



**Politechnika  
Śląska**

21 stycznia 2025,

Gliwice

## **Laboratorium ZAOWR**

Rok akademicki	Termin	Rodzaj studiów	Kierunek	Prowadzący	Grupa
2024/2025	Czwartek, 10:00-13:15	Stacjonarne	Informatyka	Dr inż. Marcin Paszkuta Mgr inż. Mateusz Płonka	IGT

Ćwiczenie nr 5

Temat: Przepływ optyczny

Informatyka, IGT

Maksymilian Kisiel

# **Spis treści**

<b>1 Zadanie 1 - Rzadki przepływ optyczny</b>	<b>2</b>
<b>2 Zadanie 2 - Gęsty przepływ optyczny</b>	<b>3</b>
<b>3 Zadanie 3 - Wykrywanie ruchomych obiektów</b>	<b>4</b>
3.1 Sparse . . . . .	4
3.2 Dense . . . . .	6
<b>4 Zadanie 4 - Pomiar czasu i parametryzacja detekcji</b>	<b>8</b>
4.1 Sparse . . . . .	8
4.1.1 Średnie czasy analizy pojedynczej klatki . . . . .	9
4.2 Dense . . . . .	10
4.2.1 Średnie czasy analizy pojedynczej klatki . . . . .	11
<b>5 Kod</b>	<b>12</b>

# 1 Zadanie 1 - Rzadki przepływ optyczny

Aby wykonać to zadanie, zmodyfikowano dostarczony przez prowadzącego fragment kodu tak, aby obsługiwał różne rodzaje danych wejściowych (**live feed, seria zdjęć, wideo z dysku**) i dodano opcję skalowania obrazu. Dla zadań 1 oraz 2 nie zamieszczano kodu w sprawozdaniu. Pełny kod programu i funkcji wykonujących poszczególne zadania został dołączony do sprawozdania, a także jest dostępny w [serwisie GitHub](#).



Obraz z naniesionymi punktami



Obraz z liniami przepływu optycznego

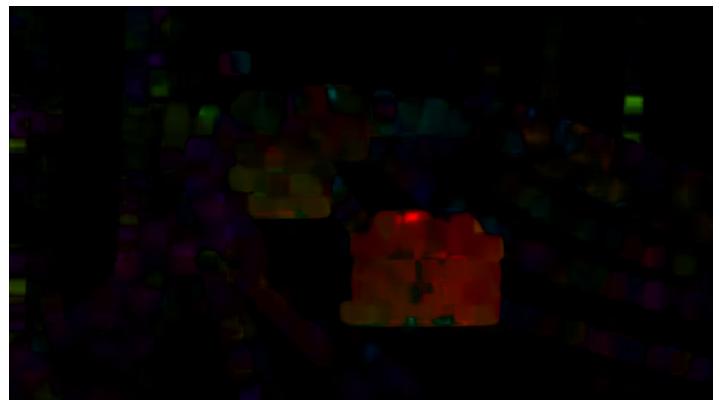
Rys. 1: Otrzymana klatka z wyznaczonym przepływem optycznym, **Sparse Optical Flow**

## 2 Zadanie 2 - Gęsty przepływ optyczny

W tym zadaniu ponownie wykorzystano dostarczony przez prowadzącego kod i dodano nieznaczące poprawki (obsługa wielu typów danych wejściowych).



Obraz oryginalny



Wyznaczony przepływ optyczny

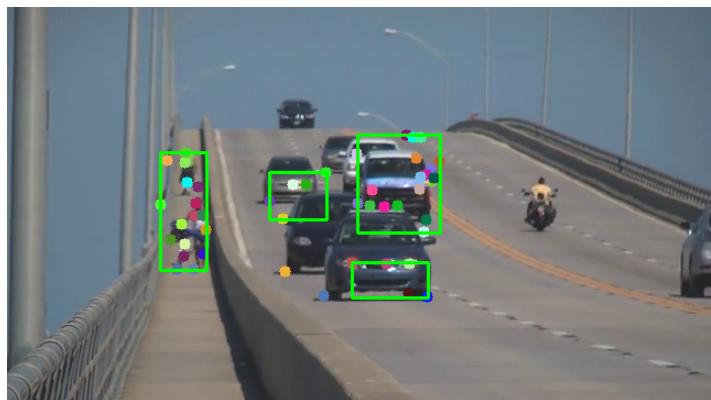
Rys. 2: Otrzymana klatka z wyznaczonym przepływem optycznym, **Dense Optical Flow**

