode `cv.HOUGH_GRADIENT`. Er ist der Schwellwert für die Kreiserkennung. Je kleiner der Wert, desto empfindlicher ist die Kreiserkennung, d.h. ein Kreis muss weniger stark ausgeprägt sein, um trotzdem als solcher erkannt zu werden. Dies bedeutet jedoch auch, dass das mit kleinerem Wert das Risiko für falsche Kreise steigt.

Es lässt sich leider nicht im Vorhinein sagen, welche Werte für die Parameter `dp`, `minDist`, `param1` und `param2` optimal sind. Diese Werte hängen stark von Bild, Licht und Kamera ab. Nur mit genau abgestimmten Werten kann eine gute und verlässliche Kreiserkennung erreicht werden.

Der Einfachheit halber habe ich eine eigene Methode nur für das Finden von Kreisen geschrieben, die die Methode `cv.HoughCircles` verwendet. Diese Methode gibt dann ein Array von Kreis-Objekten zurück, die ich dann weiterverarbeiten kann. Die Methode sieht so aus:

```

1 function FindCircles(inputMat, guiMat){
2
3     //Eingabe-Matrix in Graustufen umwandeln
4     cv.cvtColor(inputMat, grayMat, cv.COLOR_RGBA2GRAY);
5
6     //reset circlesMat
7     circlesMat.delete();
8     circlesMat = new cv.Mat();
9
10    //Hough-Transformation
11    cv.HoughCircles(grayMat, circlesMat, cv.HOUGH_GRADIENT, dp, minRadius, param1,
12                    param2, minRadius, maxRadius);
13
14    //create circle objects
15    let foundCircles = [];
16    for (let i = 0; i < circlesMat.cols; ++i) {
17        let x = circlesMat.data32F[i * 3];
18        let y = circlesMat.data32F[i * 3 + 1];
19        let radius = circlesMat.data32F[i * 3 + 2];
20
21        foundCircles.push(new Circle(x, y, radius));
22    }
23
24    //draw circles
25    for(let i = 0; i < foundCircles.length; i++){
26        DrawCircle(foundCircles[i], guiMat, [255,255,255,255]);
27    }
28
29    return foundCircles;
}

```

Man beachte, dass ich ein Array vom Typ `[Circle]` zurückgebe. Der Einfachheit halber extrahiere ich die gefundenen Kreise aus der `circlesMat`-Matrix und speichere sie in einem Array von Kreis-Objekten. Dies macht die weitere Verarbeitung der Kreise deutlich einfacher.

Auch für das Anzeigen der Kreise habe ich eine ausgelagerte Methode geschrieben. Dies wäre eigentlich nicht nötig gewesen, da für das Zeichnen von Kreisen nur eine einziger Methodenaufruf benötigt wird. Ich habe die Methode dennoch geschrieben, um die Lesbarkeit des Codes zu erhöhen. Die Methode sieht so aus:

```

1 function DrawCircle(circle, guiMat, color){
2     //return if parameter is not set
3     if(circle === undefined){
4         console.error("Circle is undefined");
5         return;
6     }
7
8     //draw circle
9     cv.circle(guiMat, new cv.Point(circle.x, circle.y), circle.radius, color, 2);
10 }

```

Bei den Parametern `dp`, `param1` und `param2` stellte sich mir nun die Frage, welche Werte ich hier verwenden soll. Eine pauschale Antwort darauf gibt es nicht, da die optimalen Werte wie bereits erwähnt stark vom Bild, vom Verwendungszweck und in meinem Beispiel auch von der Kamera abhängen.

Um nun die passenden Werte ausfindig zu machen, habe ich den Mainloop so angepasst, dass in einer Schleife die Kreiserkennung auf dem aktuellen Kamera-Frame durchgeführt wird. Die gefundenen Kreise werden dann auf eine separate "GUI-Matrix" gezeichnet, die dann auf dem Bildschirm angezeigt wird. Durch das Verändern der Parameter per Slider konnte ich so direkt die Auswirkungen auf die Kreiserkennung sehen.

Letztendlich habe ich bei diesen Werten sehr gute Ergebnisse erzielt:

- `dp = 2`
- `minDist = 18`
- `param1 = 230`
- `param2 = 129`
- `minRadius = 18`
- `maxRadius = 67`

Dies bedeutet nicht, dass diese Werte auch für andere Anwendungsfälle optimal sind. Je nach Lichtverhältnissen, Kamerawinkel und Bildqualität können diese Werte stark variieren. Wenn du also die Kreiserkennung in deinem eigenen Projekt verwenden möchtest, empfehle ich dir wärmstens, die Parameter selbst zu testen und anzupassen.

Mit der Kreiserkennung fertig implementiert, sind wir dem Ziel, Münzen zu erkennen, ein ganzes Stück näher gekommen. Jetzt wo wir Kreise (und somit auch Münzen) im Bild erkennen können, können wir uns daran machen, die Münzen zu klassifizieren. Dazu aber mehr beim nächsten Eintrag.

5 Template Matching

Also gut, ich habe also die Kreiserkennung implementiert. Jetzt möchte ich die Münzen klassifizieren. Doch wie mache ich das?

5.1 Überlegungen

Um zur Laufzeit Münzen klassifizieren zu können, muss ich dem Programm vorher beibringen, wie diese Münzen aussehen. Dafür könnte ich für jede Münze ein einzelnes Bild abspeichern, welches dann als Vorlage für die Klassifizierung dienen könnte.

Doch wie kann man in openCV.js Bilder miteinander vergleichen und auf Ähnlichkeit prüfen? Dafür gibt es viele Mittel. Man könnte beispielsweise Histogramme für beide Bilder erstellen, und diese dann mit vorgefertigten Vergleichsalgorithmen wie der Bhattacharyya-Distanz untersuchen. Jedoch geben Histogramme nur Auskunft über die Wahrscheinlichkeitsverteilung der Farbwerte. Wichtige Informationen wie die Ziffer der Münze hätten wahrscheinlich im Histogramm keinen großen Einfluss. Es wäre dann sehr schwierig, beispielsweise den Unterschied zwischen einer 1-Cent-Münze und einer 2-Cent-Münze zu erkennen, da sich die Farben der Münzen und somit auch die Histogramme nur minimal unterscheiden würden.

Eine andere Möglichkeit ist das Template Matching. Dabei wird ein Template, also eine Vorlage, über ein Bild geschoben und die Ähnlichkeit an jedem Punkt beziehungsweise Pixel berechnet. Das Ergebnis ist eine Heatmap, die die Übereinstimmung an jedem Punkt im Bild anzeigt.

Nun möchte ich jedoch nicht eine Heatmap, die mir die Übereinstimmung für jeden Pixel anzeigt, sondern ein generelles Maß für die Ähnlichkeit des Templates zum Bild. Wenn ich vor dem Vergleich dafür sorge, dass Template und Ausgangsbild die selbe Größe haben, sollte pro Vergleich lediglich ein Wert herauskommen, der die Ähnlichkeit des Templates zum gesamten Bild beschreibt. Diesen Wert könnte ich dann dafür nutzen, um die Vorlage mit der höchsten Übereinstimmung zu finden und somit die Münze zu klassifizieren.

5.2 Die Vorlagen

Damit ich die Münzen anhand ihrer Bildinformationen klassifizieren kann, benötige ich also Vorlagen, mit denen ich die Münzen vergleichen kann. Ich habe demnach Bilder von den Münzen gemacht und für jede Münze ein Bild abgespeichert. Diese Vorlagenbilder sind die Referenzbilder, anhand derer ich die Münzen im Kamerabild auf Ähnlichkeit überprüfen möchte.

