

# Dokumentation: Bewegungserkennung in statischen Videosequenzen mit OpenCV

## 1. Zielsetzung

Das Ziel dieses Projekts ist die Entwicklung eines Algorithmus zur Detektion von bewegten Objekten in einer statischen Videoszene. Dabei werden die ersten Frames des Videos zur Modellierung des Hintergrunds genutzt, indem die Farben gemittelt werden. Abweichungen vom erwarteten Hintergrund werden als Bewegung interpretiert. Zusätzlich wird eine Heatmap der Bewegungsintensität berechnet und verschiedene Verfahren zur Hintergrundsubtraktion verglichen.

## 2. Implementierungsansatz

Die Implementierung erfolgt mit Python und OpenCV. Es werden zwei verschiedene Methoden zur Hintergrundsubtraktion verwendet:

- **Eigenes Modell:** Berechnung eines gleitenden Durchschnitts zur Modellierung des Hintergrunds.
- **OpenCV-Methoden:** Verwendung von `cv2.createBackgroundSubtractorMOG2()` und `cv2.createBackgroundSubtractorKNN()`.

Zudem wird eine Heatmap erstellt, um die kumulierte Bewegungsintensität über die gesamte Videosequenz zu visualisieren.

## 3. Technische Umsetzung

### 3.1 Verarbeitung der Frames

- Für jeden Frame:
  - Berechnung eines Durchschnittsbildes
  - Berechnung der Differenz zum aktuellen Frame
  - Erstellung einer binären Bewegungsmaske mit Thresholding und Morphologischen Operationen
  - Overlay der Bewegungserkennung auf das Originalbild
  - Aktualisierung der kumulierten Bewegungsintensität (Gleitkommawert)
  - Die Heatmap wird basierend auf der kumulierten Bewegungsintensität berechnet und normalisiert.
    - Mappen des Werts zwischen 0 und 255
    - Einfärben für das Overlay
  - Bildgenerierung durch gewichtetes Hinzufügen der Heatmap zum Originalbild

### 3.2 Implementierung Bewegungserkennung

Die Segmentierung erfolgt durch Differenzbildung mit einem durchschnittlichen Hintergrundbild und nachfolgende Rauschreduzierung:

```
# calculate average frame
avgFrame = cumulatedFrame / (frameCount, frameCount, frameCount)
avgFrame = avgFrame.astype(np.uint8)

# get the difference between the current frame and the average frame
diffFrame = cv2.absdiff(frameCopy, avgFrame)

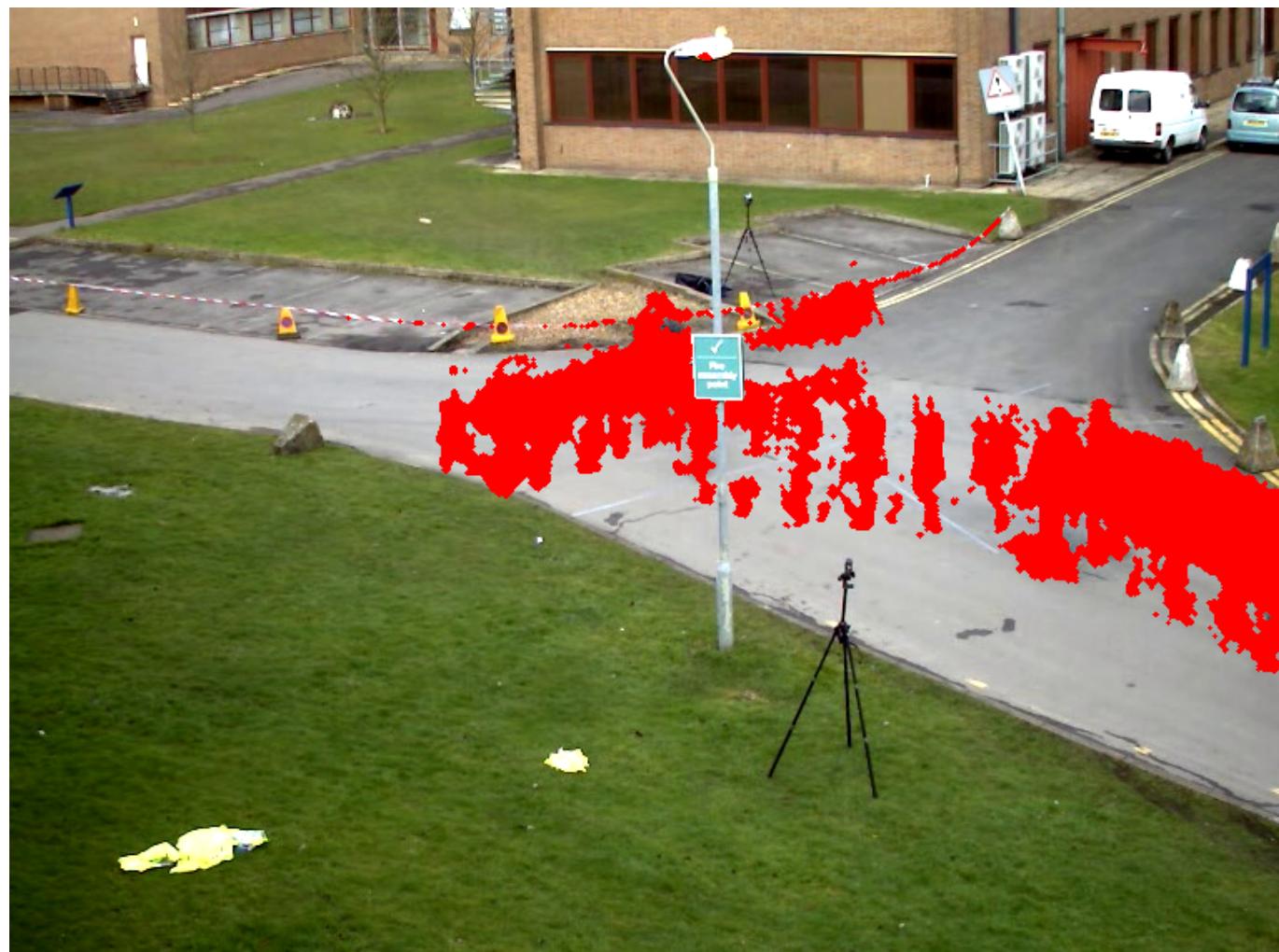
# create a motion mask by defining a threshold for the difference frame
_, motionMask = cv2.threshold(diffFrame, 50, 255, cv2.THRESH_BINARY)

# create and apply filter to reduce noise
#kernel = np.ones((3, 3), np.uint8)
# filter kernel
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
# remove single pixel artefacts
motionMask = cv2.morphologyEx(motionMask, cv2.MORPH_OPEN, kernel)
# close created gaps
motionMask = cv2.morphologyEx(motionMask, cv2.MORPH_CLOSE, kernel)

# add the motion mask as red to the original frame
movementOverlay = frame.copy()
movementOverlay[motionMask[..., 0] > 0] = [0, 0, 255]
```

