# PythonとOpenCVを用いた動画作成技術解説

## 1. 序論

OpenCV (Open Source Computer Vision Library) は、画像処理およびコンピュータビジョンタスクのための強力なオープンソースライブラリです。静止画像の処理に加えて、OpenCVは動画ファイルの読み込み、処理、そして作成のための機能も提供します。本稿では、Pythonプログラミング言語とOpenCVライブラリを使用して動画ファイルを作成するプロセスについて、技術的な詳細を解説します。基本的なプロセス、主要な関数とパラメータ、フレームの取得方法、書き込みループの実装、リソース管理の重要性、そして一般的な問題とその解決策について詳述します。

## 2. 動画作成のコア: cv2.VideoWriter クラス

OpenCVで動画ファイルを作成する際の中心的な要素は cv2.VideoWriter クラスです。このクラスは、一連の画像フレームを受け取り、指定されたパラメータに基づいて動画ファイルとしてエンコードし、ディスクに書き出す機能を提供します。
VideoWriter オブジェクトを初期化するには、以下の情報が必要です。

- 出力ファイル名 (パスを含む)
- FourCC (フォーシーシー) コーデックコード
- フレームレート (FPS: Frames Per Second)
- フレームサイズ (幅と高さ)
- (任意) カラーフラグ (フレームがカラーかグレースケールか)

これらのパラメータは、生成される動画ファイルの特性を決定します。オブジェクトが正常に初期化されたかどうかを確認するために、isOpened() メソッドを使用することが推奨されます。このメソッドが False を返した場合、パラメータの誤りや必要なコーデックの欠如など、何らかの問題が発生していることを示します。

Python

```
# 例: VideoWriterオブジェクトの初期化
output_filename = 'output.avi'
fourcc = cv2.VideoWriter_fourcc(*'MJPG')
fps = 30.0
frame_width = 640
frame_height = 480

writer = cv2.VideoWriter(output_filename, fourcc, fps, (frame_width, frame_height))
```

```
if not writer.isOpened():
    print("Error: VideoWriterの初期化に失敗しました。")
else:
    print("VideoWriterは正常に初期化されました。")
```

# 3. 動画作成の主要パラメータ

VideoWriter の初期化には、動画の特性を定義するいくつかの重要なパラメータが必要です。それぞれのパラメータの意味と設定方法を理解することが、意図した通りの動画を作成するために不可欠です。

## 3.1 出力ファイル名 (filename)

作成する動画ファイルの名前とパスを指定します。拡張子 (例: .avi, .mp4) は、使用するコンテナフォーマットを示唆しますが、最終的な互換性は選択されたFourCCコーデックとバックエンドライブラリに依存します。書き込み権限のある有効なパスを指定する必要があります。

## 3.2 FourCC コーデック (fourcc)

FourCC (Four Character Code) は、使用するビデオコーデックを指定するための4バイトのコードです。コーデックは、動画データを圧縮および解凍する方法を定義します。cv2.VideoWriter_fourcc() 関数を使用して、4つの文字からFourCCコードを生成するのが一般的です。
例:

- cv2.VideoWriter_fourcc(*'MJPG') : Motion JPEG コーデック。比較的互換性が高く、AVIコンテナでよく使用されます。
- cv2.VideoWriter_fourcc(*'XVID') : Xvid コーデック。MPEG-4 Part 2 準拠のコーデックで、AVIコンテナで一般的です。
- cv2.VideoWriter_fourcc(*'mp4v') : MPEG-4 コーデック。MP4コンテナで使用されることがあります。
- cv2.VideoWriter_fourcc(*'H264') : H.264/AVC コーデック。高圧縮率で広く使用されていますが、利用可能性はバックエンド（FFmpegなど）に依存します。

重要な注意点: 指定されたFourCCがシステムにインストールされており、OpenCVが使用しているバックエンドライブラリ（FFmpeg, GStreamerなど）から利用可能である必要があります。利用できないFourCCを指定すると、VideoWriter の初期化が失敗する（isOpened() が False を返す）主な原因となります。互換性の問題を避けるためには、広くサポートされているコーデック（例: 'MJPG'）から試すか、ターゲット環境で利用可能なコーデックを確認することが推奨されます。Windows環境では、FourCCに -1 を指定すると、利用可能なコーデックを選択するためのダイアログが表示される場合がありますが、これはデバッグ目的でのみ有用です。