Hier hat sich besonders die spiegelnde Oberfläche der Münzen als Problem herausgestellt. Ich musste viel mit dem Lichteinfallswinkel und der Beleuchtung spielen, um von jeder Münze ein gutes Bild zu bekommen. Auch der Abnutzungsgrad hat das Erstellen der Vorlagen erschwert. So musste ich insbesondere bei den 50-20-10 Cent Münzen und bei den 5-2-1 Cent Münzen darauf achten, dass die Münzen stets eine ähnliche Optik ausweisen. Andernfalls könnte es passieren, dass die Spiegelung oder die Abnutzung als Unterschied interpretiert wird und nicht mehr die Ziffern auf den Münzen.

Bei jeder Vorlage nimmt die Münze das gesamte Bild ein. Sie sehen so aus:

Bevor ich nun mit dem Vergleich der Vorlagen beginne, brauche ich zunächst eine Möglichkeit um sowohl die Vorlagen als auch die daraus resultierenden Ergebnisse zu speichern. Dafür habe ich ein global verfügbares Objekt *COINS* erstellt, welches die Münzen als Schlüssel enthält und als Wert ein Objekt mit den Eigenschaften *diameter* und *value* hat. Weitere Eigenschaften können dann zur Laufzeit



den einzelnen Münzen hinzugefügt werden.

```

1 let COINS = {
2     Euro2 : {
3         diameter: 25.75,
4         value: 2
5     },
6     Euro1 : {
7         diameter: 23.25,
8         value: 1
9 },
10    Cent50 : {
11        diameter: 24.25,
12        value: 0.5
13 },
14    Cent20 : {
15        diameter: 22.25,
16        value: 0.2
17 },
18    Cent10 : {
19        diameter: 19.75,
20        value: 0.1
21 },
22    Cent5 : {
23        diameter: 21.25,
24        value: 0.05
25 },
26    Cent2 : {
27        diameter: 18.75,
28        value: 0.02
29 },
30    Cent1 : {
31        diameter: 16.25,
32        value: 0.01
33 }
34 }
```

Diese Datenstruktur kann ich nun verwenden, um die Vorlagen zu speichern und ihnen eine Münze zuzuordnen. In der folgenden Methode iteriere ich über jede Münze und lese das Vorlagenbild ein:

```

1 const templatesLoaded = new Event('templatesLoaded');
2 let path = "../Templates/"
3
4 function InitTemplates() {
5     let coinLength = Object.keys(COINS).length;
```

```

6  let loadedCoins = 0;
7  Object.entries(COINS).forEach(([key, value]) => {
8      //is in the template folder a picture with the same name as the key?
9      let img = new Image();
10
11     img.onload = () => {
12         //save image as matrix in the coin object
13         COINS[key].template = cv.imread(img);
14
15         loadedCoins++;
16         if(loadedCoins === coinLength){
17             document.dispatchEvent(templatesLoaded);
18         }
19     }
20
21     img.onerror = () => {
22         console.log("error: " + key);
23     }
24
25     img.src = path + key + ".png";
26
27 });
28 }

```

Das Event *templatesLoaded* wird ausgelöst, sobald alle Vorlagen geladen wurden. Somit kann ich sicherstellen, dass Programmschritt welche die Vorlagen benötigen erst ausgeführt werden, wenn die Vorlagen auch wirklich geladen wurden.

5.3 Funktionsweise

Für das eigentliche Template-Matching benutzte ich in openCV.js die folgende Funktion:

```
1 cv.matchTemplate(image, templ, result, method, mask);
```

Wie bereits erwähnt, berechnet diese Funktion die Übereinstimmung eines Templates mit einem Bild. Dabei wird das Template über das Bild geschoben und an jedem Punkt die Übereinstimmung berechnet. Zurück gegeben wird eine Heatmap, die die Übereinstimmung an jedem Punkt im Bild anzeigt. Glücklicherweise benötigen die Parameter nicht so viel Trial-and-Error wie bei der Kreiserkennung. Die Parameter sind wie folgt:

- **image** - Das Bild, in dem das Template gesucht werden soll. Es muss als openCV-Matrix vorliegen.
- **templ** - Das Template, das gesucht werden soll. Es muss auch als openCV-Matrix vorliegen.
- **result** - Die Heatmap, in der die Übereinstimmung an jedem Punkt im Bild gespeichert wird. Auch diese muss als openCV-Matrix vorliegen.
- **method** - Die Methode, die zur Berechnung der Übereinstimmung verwendet werden soll. Aktuell gibt es folgende Methoden in openCV.js:

cv.TM_SQDIFF - Summe der quadrierten Differenzen (kleinere Werte bedeuten bessere Übereinstimmung)

cv.TM_SQDIFF_NORMED - Normalisierte Summe der quadrierten Differenzen (kleinere Werte bedeuten bessere Übereinstimmung)

cv.TM_CCORR - Kreuzkorrelation (größere Werte bedeuten bessere Übereinstimmung)

cv.TM_CCORR_NORMED - Normalisierte Kreuzkorrelation (größere Werte bedeuten bessere Übereinstimmung)

cv.TM_CCOEFF - Kreuzkorrelationskoeffizient (größere Werte bedeuten bessere Übereinstimmung)

cv.TM_CCOEFF_NORMED - Normalisierter Kreuzkorrelationskoeffizient (größere Werte bedeuten bessere Übereinstimmung)

- **mask** - (optional) Eine Maske, die angibt, welche Bereiche des Bildes berücksichtigt werden sollen. In meinem Fall benutze ich keine Maske, da ich bereits vor Aufruf der Funktion unerwünschte Bereiche des Bildes ausschneide.

Der einzige Parameter welcher etwas Ausprobieren erfordert hat, war die Methode. Als erstes scheint es, als würden die Methoden *cv.TM_SQDIFF_NORMED* und *cv.TM_CCOEFF_NORMED* die besten Ergebnisse erzielen.

Meine Funktion für das Template Matching sieht nun so aus:

```
1 function MatchTemplates(src, circle){  
2     //create result string  
3     let resultsString = [];  
4     Object.entries(COINS).forEach(([key, value]) => {  
5         let resultMat = new cv.Mat();  
6  
7         //resize template to the size of src  
8         let templateResized = new cv.Mat();  
9         cv.resize(COINS[key].template, templateResized, COINS[key].template.size(),  
10            0, 0, cv.INTER_AREA);  
11  
12         cv.matchTemplate(src, templateResized, resultMat, cv.TM_SQDIFF_NORMED);  
13  
14         //get highest value  
15         let minMax = cv.minMaxLoc(resultMat);  
16         let min = minMax.minVal;  
17  
18         resultsString.push(new Result(key, min));  
19  
20         templateResized.delete();  
21     });  
22  
23     //sort results from highest to lowest  
24     resultsString.sort((a, b) => a.value - b.value);  
25  
26     //console.dir(resultsString);  
27  
28     //save bestmatch in circle  
29     circle.bestMatch = COINS[resultsString[0].name];  
30     circle.matchValue = resultsString[0].value;  
31  
32     //set matchValue to 2 decimal places  
33     circle.matchValue = Math.round(circle.matchValue * 100) / 100;  
34  
35     src.delete();  
}
```

Innerhalb des main-loops wird diese Funktion nun für jeden gefundenen Kreis aufgerufen. Der Parameter *src* ist dabei eine ausgeschnittene Region des Bildes, in der sich der Kreis befindet. Der Parameter "circle" ist das Circle-Objekt, in dem die Informationen zum Kreis gespeichert sind. Die Funktion iteriert anschließend über alle Vorlagen und berechnet die Übereinstimmung für jede Vorlage. Hierbei muss beachtet werden, dass Vorlage und das zu prüfende Bild vorher auf die selbe Größe gebracht werden müssen. Die Ergebnisse werden in einem Array gespeichert und anschließend sortiert. Das beste Ergebnis wird dann im Circle-Objekt gespeichert.