### 3.3 Tests mit verschiedenen Parametern:

- **Schwellwert (threshold):** 10, 50, 75, 100 → beste Ergebnisse bei **50**
- **Morphologische Filter:** Rechteck 3×3, Ellipse 2×2, Ellipse 3×3 → **Ellipse 3×3 beste Ergebnisse**

**Difference Frame Threshold: 10 - 255**

**Difference Frame Threshold: 50 - 255**

**Difference Frame Threshold: 75 - 255**

**Difference Frame Threshold: 100 - 255**

**Noise Filter: Rectangle 3x3**

**Noise Filter: Ellipse 2x2**

### Bestes Ergebnis: Difference Frame Threshold(50 - 255), Noise Filter(Ellipse 2x2)



#### Erkenntnisse:

- **Geringe Threshold-Werte** (<50) führen zu vielen Falschpositiven, höhere Werte (>75) unterdrücken kleinere Bewegungen.
- **Morphologische Öffnung** entfernt kleine Störpixel, **Schließung** schließt Lücken in den detektierten Bereichen.
- Eine **elliptische Kernel-Form** liefert bessere Ergebnisse als rechteckige Strukturen.

#### 3.4 Implementierung Heatmap

Die Heatmap wird durch Akkumulation der Bewegungsmaske über die gesamte Videosequenz erstellt:

```
# add up the motion mask for the heatmap
cumulativeMovement += motionMask[..., 0].astype(np.float32)
normCumulativeMovement = cv2.normalize(cumulativeMovement, None, 0, 255,
cv2.NORM_MINMAX).astype(np.uint8)

# apply filter to the heatmap overlay
#kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (10, 10)) # filter
kernel
```

```
# remove clearly visible outlines
normCumulativeMovement = cv2.morphologyEx(normCumulativeMovement,
cv2.MORPH_DILATE, kernel)
# remove single pixel artefacts
#normCumulativeMovement = cv2.morphologyEx(normCumulativeMovement,
cv2.MORPH_OPEN, kernel)
# close created gaps
#normCumulativeMovement = cv2.morphologyEx(normCumulativeMovement,
cv2.MORPH_CLOSE, kernel)

heatmapColor = cv2.applyColorMap(normCumulativeMovement, cv2.COLORMAP_HOT)
heatmapOverlay = cv2.addWeighted(frameCopy, 1, heatmapColor, 0.7, 0)
```

### 3.5 Tests mit verschiedenen Parametern:

- **Filterung:** Ohne Filter, mit Morph-Operationen (Dilatation, Öffnung, Schließung)
- **Gewichtung der Überlagerung:** Verschiedene Werte getestet (z. B. 0.5, 0.7, 1.0)

**Bestes Ergebnis: Mit 'Dilate' Filter und gewichtetem Overlay zum Originalbild**



### Erkenntnisse:

- Die **Dilatation** hilft, scharfe Kanten zu vermeiden und die Heatmap flüssiger wirken zu lassen.
- Eine Überlagerung mit **0.7** sorgt für eine gute Balance zwischen Sichtbarkeit und Detailtreue.