## 3.3 フレームレート (fps)

FPS (Frames Per Second) は、1秒間に表示されるフレーム数を指定します。一般的な値には、24,

30, 60 などがあります。この値は動画の滑らかさに影響します。ソース（カメラや既存動画）からフレームを取得する場合、そのソースのFPSに合わせることが多いですが、任意の値に設定することも可能です。ただし、再生速度が意図したものと異なる可能性があります。

### 3.4 フレームサイズ (frameSize)

動画の各フレームの幅と高さをピクセル単位でタプル (width, height) として指定します。このサイズは、VideoWriter に書き込むすべてのフレームのサイズと完全に一致している必要があります。サイズが一致しないフレームを書き込もうとすると、エラーが発生するか、予期しない動作を引き起こす可能性があります。フレームソース（カメラなど）から取得する場合、cap.get(cv2.CAP_PROP_FRAME_WIDTH) や cap.get(cv2.CAP_PROP_FRAME_HEIGHT) を使用して実際のフレームサイズを取得し、それを VideoWriter に設定するのが確実です。

### 3.5 カラーフラグ (isColor)

(任意) このフラグは、書き込むフレームがカラーかグレースケールかを指定します。デフォルトは True (カラー) です。グレースケール動画を作成する場合は False に設定します。書き込むフレームの実際のチャンネル数（カラーなら3チャンネル、グレースケールなら1チャンネル）と、このフラグの設定が一致している必要があります。

## 4. フレームの取得または生成

動画は一連の静止画像（フレーム）から構成されます。VideoWriter に書き込むフレームは、様々な方法で取得または生成できます。

- 画像ファイルからの読み込み: cv2.imread() を使用して、ディスク上の画像ファイルを順番に読み込み、フレームとして使用します。ファイル名の命名規則（例: frame_001.png, frame_002.png,...）を定義し、ループ内で読み込むのが一般的です。
- プログラムによる描画: NumPy 配列としてフレームを生成し、cv2.line(), cv2.circle(), cv2.putText() などの描画関数を使用して、アニメーションやグラフィックをプログラムで作成します。各ループイテレーションでフレームの内容を更新することで、動的なコンテンツを生成できます。
- カメラからのキャプチャ: cv2.VideoCapture(0) (デフォルトのカメラ) などを使用してカメラデバイスを開き、read() メソッドでリアルタイムにフレームを取得します。これにより、ライブ映像を録画できます。
- 既存の動画ファイルからの読み込み: cv2.VideoCapture('input.mp4') を使用して既存の動画ファイルを開き、read() メソッドでフレームを一つずつ読み込みます。これにより、動画の変換や加工処理を行いながら新しい動画として保存できます。

いずれの方法でも、VideoWriter に書き込む前に、フレームが有効であり（例: read() が True を返す）、かつ VideoWriter に設定された frameSize と isColor パラメータに一致していることを確認する必要があります。

# 5. フレーム書き込みループとリソースの解放

フレームソースと VideoWriter オブジェクトが準備できたら、ループ処理内でフレームを取得し、VideoWriter の write() メソッドを使用して動画ファイルに書き込みます。

Python

```
# 例: フレーム書き込みループ
# source は cv2.VideoCapture オブジェクトまたは他のフレーム生成源
try:
    while True:
        ret, frame = source.read() # またはフレームをロード/生成
        if not ret:
            print("フレームの取得が終了または失敗しました。")
            break

        # フレーム処理 (任意)
        # 例: frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # フレームサイズとカラー設定が VideoWriter の設定と一致することを確認
        # (必要に応じてリサイズや色空間変換を行う)

        writer.write(frame) # フレームを動画ファイルに書き込む
finally:
    print("書き込みループを終了し、リソースを解放します。")
    writer.release() # VideoWriter オブジェクトを解放
    # フレームソースも解放が必要な場合 (例: カメラ)
    if isinstance(source, cv2.VideoCapture):
        source.release()
```