Du fragst dich bestimmt wie ich an die ausgeschnittene Matrix eines jeden Kreises komme. Dafür habe ich in der Circle-Klasse eine eigene Methode geschrieben, die die ausgeschnittene Region des Bildes zurückgibt. Diese Methode sieht so aus:

```

1  GetImageData(src){
2      let clippedImage = src.roi(new cv.Rect(this.x - this.radius, this.y - this.
3          radius, this.radius * 2, this.radius * 2));
4
5      return ClipCorners(clippedImage);
}

```

Dabei benutze ich die Methode `roi()`, welche es mir ermöglicht, für jede Matrix eine Region of Interest (ROI) zu definieren. Diese ROI wird dann als neue Matrix zurückgegeben. Mithilfe der XY-Koordinaten und des Radius des Kreises kann ich so in einer Matrix eine kleinere Teilmatrix erstellen, die nur den Kreis enthält. Diese Teilmatrix wird dann an die `ClipCorners()`-Funktion übergeben, welche die Ecken der Matrix abschneidet, damit das endgültige Bild nur Pixel enthält, die auch wirklich zum Kreis beziehungsweise der Münze gehören. Die Funktion `ClipCorners()` funktioniert wie folgt:

```

1  function ClipCorners(src){
2      let radius = src.cols/2;
3      let center = new cv.Point(src.cols/2, src.rows/2);
4
5      let mask = cv.Mat.zeros(src.rows, src.cols, cv.CV_8UC1);
6      cv.circle(mask, center, radius, [255, 255, 255, 255], -1);
7
8      let maskedImage = new cv.Mat();
9      let channels = new cv.MatVector();
10     cv.split(src, channels);
11
12     //apply mask to each channel
13     for(let i = 0; i < channels.size(); i++){
14         cv.bitwise_and(channels.get(i), mask, channels.get(i));
15     }
16
17     //merge channels to one image
18     cv.merge(channels, maskedImage);
19
20     //delete mats
21     mask.delete();
22     channels.delete();
23
24     return maskedImage;
25 }

```

Diese Funktion erstellt eine Maske, die nur die Pixel enthält, die zum Kreis gehören. Um die Maske anwenden zu können, muss die Matrix in ihre einzelnen Kanäle aufgeteilt werden. Anschließend wird die Maske auf jeden Kanal angewendet, bevor die Kanäle wieder zu einem Bild zusammengefügt werden. Das Ergebnis ist ein Bild, welches nur die Pixel enthält, die zum Kreis gehören. Dieses Bild wird dann zurückgegeben.

5.4 Exkurs: Mehrere Matrizen in einem Canvas anzeigen

Bei wachsender Komplexität der Webanwendung ob ich mir immer wieder gewünscht eine Möglichkeit zu haben, mehrere Matrizen in einem Canvas anzuzeigen. Dies ist standardmäßig in openCV.js nicht vorgesehen, da die Funktion `cv.imshow()` immer nur eine Matrix anzeigen kann. Also habe ich mir eine eigene Funktion geschrieben, die mehrere Matrizen in einem Canvas anzeigen kann. Diese Funktion sieht so aus:

```
1  function ShowMatrices(src, canvas) {
```

```

2   if (src.length === 0) {
3     console.error("No src to show");
4     return;
5   }
6
7   if (src.length === 1) {
8     console.warn("Only one matrix to show. Using ShowMatrix instead");
9     ShowMatrix(src[0], canvas);
10    return;
11  }
12
13  let number0fMatrices = src.length;
14
15 // Berechnung der Gittergröße
16 let gridCols = Math.ceil(Math.sqrt(number0fMatrices));
17 let gridRows = Math.ceil(number0fMatrices / gridCols);
18
19 // Maximalbreite und -höhe der Matrizen basierend auf der Größe des Canvas
20 let cellWidth = canvas.width / gridCols;
21 let cellHeight = canvas.height / gridRows;
22
23 // Canvas leeren
24 let ctx = canvas.getContext('2d');
25 ctx.clearRect(0, 0, canvas.width, canvas.height);
26
27 // Jede Matrix im Gitter zeichnen
28 src.forEach((mat, index) => {
29   // Spalte und Zeile bestimmen
30   let col = index % gridCols;
31   let row = Math.floor(index / gridCols);
32
33   // Zielbereich für diese Matrix
34   let x = col * cellWidth;
35   let y = row * cellHeight;
36   let targetSize = new cv.Size(cellWidth, cellHeight);
37
38   // Matrix auf passende Größe skalieren
39   let resizedMat = new cv.Mat();
40   cv.resize(mat, resizedMat, targetSize, 0, 0, cv.INTER_AREA);
41
42   // Matrix in RGBA umwandeln wenn es sich um eine Graustufenmatrix handelt
43   if (resizedMat.channels() === 1) {
44     let rgbaMat = new cv.Mat();
45     cv.cvtColor(resizedMat, rgbaMat, cv.COLOR_GRAY2RGBA);
46     resizedMat.delete();
47     resizedMat = rgbaMat;
48   }
49
50   // Erstelle ein temporäres Canvas für diese Matrix
51   let tempCanvas = document.createElement('canvas');
52   let tempCtx = tempCanvas.getContext('2d');
53   tempCanvas.width = resizedMat.cols;
54   tempCanvas.height = resizedMat.rows;
55
56   // In ein ImageData konvertieren und in das temporäre Canvas zeichnen
57   let imageData = new ImageData(new Uint8ClampedArray(resizedMat.data),
58                               resizedMat.cols, resizedMat.rows);
59   tempCtx.putImageData(imageData, 0, 0);
60
61   // Zeichne das temporäre Canvas auf das Haupt-Canvas
62   ctx.drawImage(tempCanvas, 0, 0, resizedMat.cols, resizedMat.rows, x, y,
63                 cellWidth, cellHeight);
64
65   // Bereinige die temporären Matrizen

```

```

64     resizedMat.delete();
65 }
66 }
```

Um mehrere Bilder in einen Canvas zu zeichnen, musste ich die Größe der Bilder anpassen, damit sie in ein Grid passen. Dafür habe ich die Funktion `cv.resize()` verwendet. Anschließend müssen die Matrizen zuerst in ein temporäres Canvas gezeichnet werden, bevor sie in das Haupt-Canvas gezeichnet werden können. Die Funktion `drawImage()` von Canvas kann nur Bilder zeichnen, keine Matrizen. Daher musste ich die Matrizen in ein `ImageData`-Objekt umwandeln, bevor ich sie in das Canvas zeichnen konnte.

5.5 Erste Ergebnisse

Es hat sich gezeigt, dass der Vergleich der Vorlagen prinzipiell funktionieren kann. Jedoch ist die Genauigkeit noch nicht zufriedenstellend. Münzen können aktuell zwar grob in die richtige Kategorie eingeordnet werden (sprich 1-2-5 Cent Münzen und 10-20-50 Cent Münzen), jedoch ist die Unterscheidung innerhalb dieser Kategorien nicht funktional. So werden aktuell nur maximal die Hälfte der Münzen korrekt zugeordnet.