## 4. Hintergrundentfernung mit OpenCV Funktionen

### 4.1 Implementierung

Zur Testung der Hintergrundentfernung und Erzeugung eines Ergebnisbildes wurde eine Funktion entwickelt, die eine übergebene OpenCV-Background-Subtractor-Klasse verwendet. Dabei werden entsprechende Masken erstellt und nachfolgend gefiltert.

```
fgbgMog2 = cv2.createBackgroundSubtractorMOG2()
fgbgKnn = cv2.createBackgroundSubtractorKNN()

def applyOpenCvBackgroundSubtraction(cframe, bgSubtractor):
    # test background subtraction with OpenCV functions
    fgMask = bgSubtractor.apply(cframe)
    _, motionMask = cv2.threshold(fgMask, 50, 255, cv2.THRESH_BINARY)
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
    motionMask = cv2.morphologyEx(motionMask, cv2.MORPH_OPEN, kernel)
    motionMask = cv2.morphologyEx(motionMask, cv2.MORPH_CLOSE, kernel)
    movementOverlay = cframe.copy()
    movementOverlay[motionMask > 0] = [0, 0, 255]
    return movementOverlay

def createHeatmap(cumulativeMovement):
    normCumulativeMovement = cv2.normalize(cumulativeMovement, None, 0,
    255, cv2.NORM_MINMAX).astype(np.uint8)
    normCumulativeMovement = cv2.morphologyEx(normCumulativeMovement,
    cv2.MORPH_DILATE, kernel)
    heatmapColor = cv2.applyColorMap(normCumulativeMovement,
    cv2.COLORMAP_HOT)
    heatmapOverlay = cv2.addWeighted(frameCopy, 1, heatmapColor, 0.7, 0)
    return heatmapOverlay

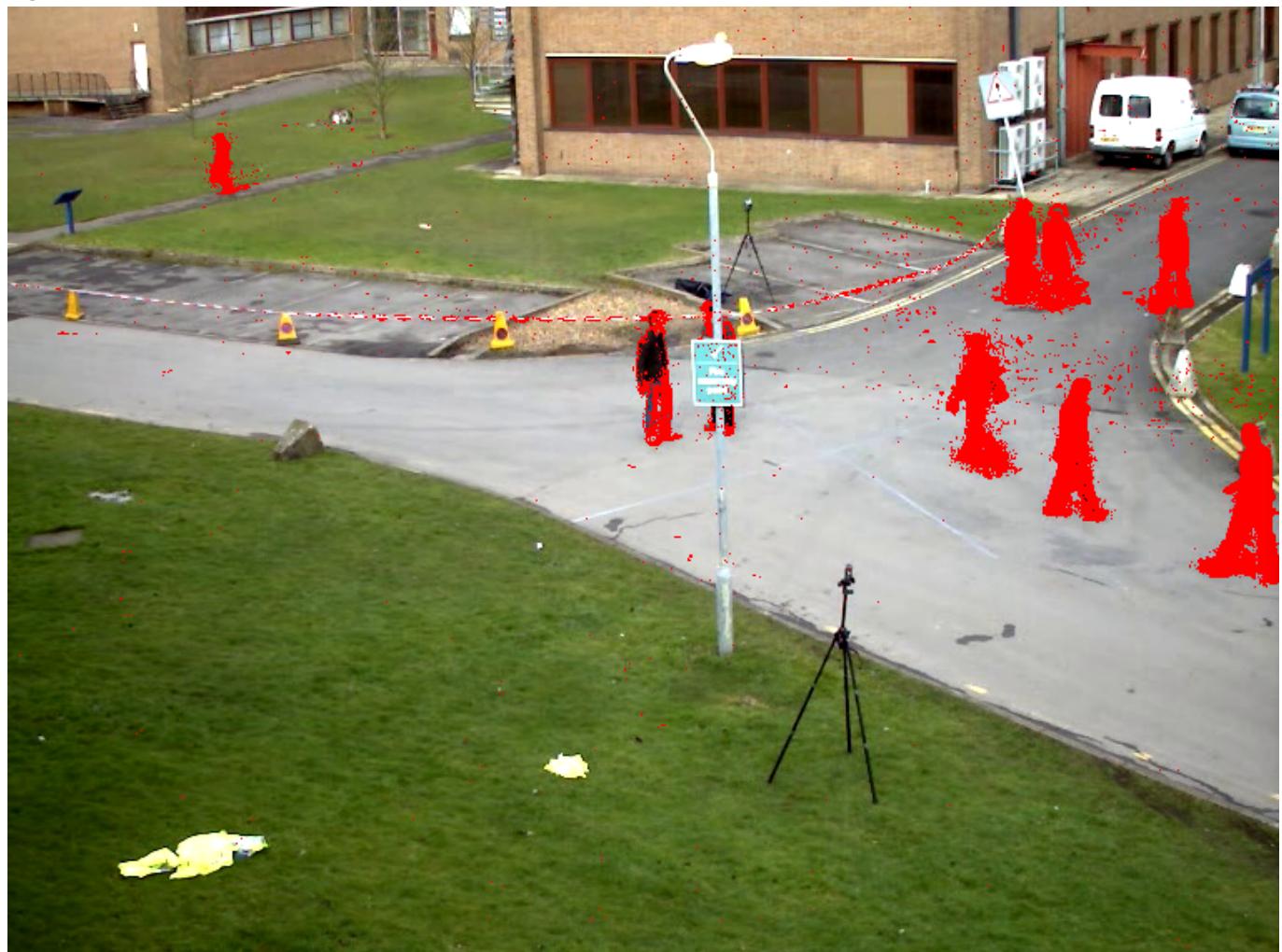
...
movementMog2, motionMaskMog2 = applyOpenCvBackgroundSubtraction(frameCopy,
fgbgMog2)
movementKnn, motionMaskKnn = applyOpenCvBackgroundSubtraction(frameCopy,
fgbgKnn)
cumulativeMovementMog2 += motionMaskMog2.astype(np.float32)
cumulativeMovementKnn += motionMaskKnn.astype(np.float32)
heatmapOverlayMog2 = createHeatmap(cumulativeMovementMog2)
heatmapOverlayKnn = createHeatmap(cumulativeMovementKnn)
...
```