### 3 Zadanie 3 - Wykrywanie ruchomych obiektów

W tym zadaniu zmodyfikowano znacząco kod (Kod 1 i 2). Dodane zostały dwie metody wyznaczania obwiedni (DBSCAN oraz progowanie) oraz dodano wyznaczanie i rysowanie prędkości i przemieszczania wykrytego obiektu.

#### 3.1 Sparse

Poniżej znajdują się porównania DBSCAN oraz progowania (Rys. 3 i 4). Zmodyfikowany fragment kodu znajduje się na kolejnej stronie (Kod 1).



Obraz oryginalny



Wyznaczony przepływ optyczny

Rys. 3: Otrzymana klatka z wyznaczonymi prędkościami i kierunkami za pomocą klasteryzacji DBSCAN, Sparse Optical Flow



Obraz oryginalny



Wyznaczony przepływ optyczny

Rys. 4: Otrzymana klatka z wyznaczonymi prędkościami i kierunkami za pomocą progowania, Sparse Optical Flow

```

if drawBoxes and len(goodNew) > 0:
    if bboxMethod == "dbSCAN":
        from sklearn.cluster import DBSCAN
        if len(goodNew) > 0:
            clustering = DBSCAN(eps=clusteringEps, min_samples=minClusterSize,
metric=clusteringMethod, n_jobs=-1).fit(goodNew)
            labels = clustering.labels_
            uniqueLabels = set(labels)
            for label in uniqueLabels:
                if label == -1:
                    continue
                clusterPoints = goodNew[labels == label]
                clusterOldPoints = goodOld[labels == label]
                x, y, w, h = cv.boundingRect(clusterPoints)
                x, y, w, h = [int(v / scaleFactor) for v in (x, y, w, h)]
                motionVectors = (clusterPoints - clusterOldPoints) / scaleFactor
                avgMotion = np.mean(motionVectors, axis=0)
                speed = np.linalg.norm(avgMotion)
                directionAngle = np.arctan2(avgMotion[1], avgMotion[0]) * 180 / np.pi
                cv.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
                startPoint = (int(x + w / 2), int(y + h / 2))
                endPoint = (int(startPoint[0] + avgMotion[0] * 20), int(startPoint[1]
+ avgMotion[1] * 20))
                cv.arrowedLine(frame, startPoint, endPoint, (0, 0, 255), 2,
tipLength=0.5)
                txtColor = (0, 255, 0)
                speedText = f"Speed: {speed:.2f} px/frame"
                directionText = f"Dir: {directionAngle:.1f} deg"
                cv.putText(frame, speedText, (x, y - 20), cv.FONT_HERSHEY_SIMPLEX,
0.7, txtColor, 1)
                cv.putText(frame, directionText, (x, y - 40),
cv.FONT_HERSHEY_SIMPLEX, 0.7, txtColor, 1)
    elif bboxMethod == "threshold":
        motionVectors = goodNew - goodOld
        motionMagnitude = np.sqrt((motionVectors[:, 0] ** 2) + (motionVectors[:, 1]
** 2))
        motionMagnitude = cv.normalize(motionMagnitude, None, 0, 255,
cv.NORM_MINMAX).astype(np.uint8)
        motionMask = np.zeros(frameGray.shape, dtype=np.uint8)
        for (x, y), magnitude in zip(goodNew, motionMagnitude):
            x, y = int(x), int(y)
            if 0 <= y < motionMask.shape[0] and 0 <= x < motionMask.shape[1]:
                motionMask[y, x] = 255
        kernel = np.ones((5, 5), np.uint8)
        motionMask = cv.dilate(motionMask, kernel, iterations=2)
        contours, _ = cv.findContours(motionMask, cv.RETR_EXTERNAL,
cv.CHAIN_APPROX_SIMPLE)
        for cnt in contours:
            if cv.contourArea(cnt) > (frameGray.shape[1]) / 10:
                x, y, w, h = cv.boundingRect(cnt)
                x, y, w, h = [int(v / scaleFactor) for v in (x, y, w, h)]
                tmpMask = np.zeros_like(motionMask, dtype=np.uint8)
                cv.drawContours(tmpMask, [cnt], -1, color=255, thickness=-1)
                motionVectorsInContour = []
                for i, point in enumerate(goodNew):
                    px, py = point.ravel()
                    if cv.pointPolygonTest(cnt, (px, py), False) >= 0:
                        motionVectorsInContour.append(motionVectors[i])
                motionVectorsInContour = np.array(motionVectorsInContour)
                if motionVectorsInContour.size > 0:
                    avgMotion = np.mean(motionVectorsInContour, axis=0)

                    speed = np.linalg.norm(avgMotion)
                    directionAngle = np.arctan2(avgMotion[1], avgMotion[0]) * 180 /
np.pi
                else:
                    avgMotion = [0, 0]
                    speed = 0
                    directionAngle = 0
                cv.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
                startPoint = (int(x + w / 2), int(y + h / 2))
                endPoint = (int(startPoint[0] + avgMotion[0] * 20), int(startPoint[1]
+ avgMotion[1] * 20))
                cv.arrowedLine(frame, startPoint, endPoint, (0, 0, 255), 2,
tipLength=0.5)
                txtColor = (0, 255, 0)
                speedText = f"Speed: {speed:.2f} px/frame"
                directionText = f"Dir: {directionAngle:.1f} deg"
                cv.putText(frame, speedText, (x, y - 20), cv.FONT_HERSHEY_SIMPLEX,
0.7, txtColor, 1)
                cv.putText(frame, directionText, (x, y - 40),
cv.FONT_HERSHEY_SIMPLEX, 0.7, txtColor, 1)