Es scheint, als wären die Informationen der Ziffern auf den Münzen nicht ausreichend, um eine genaue Klassifizierung zu ermöglichen. Die Farbe einer Münze hat im Gegensatz dazu einen relativ großen Einfluss auf das Ergebnis. So werden Münzen meist der korrekten Farbkategorie zugeordnet, jedoch nicht dem korrekten Wert. Dies ist ein Hinweis darauf, dass die Farbinformationen der Münzen stärker gewichtet werden als die Forminformationen. Da sich Münzen innerhalb einer Kategorie (z.B. 1-2-5 Cent Münzen) jedoch nur in der Größe und ihrer Ziffer optisch unterscheiden, ist es schwierig eine weitere Methode der Klassifizierung zu finden. Ein Prüfen der Größe würde in meinem Anwendungsfall nicht funktionieren, da sowohl der Abstand zur Kamera als auch die Position der Münzen im Bild variabel sind.

Eine Möglichkeit die Genauigkeit zu erhöhen wäre es, den störenden Faktor der Farbinformationen zu eliminieren, und stattdessen nur Vorlagen zu verwenden, welche nur die notwendigen Informationen enthalten. Dies könnte z.B. durch das Erstellen von Kantenbildern der Münzen erreicht werden. Solche Bilder wie sie vom Canny-Algorithmus erstellt werden, enthalten nur die Kanten des Ausgangsbildes und keine Farbinformationen, da sie rein als Schwarz-Weiß-Bilder vorliegen. Diese Kantenbilder könnten dann an Stelle der aktuellen Farbbilder als Vorlagen verwendet werden.

Jedoch gibt es noch ein weiteres Problem: die Rotation. Bereits eine kleine Drehung des Ausgangsbildes verursacht große Unterschiede in der Übereinstimmung. Das Template Matching in openCV berechnet standardmäßig nur die Übereinstimmung für das Template in der gleichen Rotation wie das Bild. Dies ist ein Problem, da die Münzen im Kamerabild in unterschiedlichen Rotationen vorkommen können. Eine Möglichkeit dieses Problem zu lösen wäre es, das Template in verschiedenen Rotationen zu speichern und für jede Rotation die Übereinstimmung zu berechnen. Dies würde jedoch die Anzahl der Vorlagen vervielfachen und einen unverhältnismäßig hohen Aufwand bedeuten. Eine andere Möglichkeit wäre es, das Ausgangsbild mehrfach zu rotieren und für jede Rotation die Übereinstimmung zu den Vorlagen berechnen. So wäre bei ausreichend kleinen Rotationsschritten sichergestellt, dass jede Münze mindestens einmal eine ähnliche Ausrichtung wie das zutreffende Template hat. Dies würde jedoch die Rechenzeit signifikant erhöhen, da bedeutend mehr Berechnungen pro Münze durchgeführt werden müssten. Zudem braucht es dafür eine Möglichkeit, alle Zwischenergebnisse der Rotationen zu speichern, um anschließend das beste Ergebnis zu wählen.

Alles in allem ein durchwachsenes Ergebnis. Die Klassifizierung der Münzen ist noch nicht zufriedenstellend, jedoch gibt es noch viele Möglichkeiten, die Genauigkeit zu erhöhen. Ich werde mich in den

nächsten Tagen mit diesen Möglichkeiten beschäftigen und einige Änderungen vornehmen. Hoffentlich kann ich so die Genauigkeit der Klassifizierung noch um einiges erhöhen.

6 Anpassungen

6.1 Kantenerkennung

Die Kantenerkennung ist ein häufig genutztes Verfahren in der Bildverarbeitung, um Objekte beziehungsweise geometrische Formen in Bildern zu erkennen. Sie ist häufig ein wichtiger Schritt für weitere Verarbeitungsschritte. So möchte auch ich die Kantenerkennung nun nutzen, um für mein Template Matching bessere Ergebnisse zu erzielen

6.1.1 Funktionsweise der Canny-Methode

. Um in openCV.js Kanten zu erkennen, muss der Canny-Algorithmus genutzt werden. Die Methode sieht wie folgt aus:

```
1 cv.Canny(src, dst, threshold1, threshold2, apertureSize, L2gradient);
```

- **src** - Die Eingabematrix, in der die Kanten erkannt werden sollen. Für gewöhnlich wird sie vorher in ein Graustufenbild umgewandelt.
- **dst** - Die Ausgabematrix, in der die Kanten gespeichert werden.
- **threshold1** - Der erste Schwellenwert für die Hysterese. Der Algorithmus nutzt zwei Schwellenwerte, um die Kanten zu erkennen. Der erste Schwellenwert ist der niedrigere Schwellenwert, der dazu genutzt wird, um Pixel als Kanten zu markieren, wenn sie stärker sind als der Schwellenwert.
- **threshold2** - Der zweite Schwellenwert für die Hysterese. Der zweite Schwellenwert ist der höhere Schwellenwert, der dazu genutzt wird, um Pixel als Kanten zu markieren, wenn sie mit einem starken Pixel verbunden sind.
- **apertureSize** - Die Größe des Sobel-Operators, der für die Kantenerkennung genutzt wird. Ein kleinerer Wert bedeutet eine genauere Kantenerkennung, aber auch mehr Rauschen.
- **L2gradient** - Ein boolscher Wert, der angibt, ob die L2-Norm für die Gradienten berechnet werden soll. Wenn der Wert *true* ist, wird die L2-Norm berechnet, ansonsten die L1-Norm. Die L2-Norm ist genauer, aber auch rechenaufwändiger.

Wie man sieht, arbeitet der Canny-Algorithmus mit zwei Parametern *threshold1* und *threshold2*, die die Schwellenwerte für die Hysterese darstellen. Für beide Parameter gilt: ein zu niedriger Wert führt zu einer hohen Anzahl an Kanten, ein zu hoher Wert zu einer geringen Anzahl an Kanten. Um die richtigen Werte zu finden, bedarf es also wieder viel "trial and error". Meine zuvor erstellten Slider werden sich hierbei als sehr nützlich erweisen.

6.1.2 Die korrekten Parameter finden

Während ich versuchte die korrekten Parameter zu ermittelten stellte sich relativ schnell heraus, dass ich für die Vorlagen andere Parameter benötige als für das Kamerabild. Die Vorlagen sind sehr klar und zudem deutlich größer als die Münzen im Kamerabild. Daher benötigen sie auch andere Schwellenwerte. Um dieses Problem zu lösen, habe ich für beide Anwendungsfälle ein eigene SSettingsObjekt erstellt, in dem die Parameter für die Kantenerkennung gespeichert werden. So kann ich für die Vorlagen und das Kamerabild unterschiedliche Parameter nutzen:

```

1 class Settings{
2     constructor(threshold1, threshold2, apertureSize, blurSize){
3         this.threshold1 = threshold1;
4         this.threshold2 = threshold2;
5         this.apertureSize = apertureSize;
6         this.blurSize = blurSize;
7     }
8 }
9
10 let SettingsTemplate = new Settings(9400, 23300, 7, 7);
11 let SettingsLive = new Settings(79, 56, 3, 1);