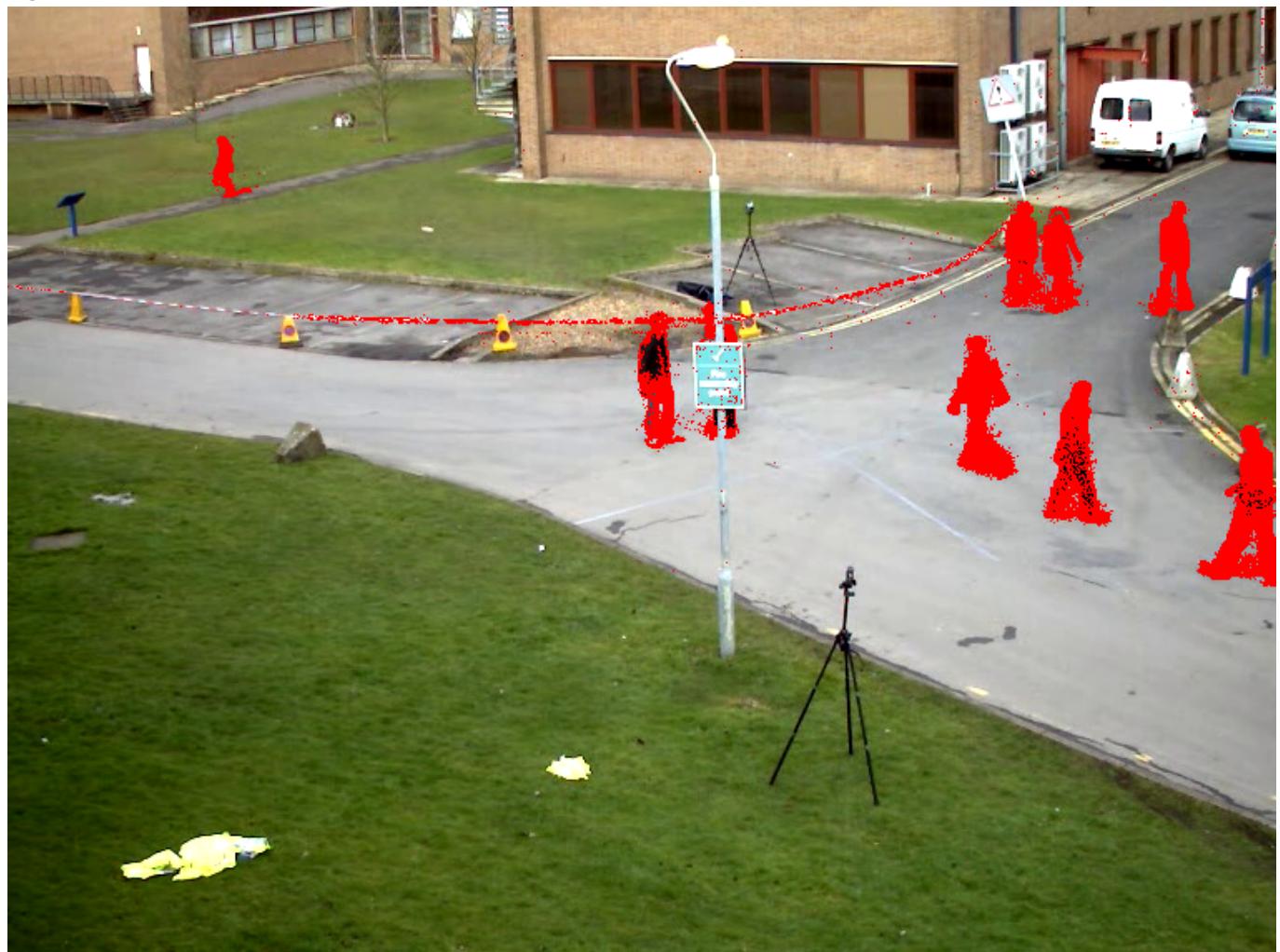
## 4.2 Testen der OpenCV Hintergrundentfernung

Die Algorithmen wurden anhand der Bewegungserkennung und der Heatmap getestet. Dabei wurde geprüft, wie gut die verschiedenen Verfahren Bewegungen vom statischen Hintergrund trennen und wie sich unterschiedliche Schwellenwerte und Filter auf das Ergebnis auswirken.