```

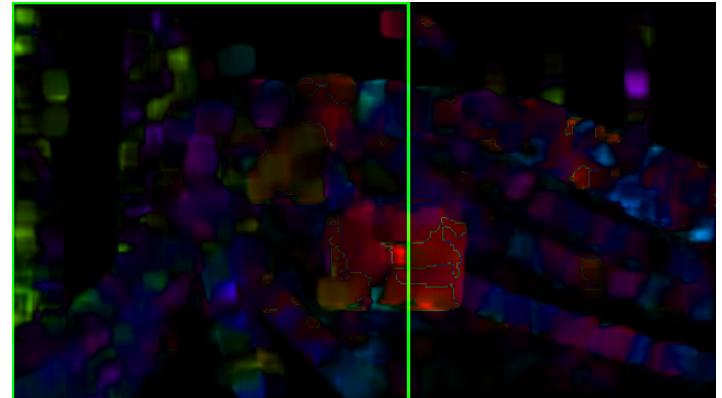
Kod 1: Fragment kodu wykrywający obiekty dla **Sparse Optical Flow**

### 3.2 Dense

Poniżej znajdują się porównania DBSCAN oraz progowania (Rys. 5 i 6). Zmodyfikowany fragment kodu znajduje się na kolejnej stronie (Kod 2).



Obraz oryginalny

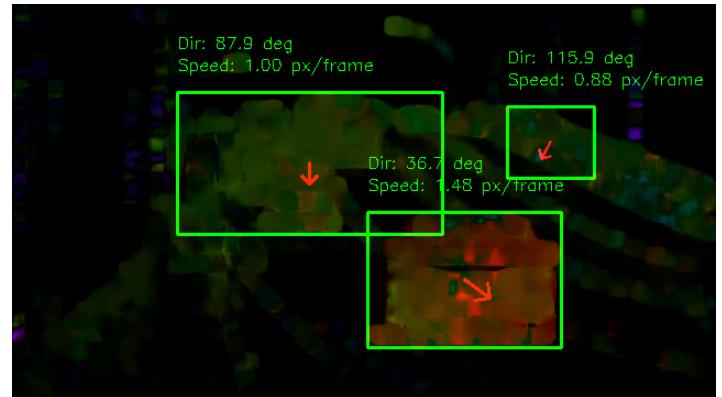


Wyznaczony przepływ optyczny

Rys. 5: Otrzymana klatka z wyznaczonymi prędkościami i kierunkami za pomocą  
**klasteryzacji DBSCAN, Dense Optical Flow**