```

Meine Methode für die Kantenerkennung sieht entsprechend wie folgt aus:

```

1 function DetectEdges(src, settings){
2     if(!CheckForCorrectMatType(src, [MatTypes.CV_8UC4, MatTypes.CV_8UC1])) return
3         null;
4
5     //create output mat
6     let edgesMat = new cv.Mat();
7
8     //create gray mat
9     let grayMat = new cv.Mat();
10
11    if(src.type() === MatTypes.CV_8UC4){
12        cv.cvtColor(src, grayMat, cv.COLOR_RGBA2GRAY, 0);
13    }else if(src.type() === MatTypes.CV_8UC1) {
14        //clone the matrix
15        console.warn("src is already a gray matrix. Cloning it");
16        grayMat = src.clone();
17    }else{
18        console.error("Unsupported type of src-matrix: " + GetMatrixType(src));
19        return null;
20    }
21
22    if(settings === undefined){
23        console.warn("No settings provided. Using template settings");
24        settings = SettingsTemplate;
25    }
26
27    //clamp apertureSize to 3, 5 or 7
28    let allowedValues = [3, 5, 7];
29    settings.apertureSize = allowedValues.includes(settings.apertureSize)
30        ? settings.apertureSize
31        : allowedValues.reduce((closest, current) =>
32            Math.abs(current - settings.apertureSize) < Math.abs(closest - settings.
33                apertureSize) ? current : closest
34        );
35
36    //gaussian blur
37    allowedValues = [1, 3, 5, 7];
38    settings.blurSize = allowedValues.includes(settings.blurSize)
39        ? settings.blurSize
40        : allowedValues.reduce((closest, current) =>
41            Math.abs(current - settings.blurSize) < Math.abs(closest - settings.
42                blurSize) ? current : closest
43        );
44    cv.GaussianBlur(grayMat, grayMat, new cv.Size(settings.blurSize, settings.
45        blurSize), 0, 0, cv.BORDER_DEFAULT);
46
47    //detect edges
48    cv.Canny(grayMat, edgesMat, settings.threshold1, settings.threshold2, settings.
49        apertureSize, L2gradient);
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
637
638
639
639
640
641
642
643
644
645
645
646
647
647
648
649
649
650
651
652
653
653
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523

```

```

46     //delete mats
47     grayMat.delete();
48
49     return edgesMat;
50 }

```

Der Großteil der Funktion stellt lediglich sicher, dass alle Parameter einen gültigen Wert haben. So sind für *apertureSize* und *blurSize* nur die Werte 3, 5 und 7 bzw. 1, 3, 5 und 7 erlaubt. Sollte ein anderer Wert übergeben werden, wird der nächstgelegene verwendet. Vor der eigentlichen Kantenerkennung wird zudem eine Gaußsche Unschärfe auf das Graustufenbild angewendet. Dies kann mitunter helfen, das Rauschen zu reduzieren und die Kantenerkennung zu verbessern. Auch dieser Wert muss jedoch wie alle anderen feinfühlig per Slider eingestellt werden.

Diese Methode wird nun sowohl für jede Vorlage als auch für das Kamerabild aufgerufen. Das Template Matching arbeitet somit nicht mehr mit den Originalbildern, sondern mit den Kantenbildern. Dies sollte die Ergebnisse deutlich verbessern.

Wie dir vielleicht aufgefallen ist, habe ich in der Methode einen weiteren Check eingebaut, der überprüft, ob die übergebene Matrix ein gültiger Typ ist. Dies ist die Folge eines Problems, welches sich mir bei der Implementierung des Template Matchings stellte. Dazu mehr im nächsten Abschnitt.

6.2 Neues Problem: Mat-Types

Wie mir ja bereits seit einiger Zeit bekannt ist, wird in openCV die Matrixklasse *Mat* genutzt, um Bilder zu speichern und zu verarbeiten. Diese Klasse hat jedoch eine Eigenschaft, die mir bislang noch nicht aufgefallen ist: den *type()*. So hat nämlich jede Matrix einen Typ, der angibt, wie die Matrix aufgebaut ist. Leider wird dieser Typ nur als Integer zurückgegeben, was zunächst nicht gerade hilfreich ist. Praktischerweise gibt es im Internet jedoch Listen, in denen die Integerwerte den Typen zugeordnet werden können. So habe ich mir eine solche Methode erstellt, die mir anhand des Integerwertes den Typ der Matrix zurückgibt:

```

1 function GetMatrixType(mat) {
2     const types = {
3         0: "CV_8UC1", 1: "CV_8SC1", 2: "CV_16UC1", 3: "CV_16SC1",
4         4: "CV_32SC1", 5: "CV_32FC1", 6: "CV_64FC1",
5         8: "CV_8UC2", 9: "CV_8SC2", 10: "CV_16UC2", 11: "CV_16SC2",
6         12: "CV_32SC2", 13: "CV_32FC2", 14: "CV_64FC2",
7         16: "CV_8UC3", 17: "CV_8SC3", 18: "CV_16UC3", 19: "CV_16SC3",
8         20: "CV_32SC3", 21: "CV_32FC3", 22: "CV_64FC3",
9         24: "CV_8UC4", 25: "CV_8SC4", 26: "CV_16UC4", 27: "CV_16SC4",
10        28: "CV_32SC4", 29: "CV_32FC4", 30: "CV_64FC4"
11    };
12
13    return types[mat.type] || 'Unknown type: ${mat.type}';
14 }

```

6.3 Matrizen rotieren und transformieren

Für meine MatchTemplate-Methode habe ich die Anforderung eine openCV-Matrix um einen beliebigen Winkel zu rotieren. Leider gibt es in openCV.js keine Methode, die dies direkt ermöglicht. Daher habe ich mir eine eigene Methode erstellt, die eine Matrix um einen beliebigen Winkel rotiert:

```

1 function RotateMatrix(src, dist, angle){
2
3     let center = new cv.Point(src.cols / 2, src.rows / 2);

```

```

4 let interpolation = cv.INTER_LINEAR;
5 let borderMode = cv.BORDER_CONSTANT;
6 let borderValue = new cv.Scalar(0, 0, 0, 255);
7
8 //Rotationsmatrix erstellen
9 let rotationMatrix = cv.getRotationMatrix2D(center, angle, 1);
10
11 //Matrix rotieren
12 cv.warpAffine(src, dist, rotationMatrix, new cv.Size(src.cols, src.rows),
    interpolation, borderMode, borderValue);
13
14 //Matrix freigeben
15 rotationMatrix.delete();
16 }

```

Für das Rotieren benutze ich die cv-eigene Methode cv.warpAffine. Sie benötigt folgende Parameter:

- **src** - Die Eingabematrix, die rotiert werden soll.
- **dist** - Die Ausgabematrix, in der das rotierte Bild gespeichert wird.
- **angle** - Die Rotationsmatrix, die sowohl Grad als auch Mittelpunkt der Rotation enthält.
- **dsize** - Die Größe der Ausgabematrix. In meinem Fall ist sie gleich groß wie die Eingabematrix.
- **interpolation** - Die Interpolationsmethode, die genutzt werden soll. In meinem Fall nutze ich die LINEAR-Interpolation.
- **borderMode** - Der Randmodus, der genutzt werden soll. Er entscheidet, wie die Pixel am Rand des Bildes behandelt werden. In meinem Fall nutze ich den CONSTANT-Modus, der die Pixel mit einer festen Farbe füllt.
- **borderValue** - Die Farbe, mit der die Pixel am Rand gefüllt werden. In meinem Fall zeichne ich alle Pixel außerhalb des Bildes schwarz.

6.4 Asynchronität und Ladebalken

Da ich nun jeden Kreis mehrfach drehe und die Kantenerkennung für jedes Bild neu berechnen muss, dauert der Prozess des Template Matchings sehr lange. Der ursprüngliche Plan, die Methode in einer Schleife aufzurufen und das Ergebnis in Echtzeit anzuzeigen, ist somit nicht mehr möglich. Daher habe ich mich dazu entschieden, die Methode asynchron zu machen und einen Ladebalken einzufügen, der den Fortschritt anzeigen.