## 5.1 write() メソッド

ループ内で取得または生成された各フレーム（NumPy 配列）は、writer.write(frame) を呼び出すことで動画ファイルに追加されます。前述の通り、このフレームのサイズと色空間は VideoWriter の初期化時に指定されたパラメータと一致している必要があります。

## 5.2 release() メソッドの重要性

動画作成プロセスにおいて、**writer.release()** メソッドの呼び出しは極めて重要です。すべてのフレームを書き込んだ後、このメソッドを呼び出すことで、以下の処理が行われます。

1. ファイルヘッダとメタデータの書き込み: コーデックやコンテナフォーマットが必要とするヘッダ情報、インデックス情報などがファイルに書き込まれ、ファイルが「完成」します。
2. バッファのフラッシュ: 書き込み処理中にメモリ内にバッファリングされていたデータがディスクに確実に書き込まれます。
3. リソースの解放: ファイルハンドルやコーデックインスタンスなど、VideoWriter が確保していたシステムリソースが解放されます。

release() を呼び出さない場合、動画ファイルが不完全になったり、サイズが0バイトになったり、全く再生できなくなったりする可能性が非常に高いです。これは、ファイルが適切に閉じられていないためです。

Pythonのガベージコレクションはメモリオブジェクトを管理しますが、ファイルハンドルやシステムライブラリへの接続のような外部リソースを即座に、あるいはエラー発生時に確実に解放するとは限りません。release() は、OpenCVとそのバックエンドライブラリに対して、必要なクリーンアップとファイナライズ処理を実行するよう明示的に指示する手段を提供します。
この確実なリソース解放を実現するために、try...finally ブロックを使用することが強く推奨されます。finally 節内のコードは、try ブロック内でエラーが発生した場合でも、必ず実行されます。これにより、プログラムが予期せず終了した場合でも writer.release() が呼び出され、動画ファイルが破損するリスクを最小限に抑えることができます。VideoWriter オブジェクトは Python の with ステートメント（コンテキストマネージャ）をネイティブにはサポートしていないため、try...finally が標準的なパターンとなります。

# 6. 完全なコード例: 基本的な動画の作成

ここでは、カメラから5秒間映像をキャプチャし、それをAVIファイルとして保存する簡単なPythonスクリプトの例を示します。

Python

```
import cv2
import numpy as np
import time

# パラメータ設定
output_filename = 'output_camera.avi'
fourcc = cv2.VideoWriter_fourcc(*'MJPG') # AVIにはMJPGが比較的互換性が高い
fps = 20.0 # カメラの実際のFPSに近づけるか、任意の値に設定
capture_duration = 5 # 録画時間（秒）

# 1. フレームソースの初期化 (デフォルトカメラ)
cap = cv2.VideoCapture(0)
```

```
if not cap.isOpened():
    print("Error: カメラを開けませんでした。")
    exit()

# 2. フレームサイズをカメラから取得
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
print(f"カメラ解像度: {frame_width}x{frame_height}")

# 3. VideoWriter の初期化
writer = cv2.VideoWriter(output_filename, fourcc, fps, (frame_width, frame_height))

if not writer.isOpened():
    print(f"Error: VideoWriterの初期化に失敗しました。FourCC: {''.join(chr((fourcc >> 8*i) & 0xFF)
for i in range(4))}, ファイル名: {output_filename}")
    cap.release()
    exit()

print(f"{output_filename} への書き込みを開始します。{capture_duration}秒間録画します...")

start_time = time.time()

# 4. フレーム書き込みループ (try...finally を使用)
try:
    while True:
        # フレーム取得
        ret, frame = cap.read()

        if not ret:
            print("Error: カメラからフレームを取得できませんでした。")
            break

        # 経過時間チェック
        current_time = time.time()
        if current_time - start_time > capture_duration:
            print(f"{capture_duration}秒経過したため、録画を終了します。")
            break

        # (任意) フレームへの処理 (例: タイムスタンプの描画)
        timestamp = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime(current_time))
        cv2.putText(frame, timestamp, (10, frame_height - 10), cv2.FONT_HERSHEY_SIMPLEX,
0.6, (0, 255, 0), 2)
```

```
        # 5. フレーム書き込み
        writer.write(frame)

        # (任意) 画面表示と終了キー処理
        cv2.imshow('Recording...', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            print("'q'キーが押されたため、録画を中断します。")
            break

finally:
    # 6. リソース解放
    print("リソースを解放しています...")
    cap.release()
    writer.release()
    cv2.destroyAllWindows()
    print(f"動画ファイル {output_filename} が保存されました。")
```