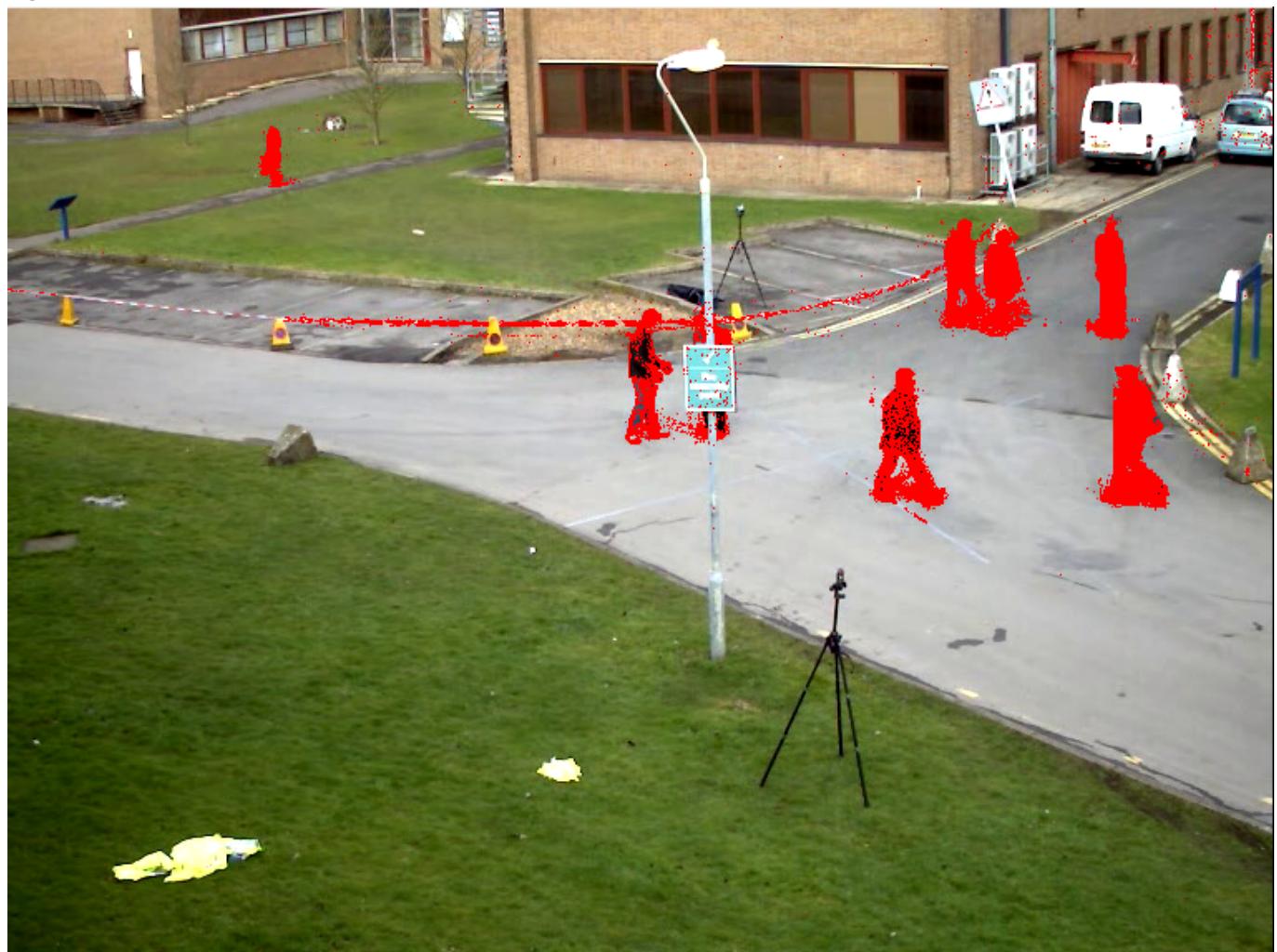
**Difference Frame Threshold: 30 - 255**

**OpenCV MOG2:**



**OpenCV KNN:**

**Difference Frame Threshold: 50 - 255****OpenCV MOG2:**

**OpenCV KNN:**

**Difference Frame Threshold: 50 - 255 and Erosion + Dilation Filter**

Um das Rauschen weiter zu minimieren, wurden Erosion- und Dilation-Filter angewendet.