Ein Ladebalken kann in OpenCV relativ leicht mittels einigen Rechtecken erstellt werden:

```

1 function DrawLoadingBar(dist, value){
2     //clamp value between 0 and 1
3     value = Math.min(1, Math.max(0, value));
4
5     let outerMargin = 10;
6     let outerHeight = 15;
7     let innerMargin = 4;
8     let startPosition = new cv.Point(outerMargin, dist.rows-outerMargin-outerHeight
9         );
10    let endPosition = new cv.Point(dist.cols-outerMargin, dist.rows-outerMargin);
11
12    //draw outer frame
13    cv.rectangle(dist, startPosition, endPosition, [255, 255, 255, 255], -1);

```

```

14 //draw inner frame
15 let innerStartPosition = new cv.Point(startPosition.x + innerMargin,
16   startPosition.y + innerMargin);
16 let innerEndPosition = new cv.Point(endPosition.x - innerMargin, endPosition.y
17   - innerMargin);
17 cv.rectangle(dist, innerStartPosition, innerEndPosition, [0, 0, 0, 255], -1);
18
19 //draw progress bar
20 let progressEndPosition = new cv.Point(innerStartPosition.x + (innerEndPosition
20   .x - innerStartPosition.x) * value, innerEndPosition.y);
21 cv.rectangle(dist, innerStartPosition, progressEndPosition, [0, 175, 0, 255],
22   -1);
22 }

```

Die Methode benötigt die Ausgabematrix, in der der Ladebalken gezeichnet werden soll, und den Wert, der den Fortschritt angibt. Der Wert muss zwischen 0 und 1 liegen. Der Ladebalken besteht aus einem weißen Rahmen, einem schwarzen Hintergrund und einem grünen Balken, der den Fortschritt anzeigt.

Jedoch darf man nicht vergessen, dass JavaScript standardmäßig single-threaded ist. Das bedeutet, dass der Browser während der Berechnung des Template Matchings nicht mehr reagiert. Um dies zu verhindern, muss dem Interpreter die Möglichkeit gegeben werden, zwischendurch von der Aufgabe abzuspringen und andere Aufgaben, wie das Zeichnen des Ladebalkens und das Aktualisieren der UI zu erledigen. Dies kann mittels der *setTimeout*-Methode erreicht werden. Wenn man jedoch nicht die Methode wieder von vorne ausführen möchte, sondern direkt in der nächste Programmzeile fortfahren möchte, muss man die Methode mit einem Promise verbinden:

```
1 await new Promise(r => setTimeout(r, 0));
```

Diese Zeile bewirkt, dass der Interpreter die Methode verlässt und erst nach Ablauf der Zeit, die in *setTimeout* übergeben wird, wieder zurückkehrt. Da die Zeit auf 0 gesetzt ist, wird die Methode direkt im nächsten Frame fortgesetzt. Dies gibt dem Browser jedoch die Möglichkeit, andere Aufgaben zu erledigen.

7 Ergebnisse

7.1 Neue MatchTemplate-Funktion

Meine finale MatchTemplate-Funktion sieht nun so aus:

```
1  async function MatchTemplates(src, circle){
2      let matchType = cv.TM_CCOEFF_NORMED;
3      let iterations = 180;
4      let angleStep = 360 / iterations;
5      let allResults = [];
6
7      for(let i = 0; i < iterations; i++){
8          //pause so the browser can update the gui
9          await new Promise(r => setTimeout(r, 0));
10
11
12         let rotatedSrc = new cv.Mat();
13         if(i>0){
14             //rotate image
15             RotateMatrix(src, rotatedSrc, angleStep * i);
16         }else{
17             rotatedSrc = src.clone();
18         }
19
20         let templateSimilarity = [];
21         Object.entries(COINS).forEach(([key, value]) => {
22
23             //check if src and template have the same type
24             if(rotatedSrc.type() !== COINS[key].edges.type()){
25                 console.error("Type of src and template are not the same");
26                 console.error("src: " + GetMatrixType(rotatedSrc));
27                 console.error("template(edges): " + GetMatrixType(COINS[key].edges)
28                     );
29                 return;
30             }
31
32             let templateResized = new cv.Mat();
33             cv.resize(COINS[key].edges, templateResized, rotatedSrc.size(), 0, 0,
34                     cv.INTER_AREA);
35
36             let resultMat = new cv.Mat();
37             cv.matchTemplate(rotatedSrc, templateResized, resultMat, matchType);
38
39             //get highest and lowest value
40             let minMax = cv.minMaxLoc(resultMat);
41             let max = minMax.maxVal;
42             let min = minMax.minVal;
43
44             templateSimilarity.push(new Result(key, min, max));
45
46             //console.log("Match value: " + max);
47
48             //free memory
49             templateResized.delete();
50             resultMat.delete();
51         });
52
53         //find lowest and highest value
54         let lowest = templateSimilarity.reduce((prev, current) => (prev.min <
55             current.min) ? prev : current);
56         let highest = templateSimilarity.reduce((prev, current) => (prev.max >
57             current.max) ? prev : current);
```

```

54         allResults.push(new Result("Iteration " + i, lowest, highest));
55
56         //draw loading bar
57         DrawLoadingBar(guiMat, (i+1) / iterations);
58         DrawMemoryUsage(guiMat);
59         //ShowMatrix(guiMat, outputCanvas);
60         ShowMatrices([guiMat, COINS[lowest.name].edges, src, rotatedSrc],
61                      outputCanvas);
61     }
62
63     console.log("Results for all iterations:");
64     console.dir(allResults);
65
66     //sort results from highest to lowest
67     if (matchType === cv.TM_SQDIFF ||
68         matchType === cv.TM_SQDIFF_NORMED) {
69         //find lowest value
70         allResults.sort((a, b) => a.min.min - b.min.min);
71         circle.matchValue = allResults[0].min.min;
72         circle.bestMatch = allResults[0].min.name;
73     } else{
74         //find highest value
75         allResults.sort((a, b) => b.max.max - a.max.max);
76         circle.matchValue = allResults[0].max.max;
77         circle.bestMatch = allResults[0].max.name;
78     }
79
80     console.log("Best match:");
81     console.dir(allResults[0]);
82
83     //set matchValue to 2 decimal places
84     circle.matchValue = Math.round(circle.matchValue * 100) / 100;
85
86     DrawCoinValue([circle], guiMat);
87
88     src.delete();
89 }

```

Lasst uns die Funktion im Detail betrachten:

Zunächst wird die Anzahl der Iterationen für die Rotation festgelegt. In meinem Fall sind es 180, was bedeutet, dass das Bild um 2 Grad pro Iteration gedreht wird und somit insgesamt 180 Bilder erstellt und verglichen werden. Da dies einen hohen Rechenaufwand bedeutet, wird vor jeder Iteration mit `await new Promise(r => setTimeout(r, 0))`; eine kurze Pause eingelegt, damit der Browser die GUI aktualisieren kann. Dafür muss die Funktion als `async` deklariert sein. Die Rotation erfolgt dann über meine vorher erstellte Funktion `RotateMatrix`, in welcher der Winkel für die Rotation übergeben wird.

Nachdem eine Matrix mit dem rotierten Bild erstellt wurde, wird diese mit allen Templates verglichen. Dafür wird über alle Templates iteriert und für jedes Template die Funktion `cv.matchTemplate` aufgerufen. Die Funktion gibt eine Matrix zurück, in der die Übereinstimmungswerte für das Template und das Bild enthalten sind. Diese Matrix wird dann nach dem höchsten und niedrigsten Wert durchsucht und in einem Array gespeichert.