Obraz oryginalny



Wyznaczony przepływ optyczny

Rys. 6: Otrzymana klatka z wyznaczonymi prędkościami i kierunkami za pomocą  
**progowania, Dense Optical Flow**

```

if drawBboxes:
    if bboxMethod == "dbSCAN":
        from sklearn.cluster import DBSCAN
        h, w = mag.shape
        motionPoints = np.column_stack((np.where(mag > 0)))
        if len(motionPoints) > 0:
            motionVectors = np.column_stack((motionPoints, mag[motionPoints[:, 0],
            motionPoints[:, 1]]))
            clustering = DBSCAN(eps=clusteringEps, min_samples=minClusterSize,
            metric=clusteringMethod, n_jobs=-1).fit(motionPoints)
            labels = clustering.labels_
            for label in set(labels):
                if label == -1:
                    continue
                clusterPoints = motionPoints[labels == label]
                x, y, w, h = cv.boundingRect(clusterPoints)
                x, y, w, h = [int(v / scaleFactor) for v in (x, y, w, h)]
                avgSpeed = mag[clusterPoints[:, 0], clusterPoints[:, 1]].mean()
                avgAngle = np.degrees(ang[clusterPoints[:, 0], clusterPoints[:, 1]].mean())
                cv.rectangle(overlay, (x, y), (x + w, y + h), (0, 255, 0), 2)
                txtColor = (0, 255, 0)
                speedText = f"Speed: {avgSpeed:.2f} px/frame"
                directionText = f"Dir: {avgAngle:.1f} deg"
                cv.putText(overlay, speedText, (x, y - 20), cv.FONT_HERSHEY_SIMPLEX,
                0.5, txtColor, 1)
                cv.putText(overlay, directionText, (x, y - 40),
                cv.FONT_HERSHEY_SIMPLEX, 0.5, txtColor, 1)
    elif bboxMethod == "threshold":
        motionMagnitude = cv.normalize(mag, None, 0, 255, cv.NORM_MINMAX)
        _, motionMask = cv.threshold(motionMagnitude, thresholdMagnitude, 255,
        cv.THRESH_BINARY)
        motionMask = motionMask.astype(np.uint8)
        contours, _ = cv.findContours(motionMask, cv.RETR_EXTERNAL,
        cv.CHAIN_APPROX_SIMPLE)
        for cnt in contours:
            if cv.contourArea(cnt) > 500:
                x, y, w, h = cv.boundingRect(cnt)
                x, y, w, h = [int(v / scaleFactor) for v in (x, y, w, h)]
                tmpMask = np.zeros_like(motionMask, dtype=np.uint8)
                cv.drawContours(tmpMask, [cnt], -1, color=255, thickness=-1)
                mag_inside = mag[tmpMask > 0]
                ang_inside = ang[tmpMask > 0]
                if mag_inside.size > 0:
                    avgSpeed = np.mean(mag_inside)
                    avgAngle = np.degrees(np.mean(ang_inside))
                else:
                    avgSpeed, avgAngle = 0, 0
                cv.rectangle(overlay, (x, y), (x + w, y + h), (0, 255, 0), 2)
                avgMotionX = avgSpeed * np.cos(np.radians(avgAngle))
                avgMotionY = avgSpeed * np.sin(np.radians(avgAngle))
                startPoint = (int(x + w / 2), int(y + h / 2))
                endPoint = (
                    int(startPoint[0] + avgMotionX * 20),
                    int(startPoint[1] + avgMotionY * 20),
                )
                cv.arrowedLine(overlay, startPoint, endPoint, (0, 0, 255), 2,
                tipLength=0.5)
                txtColor = (0, 255, 0)
                speedText = f"Speed: {avgSpeed:.2f} px/frame"
                directionText = f"Dir: {avgAngle:.1f} deg"
                cv.putText(overlay, speedText, (x, y - 20), cv.FONT_HERSHEY_SIMPLEX,
                0.5, txtColor, 1)
                cv.putText(overlay, directionText, (x, y - 40),
                cv.FONT_HERSHEY_SIMPLEX, 0.5, txtColor, 1)

```

Kod 2: Fragment kodu wykrywający obiekty dla **Dense Optical Flow**

## 4 Zadanie 4 - Pomiar czasu i parametryzacja detekcji

W tym zadaniu dodano warunek, który pomija rysowanie bbox i wyznaczonych wartości, gdy nie przekraczają one zadanego przez nas progu.