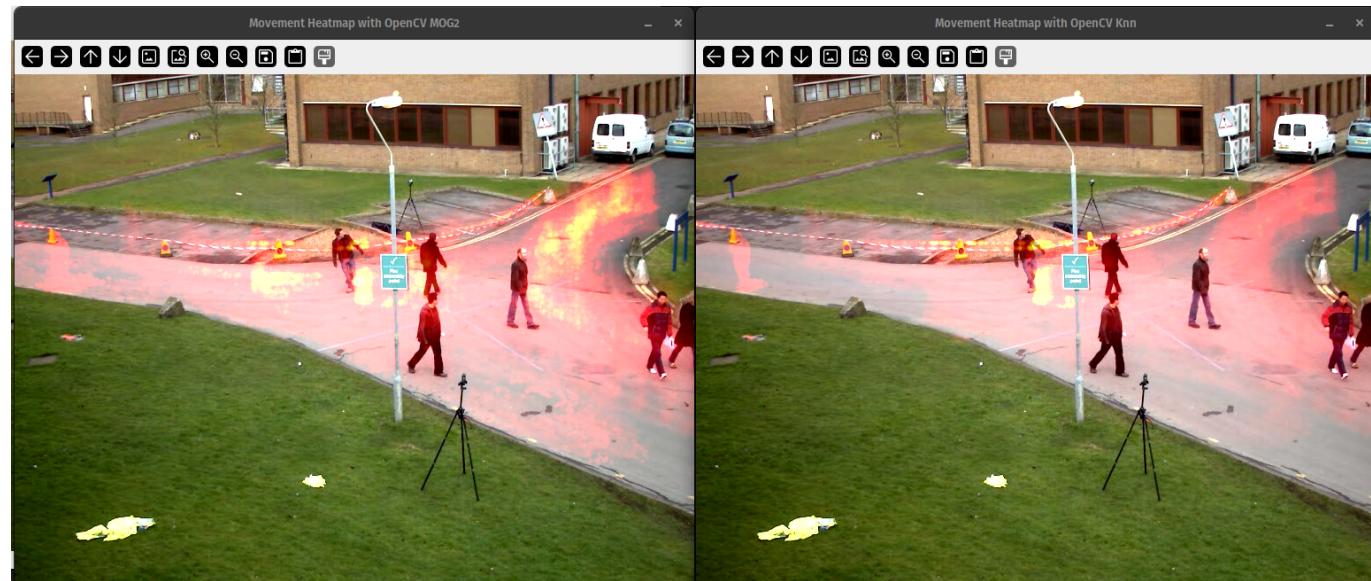
**OpenCV MOG2:**



**OpenCV KNN:**

## Testen der Unterschiede bei der Heatmap mit den Parametern: Difference Frame Threshold: 50 - 255 and Erosion + Dilection Filter

OpenCV MOG2 (links) | OpenCV KNN (rechts)



### Erkenntnisse:

- KNN liefert insgesamt bessere Ergebnisse als MOG2, da es weniger Artefakte erzeugt und den Hintergrund sauberer trennt.
- MOG2 hat insgesamt mehr Rauschen, das durch zusätzliche Filter entfernt werden muss.
- Beide OpenCV-Methoden übertreffen die eigene Implementierung der Hintergrundsubtraktion in Präzision und Effizienz.
- Durch geschickte Wahl von Thresholds und Filtern kann das Ergebnis weiter optimiert werden.
- Die Heatmap zeigt, dass die Rotfärbung bei der Bewegungserkennung ('Hitze') bei MOG2 deutlich länger anhält als bei KNN – trotz identischer Parametereinstellungen. Abgesehen davon liefern beide Verfahren vergleichbare Ergebnisse. Die Dauer der 'Hitze'-Anzeige ist letztlich eine Frage der Präferenz und abhängig vom gewünschten Verhalten der Bewegungserkennung.

## 5. Testen des eigenen Videos

Für das Testvideo wurde zunächst einige Sekunden lang ein Tisch ohne Bewegung gefilmt. Dies sollte der Mittelwertberechnung für die Hintergrunderkennung helfen. Nach dieser Kalibrierungsphase wurden mehrere Karten auf den Tisch geworfen, um die Bewegungserkennung zu testen. Zusätzlich wurden die Ergebnisse der Heatmap analysiert.

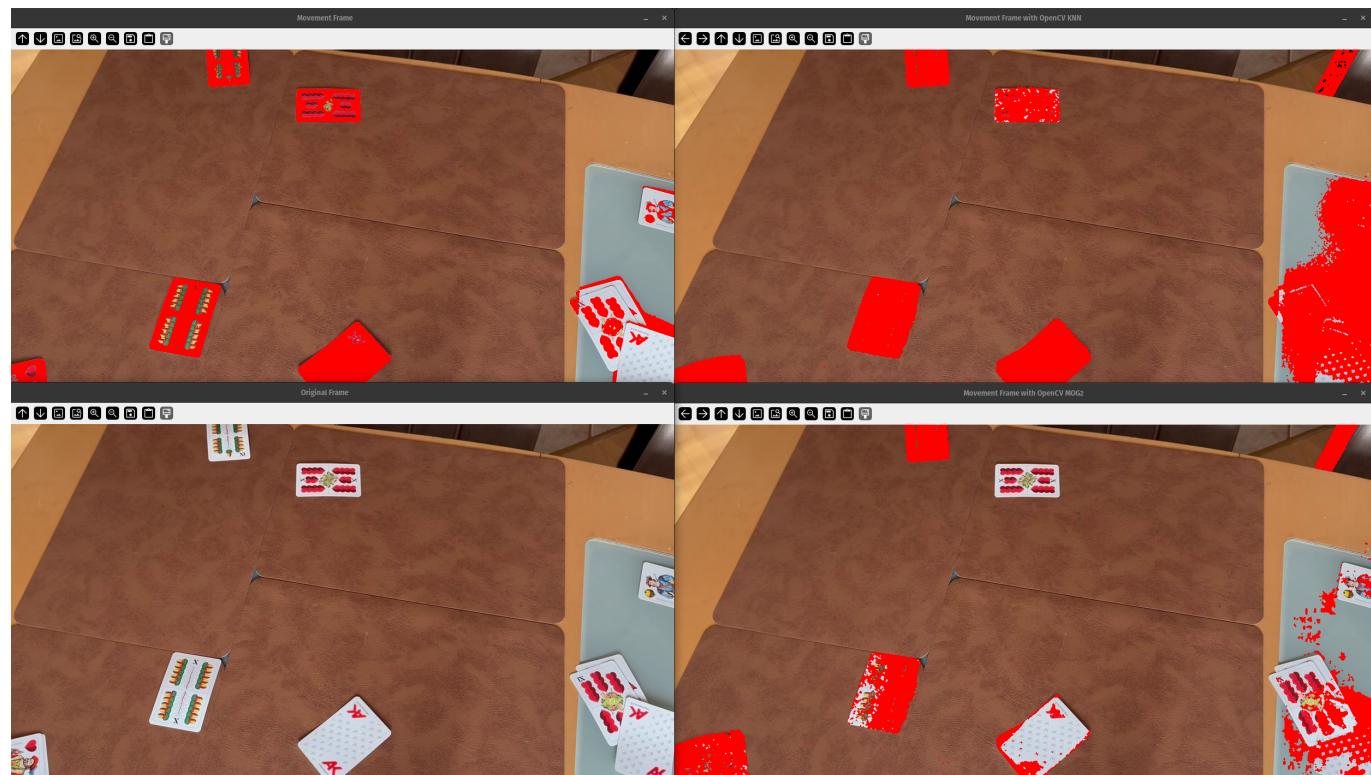
Die folgenden Bilder sind wie folgt angeordnet:

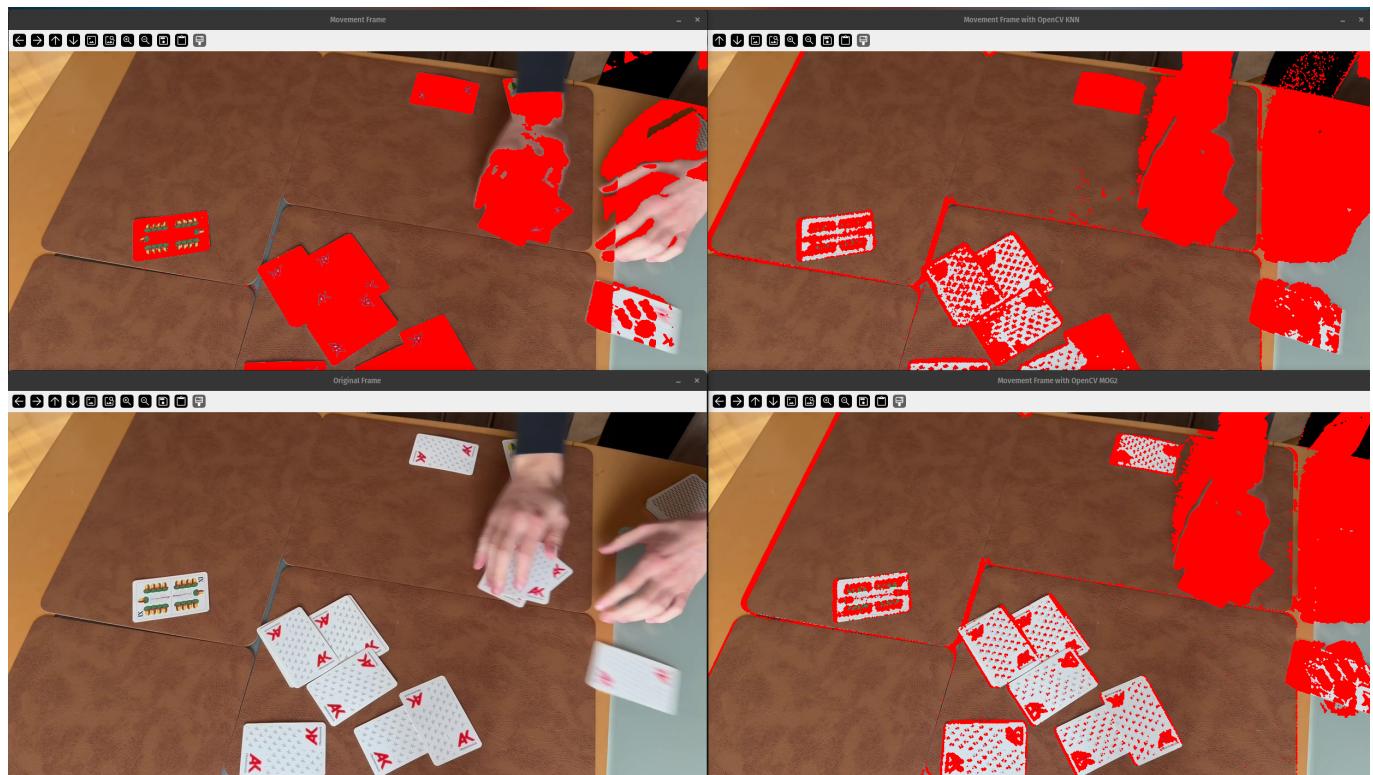
- Links Oben: Eigene Implementierung
- Rechts Oben: OpenCV KNN
- Links Unten: Originalbild
- Rechts Unten: OpenCV MOG2

### Bewegungserkennung der Karten

Im statischen Bild ohne jegliche Bewegung zeigte sich sofort, dass sich die OpenCV-Algorithmen **MOG2** und **KNN** mit der Glasfläche des Tisches und den auftretenden Schatten schwer taten. Dadurch kam es zu Artefakten in der Hintergrunderkennung.