Für die Speicherung der Ergebnisse wird ein Array `allResults` erstellt, welches Objekte der folgenden Klasse enthält:

```

1 class Result {
2     name;
3     min;
4     max;

```

```

5
6     constructor(name, min, max){
7         this.name = name;
8         this.min = min;
9         this.max = max;
10    }
11 }
12 }
```

Die Eigenschaften `min` und `max` enthalten dabei das Template mit der niedrigsten und höchsten Übereinstimmung. Da manche Vergleichsmethoden den niedrigsten Wert als besten Wert zurückgeben und andere den höchsten, werden somit beide Werte gespeichert. Die Eigenschaft `name` enthält in der inneren Schleife den Namen des Templates, welches die Übereinstimmung erzielt hat. In der äußeren Schleife wird der Name der Iteration gespeichert.

Innerhalb der inneren Schleife wird pro Vorlage ein Objekt dieser Klasse erstellt und in das Array `templateSimilarity` gespeichert. Dieses Array enthält somit für diesen Winkel alle Übereinstimmungswerte für alle Vorlagen. Nachdem alle Vorlagen für diesen einen Winkel verglichen wurden, wird das Array `templateSimilarity` nach dem Ergebnis mit dem niedrigsten und höchsten Wert sortiert und in Form eines neuen Ergebnis-Objektes in das Array `allResults` gespeichert. Danach iteriert die äußere Schleife weiter und das nächste Bild wird rotiert und verglichen.

Zum Schluss enthält das Array `allResults` ein Objekt für jede Iteration / jeden Winkel, welches die besten und schlechtesten Übereinstimmungswerte enthält. Je nach Vergleichsmethode wird das Array `allResults` dann nach dem höchsten oder niedrigsten Wert sortiert und der passendste Wert in das Attribut `circle.matchValue` zusammen mit dem Namen der Vorlage in `circle.bestMatch` gespeichert.

Nachdem die Vorlage mit der besten Übereinstimmung gefunden wurde, wird die GUI aktualisiert und das Ergebnis angezeigt. Um den Wert der Münze letztendlich anzuzeigen, wird die Funktion `DrawCoinValue` aufgerufen, welche den Wert der Münze direkt an die Position der Münze in der GUI einzeichnet. Die Funktion `DrawCoinValue` sieht wie folgt aus:

```

1 function DrawCoinValue(circles, dist){
2     if (circles === undefined || circles.length === 0) {
3         return;
4     }
5
6     for(let i = 0; i < circles.length; i++){
7         //does circle have "bestMatch" property?
8         if (circles[i].bestMatch === undefined) {
9             //print "?"
10            cv.putText(dist, "?", new cv.Point(circles[i].x, circles[i].y), cv.
11                         FONT_HERSHEY_SIMPLEX, 0.5, new cv.Scalar(255, 0, 255, 255), 1);
12        }else{
13            //print bestMatch
14            cv.putText(dist, circles[i].bestMatch, new cv.Point(circles[i].x-10,
15                         circles[i].y-6), cv.FONT_HERSHEY_SIMPLEX, 0.5, new cv.Scalar(255,
16                         255, 255, 255), 1);
17            cv.putText(dist, circles[i].matchValue.toString(), new cv.Point(circles
18                         [i].x-10, circles[i].y+6), cv.FONT_HERSHEY_SIMPLEX, 0.5, new cv.
19                         Scalar(255, 255, 255, 255), 1);
20        }
21    }
22 }
```

7.2 Finale Ergebnisse

Nun kommt der Moment der Wahrheit. Wie gut funktioniert die Münzerkennung? Um dies zu überprüfen, habe ich eine kleine Testreihe durchgeführt, bei der ich für 3 verschiedene Einstellungen je 5 Bilder aufgenommen habe. Die Einstellungen waren:

- **180 CCOEFF_NORMED** - 180 Iterationen, Vergleichsmethode CCOEFF_NORMED
- **180 SQDIFF_NORMED** - 180 Iterationen, Vergleichsmethode SQDIFF_NORMED
- **360 CCOEFF_NORMED** - 360 Iterationen, Vergleichsmethode CCOEFF_NORMED

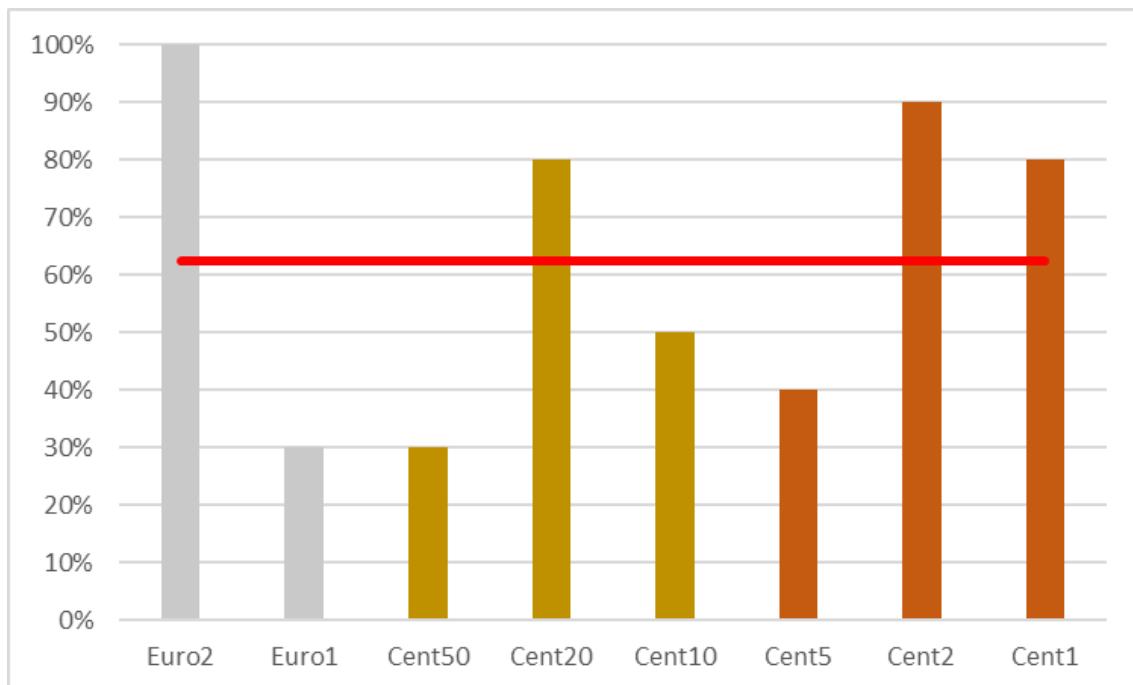
Nachdem ich für jede Einstellung 5 verschiedene Bilder aufgenommen und geprüft habe, bei denen alle 8 Münzen in zufälliger Rotation und Position auf einem Tisch lagen, bin ich zu folgendem Ergebnis gekommen:

- **180 CCOEFF_NORMED** - Durchschnittliche Erfolgsrate: 67,5%
- **180 SQDIFF_NORMED** - Durchschnittliche Erfolgsrate: 60,0%
- **360 CCOEFF_NORMED** - Durchschnittliche Erfolgsrate: 62,5%

Die Erfolgsrate ist dabei definiert als die Anzahl der Durchläufe, bei denen die Münze korrekt klassifiziert wurde, geteilt durch die Gesamtanzahl der Durchläufe.