### 4.1 Sparse

Poniżej znajdują się wyniki dla klasteryzacji i progowania (plik wideo z dysku, Rys. 7 i 8) oraz progowanie (obraz na żywo z kamery, Rys. 9). Fragment kodu znajduje się na kolejnej stronie (Kod 3).



Prędkość ograniczona do  $0.6 \frac{px}{frame}$



Prędkość ograniczona do  $2.0 \frac{px}{frame}$

Rys. 7: Otrzymana klatka z ograniczeniem prędkości za pomocą **klasteryzacji DBSCAN, Sparse Optical Flow**

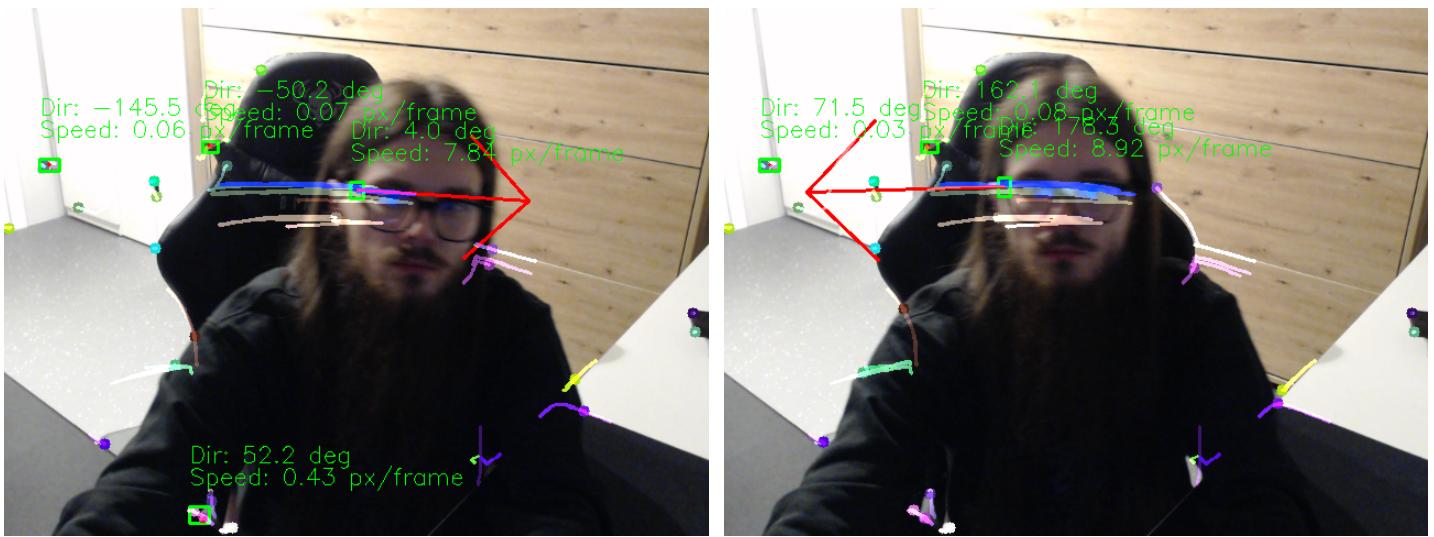


Prędkość ograniczona do  $0.8 \frac{px}{frame}$



Prędkość ograniczona do  $2.0 \frac{px}{frame}$

Rys. 8: Otrzymana klatka z ograniczeniem prędkości za pomocą **progowania, Sparse Optical Flow**



Rys. 9: Otrzymana klatka **live feed** bez ograniczenia prędkości za pomocą **progowania, Sparse Optical Flow**

```
...
# Filter by speed and direction
if directionFilter:
    cv.putText(frame, f"Direction filter: <{directionFilter[0]:.1f}, {directionFilter[1]:.1f}> deg", (frame.shape[0] // 2, 40),
    cv.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 1)
    lower, upper = directionFilter
    if not (lower <= directionAngle <= upper):
        continue

if speedFilter:
    cv.putText(frame, f"Speed filter: {speedFilter:.2f} px/frame", (frame.shape[0] // 2, 20),
    cv.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 1)
    if speed < speedFilter:
        continue
...