Die eigens entwickelte Methode zur Hintergrunderkennung lieferte in diesem Fall stabilere Ergebnisse. Nach dem Einwerfen der Karten wurden jedoch alle Karten von den Algorithmen zuverlässig erkannt.

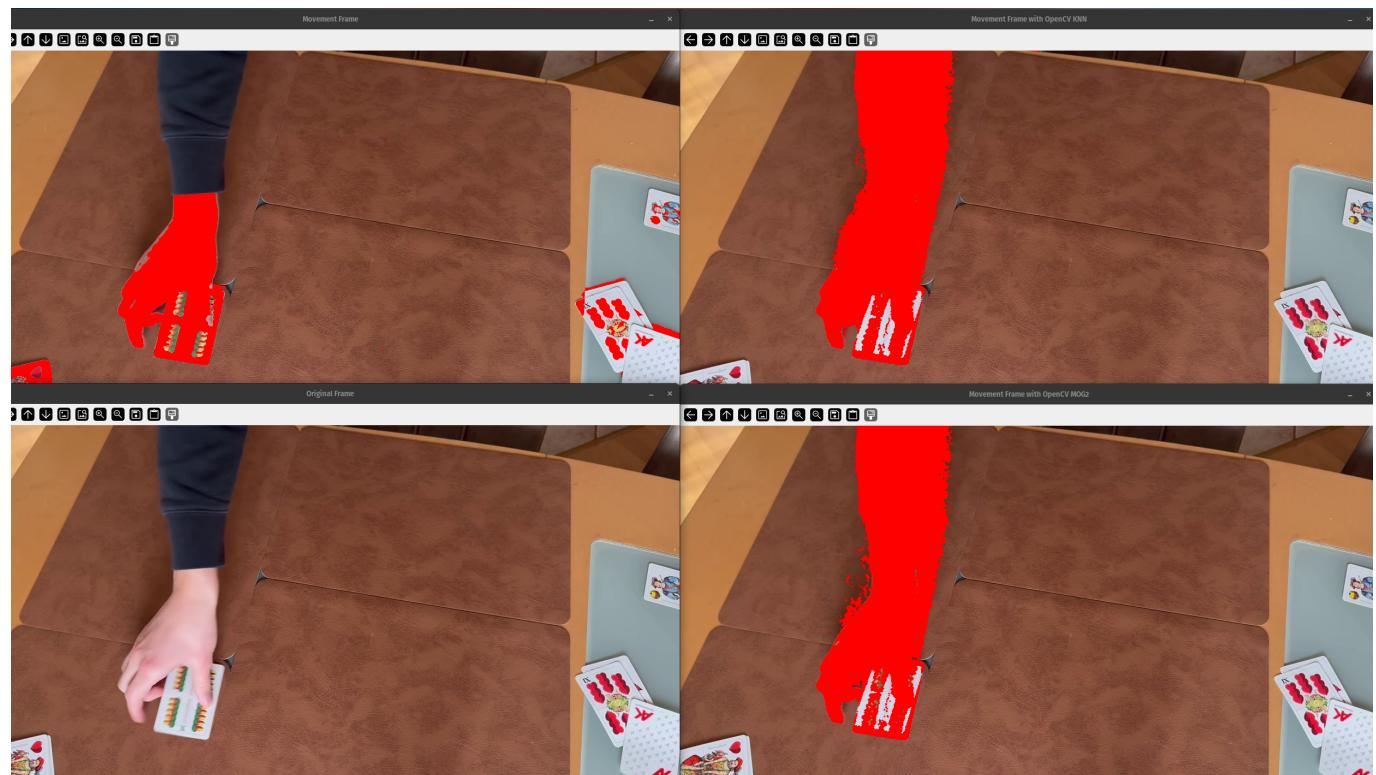




## Bewegungserkennung der Karten und der Hand

Bei der Bewegung einer Hand im Bild zeigte sich ein umgekehrtes Verhalten:

Die OpenCV-Algorithmen **MOG2** und **KNN** erkannten die Handbewegung besser als die eigene Entwicklung. Der Ärmel der Hand von der eigenen Implementierung nicht korrekt detektiert, dafür gibt es nicht soviele Artefakte.



## Bewegungserkennung der Karten nach Ruhephase

Nach einer gewissen Ruhephase, in der keine Bewegung stattfand, zeigten sich deutliche Unterschiede zwischen den Algorithmen:

- Die **eigene Entwicklung** erkannte die Karten weiterhin korrekt.
- Die OpenCV-Algorithmen **MOG2** und **KNN** hingegen zählten die Karten nach der Ruhephase zum Hintergrund, was zu einem Informationsverlust führte.



## Heatmap der Bewegungserkennung

Da die OpenCV-Algorithmen mit Schatten und der Reflexion des Glases Schwierigkeiten hatten, war die daraus resultierende Heatmap nur eingeschränkt aussagekräftig.

Die eigene Entwicklung schnitt hier etwas besser ab:

- Die Heatmap zeigte klar, wo genau die Bewegung aufgetreten war.
- Dies erleichtert die spätere Analyse und Optimierung der Erkennung.