Anschließend habe ich für die Einstellungen mit dem besten Ergebnis (180 CCOEFF_NORMED) eine weitere Testreihe mit 10 Bildern durchgeführt, um etwas aussagekräftigere Werte zu erhalten. Jedes Bild hatte wie in der vorherigen Testreihe wieder alle verfügbaren Euromünzen in einem zufälligen Winkel abgebildet. Die Ergebnisse sind in Abbildung 2 dargestellt:

Abbildung 2: Erfolgsrate der Münzerkennung



Wie man sofort sieht, gibt es starke Unterschiede in der Erfolgsrate der einzelnen Münzen. Während die 2-Euro Münze in fast allen Fällen korrekt erkannt wurde, lag die Erfolgsquote für die 1 Euro Münze

nur bei etwa 30%. Neben der 2-Euro Münze werden auch die 20-Cent, 2-Cent und 1-Cent Münzen in den meisten Fällen (>50%) korrekt erkannt. Dagegen werden die 1-Euro, 50-Cent und 5-Cent Münzen nur in etwa 30-40% der Fälle korrekt erkannt. Für die 10-Cent Münze liegt die Erfolgsrate bei etwa 50%. Insgesamt beträgt die durchschnittliche Erfolgsrate für alle Münzen somit 62,5%.

Alles in allem wieder recht durchwachsene Ergebnisse. Zwar konnte ich durch meine Änderungen die Erfolgsrate im Vergleich zu meinem letzten Blogbeitrag auf über 50% steigern, jedoch ist dies immer noch weit von einer zuverlässigen Münzerkennung entfernt. Die Kantenerkennung scheint dabei der richtige Weg gewesen zu sein, dennoch scheint sie noch nicht einwandfrei zu funktionieren. Die Kanten-Bilder der Vorlagen sehen nahezu perfekt aus, jedoch scheinen besonders die zu prüfenden Münzen nicht immer in ein verwertbares Kantenbild umgewandelt zu werden. Es hat sich als sehr schwierig erwiesen Parameter zu finden, welche für alle Münzen ein gutes Ergebnis liefern, sodass die Konturen der Ziffer klar erkennbar sind, jedoch nicht zu viele Störungen vorhanden sind.

Ich denke jedoch, dass mich dennoch beweisen konnte, dass mittels Template-Matching und Open-CV.js eine Münzerkennung im Browser möglich ist. Die Ergebnisse könnten sicherlich mit etwas weiterem Feintuning verbessert werden, jedoch würde dies den Rahmen dieses Blogs sprengen. Jener Blog neigt sich nun auch dem Ende zu, daher möchte ich im nächsten Abschnitt noch ein Fazit ziehen und einen Ausblick auf mögliche Weiterentwicklungen geben.

8 Fazit

Das war mein Blog über den CoinFinder! Ich hoffe, dass ich dir einen guten Einblick in die Möglichkeiten von OpenCV.js geben konnte und dass du jetzt ein besseres Verständnis dafür hast, wie Bildverarbeitungsalgorithmen im Browser implementiert werden können. In diesem Abschnitt werde ich noch einmal kurz zusammenfassen, was für mich die größten Probleme und Herausforderungen bei der Implementierung waren und einen Ausblick darauf geben, wie das Projekt in Zukunft weiterentwickelt werden könnte. Vielen Dank schonmal fürs Lesen!

8.1 Die größten Probleme

Es hat sich gezeigt, dass mit openCV sehr anspruchsvolle Probleme in der Bildverarbeitung gelöst werden können. Je nach Umfang und Komplexität des Problems kann die Implementierung jedoch sehr aufwendig sein. Die größten Probleme, die ich während der Implementierung hatte, waren die folgenden:

- **Dokumentation** - Speziell für OpenCV.js ist die Dokumentation sehr spärlich. Die offizielle Dokumentation ist nur für C++ verfügbar und die JavaScript-Tutorials sind oft nicht ausreichend. Zudem erschwert die Emscripten-Übersetzung den Einstieg, da in der IDE keine Autovervollständigung für die OpenCV-Funktionen verfügbar ist. Es gibt keine leichte Möglichkeit, alle Funktionen und Parameter in Erfahrung zu bringen, was die Implementierung erschwert.
- **Umgebungsbedingungen** - Viele Methoden der Bilderkennung haben sich als sehr empfindlich gegenüber den Umgebungsbedingungen herausgestellt. So haben beispielsweise die Lichtverhältnisse und die Kameraqualität einen großen Einfluss auf die Ergebnisse. Vor allem das Licht stellt ein nur schwer zu lösendes Problem dar, welches nur schwer "genormt" werden kann.
- **Parameter** - Die Parameter der OpenCV-Funktionen sind oft sehr spezifisch und müssen genau auf den Anwendungsfall abgestimmt werden. So kann es sein, dass eine Funktion bei falscher Einstellung gar nicht oder nur sehr schlechte Ergebnisse liefert. Das Finden der richtigen Parameter hat sich als sehr zeitaufwendig herausgestellt.

Ganz konkret war beispielsweise die Kantenerkennung sehr anfällig gegenüber äußeren Verhältnissen. So musste ich für die Kantenerkennung nicht einmal, sondern gleich zweimal korrekte Parameter ermitteln, da die Vorlagenbilder für die Kantenerkennung bei unterschiedlichen Lichtverhältnissen und Auflösungen aufgenommen wurden. Die Bilder, welche live aus der Kamera kommen, haben eine deutlich schlechtere Qualität und Auflösung, wodurch die Parameter nicht mehr passten.

Trotz der Slider, die ich genau für diesen Fall eingebaut habe, war es leider sehr zeitaufwendig alles aufeinander abzustimmen. Da der CoinFinder wie eine "Pipeline" funktioniert, beeinflusst ein ungenaueres Zwischenergebnis auch die Ergebnisse der folgenden Schritte. So kann es sein, dass ein ungenau erkannter Kreis oder ein ungünstig erstelltes Kantenbild auch die Ergebnisse des Template-Matchings beeinflusst.

8.2 Ausblick

Ich hoffe mit diesem Blog einen Einblick in die Möglichkeiten von OpenCV.js gegeben zu haben. Die Implementierung von Bildverarbeitungsalgorithmen im Browser ist eine spannende Entwicklung, die in Zukunft sicherlich noch an Bedeutung gewinnen wird. Selbst wenn die letztendlichen Ergebnisse des CoinFinders nicht perfekt sind, so zeigt das Projekt doch sehr gut, für welche verschiedene Anwendungsfälle OpenCV.js genutzt werden kann.

Man darf nicht vergessen, dass die Kreiserkennung und das Template-Matching, wie ich es in diesem Blog implementiert habe, nur ein kleiner Teil der Möglichkeiten von OpenCV sind. OpenCV bietet eine Vielzahl von weiteren Funktionen, die womöglich auch für das Problem des CoinFinders verwendet werden könnten. So gibt es in OpenCV weitere Möglichkeiten der Objekterkennung, welche mithilfe von Deep Neural Networks (DNN) realisiert werden. Diese sind natürlich deutlich komplizierter und aufwendiger zu implementieren als die Methoden die ich in diesem Blog vorgestellt habe, sie könnten jedoch auch deutlich bessere Ergebnisse liefern. Dies wäre eine spannende Weiterentwicklung des Projekts, welche jedoch den Scope dieses Blogs übersteigen würde. Aber vielleicht ist das ja ein Thema für einen weiteren Blog?

Lasst mich gerne wissen, was ihr von dem Blog haltet und ob ihr euch eine Fortsetzung wünscht. Ich freue mich über Feedback und Anregungen!