```

Kod 3: Fragment kodu ograniczający wykrywanie dla **Sparse Optical Flow**

#### 4.1.1 Średnie czasy analizy pojedynczej klatki

- **Klasteryzacja (plik)**

Average mean operation time per frame: 0.0660 sec/frame

- **Progowanie (plik i obraz na żywo)**

Average mean operation time per frame: 0.0395 sec/frame

## 4.2 Dense

Podobnie jak w sekcji **Sparse**, poniżej znajdują się wyniki dla klasteryzacji i progowania (Rys. 10 i 11) oraz progowanie (Rys. 12). Fragment kodu znajduje się na kolejnej stronie (Kod 4).

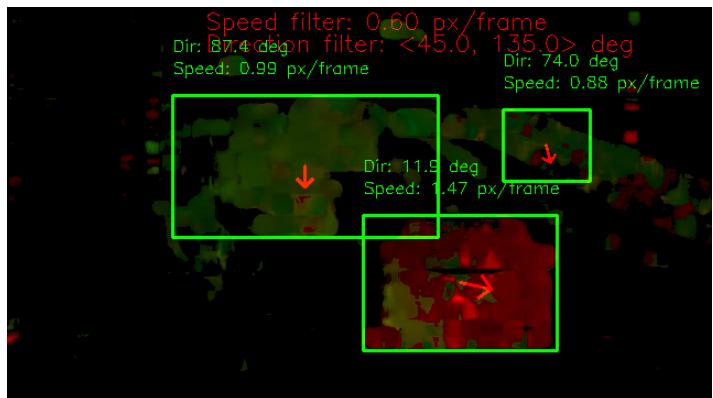


Prędkość ograniczona do  $0.6 \frac{px}{frame}$



Prędkość ograniczona do  $2.0 \frac{px}{frame}$

Rys. 10: Otrzymana klatka z ograniczeniem prędkości za pomocą **klasteryzacji DBSCAN, Dense Optical Flow**

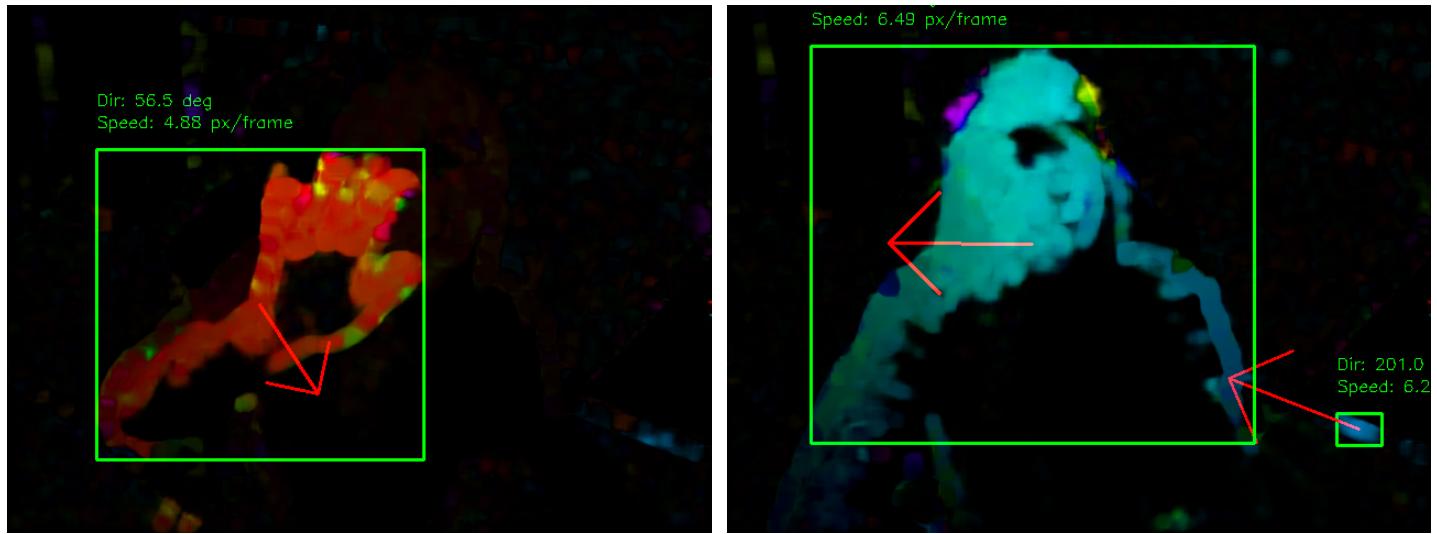


Prędkość ograniczona do  $0.6 \frac{px}{frame}$



Prędkość ograniczona do  $2.0 \frac{px}{frame}$

Rys. 11: Otrzymana klatka z ograniczeniem prędkości za pomocą **progowania, Dense Optical Flow**



Rys. 12: Otrzymana klatka **live feed** bez ograniczenia prędkości za pomocą **progowania, Dense Optical Flow**

```

...
# Filter by speed and direction
if speedFilter or directionFilter:
    tmpMaskMag = np.ones_like(mag, dtype=bool)
    tmpMaskAng = np.ones_like(ang, dtype=bool)

    if speedFilter:
        cv.putText(overlay, f"Speed filter: {speedFilter:.2f} px/frame",
                   (frame.shape[0] // 2, 20), cv.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 1)
        tmpMaskMag &= (mag > speedFilter)

    if directionFilter:
        cv.putText(overlay, f"Direction filter: <{directionFilter[0]:.1f}, {directionFilter[1]:.1f}> deg",
                   (frame.shape[0] // 2, 40), cv.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 1)
        lower, upper = directionFilter
        tmpMaskAng &= (ang * 180 / np.pi >= lower) & (ang * 180 / np.pi <= upper)

    mag = mag * tmpMaskMag
    ang = ang * tmpMaskAng
...

```

Kod 4: Fragment kodu ograniczający wykrywanie dla **Dense Optical Flow**

#### 4.2.1 Średnie czasy analizy pojedynczej klatki

- **Klasteryzacja (plik)**

Average mean operation time per frame: 0.1323 sec/frame

- **Progowanie (plik i obraz na żywo)**

Average mean operation time per frame: 0.1088 sec/frame

## 5 Kod

Przygotowana na potrzeby tego laboratorium paczka, została zaktualizowana, aby wykonywać wszystkie wymagane zadania z tego laboratorium:

- odczyt serii zdjęć ze wskazanego folderu,
- odczyt wskazanego pliku wideo,
- odczyt kamery na żywo,
- wyznaczanie rzadkiego przepływu optycznego (**Shi-Tomasi corner detection** i **Lucas-Kanade optical flow**),
- wyznaczanie gęstego przepływu optycznego (**Farneback's algorithm**),
- wykrywanie ruchomych obiektów (oba przepływy) za pomocą klasteryzacji (**DBSCAN** - **Density-Based Spatial Clustering of Applications with Noise**) oraz programowania wartości skali ruchu (**Thresholding the motion magnitude**),
- otaczanie wykrytych obiektów (obwiednia),
- określenie i wizualizacja prędkości ruchu wykrytego obiektu (tekst nad bounding box),
- określenie i wizualizacja kierunku ruchu obiektu (tekst nad bbox i strzałka reprezentująca wektor kierunku),
- obliczanie średniego czasu przetwarzania pojedynczej klatki obrazu,
- możliwość parametryzacji programu (określanie minimalnej prędkości potrzebnej do zarejestrowania ruchu i limitów dla kątów przemieszczania),
- możliwość skalowania obrazu wejściowego (dla dużych rozdzielczości).

Kod paczki jest dostępny w [serwisie GitHub](#), jednak kopia kodu źródłowego została załączona do sprawozdania w celu łatwiejszej weryfikacji wyników.