コード解説:

1. インポート: cv2, numpy, time ライブラリをインポートします。
2. パラメータ定義: 出力ファイル名、FourCC、FPS、録画時間を定義します。
3. フレームソース初期化: cv2.VideoCapture(0) でデフォルトカメラを開きます。
4. フレームサイズ取得: カメラから実際のフレーム幅と高さを取得し、VideoWriter の初期化に使用します。これにより、サイズ不一致のエラーを防ぎます。
5. **VideoWriter** 初期化: 取得したパラメータで VideoWriter オブジェクトを作成し、isOpened() で成功を確認します。
6. 書き込みループ: try...finally ブロック内でループを実行します。
    - cap.read() でフレームを取得します。
    - ret が False ならループを抜けます。
    - 録画時間を超えたらループを抜けます。
    - (任意) cv2.putText でフレームにタイムスタンプを描画します。
    - writer.write(frame) でフレームをファイルに書き込みます。
    - (任意) cv2.imshow で録画中の映像を表示し、q キーで中断できるようにします。
7. リソース解放: finally ブロック内で、cap.release() と writer.release() を呼び出してカメラと VideoWriter を確実に解放します。cv2.destroyAllWindows() で表示ウィンドウを閉じます。
8. メッセージ: 処理の開始、終了、エラーを示すメッセージを出力します。

# 7. 一般的な動画作成の問題とトラブルシューティング

OpenCVでの動画作成は、コーデックの互換性やパラメータ設定の誤りなど、いくつかの要因で問題が発生することがあります。以下に一般的な問題とその解決策を示します。

## 7.1 VideoWriter オブジェクトが作成されない、または isOpened() が False を返す

- 症状: writer オブジェクトが None になる、または writer.isOpened() が False を返す。その後の write() 呼び出しは失敗する。
- 一般的な原因:
  - 無効な**FourCC:** 指定されたFourCCコードがシステムにインストールされていないか、OpenCVのバックエンドでサポートされていない。
  - バックエンドの問題: 基盤となるビデオライブラリ（FFmpeg, GStreamer等）に問題があるか、OpenCVが適切なバックエンドサポート付きでビルドされていない。
  - 権限不足: プログラムが出力先ディレクトリへの書き込み権限を持っていない。
  - 無効なファイル名/パス: パスが存在しないか、無効な文字が含まれている。
- デバッグ手順:
  - 広く互換性のある別のFourCCコード（例: 'MJPG', 'DIVX', 'XVID' (AVI用); 'mp4v' (MP4用、要確認)）を試す。
  - FourCC引数に -1 を指定してみる（Windowsのみ）。コーデック選択ダイアログが表示されれば、利用可能なコーデックのヒントになる。
  - 出力パスが存在し、書き込み権限があることを確認する。
  - OpenCVのビルド情報 (cv2.getBuildInformation()) を確認し、どのビデオバックエンドが有効になっているか調べる。必要であれば、OpenCVを再インストールするか、不足しているバックエンドライブラリ（例: FFmpeg）をインストールする。
  -

## 7.2 動画ファイルは作成されるが、サイズが0バイトまたは再生できない

- 症状: output.avi のようなファイルは存在するが、サイズが0 KBであるか、メディアプレーヤーで開くとエラーが表示される。
- 一般的な原因:
  - **writer.release()** が呼び出されていない: ファイルが適切にファイナライズされていない。これが最も一般的な原因です。
  - フレームが書き込まれていない: 書き込みループが一度も実行されなかったか、すぐに終了した。
  - 早期のエラー: 書き込みループ中にエラー（例: フレームサイズの不一致）が発生し、release() が呼び出される前にプログラムが終了し、かつ try...finally が使用されていなかった。
- デバッグ手順:
  - **release()** の呼び出しを確認: コードを見直し、特に try...finally ブロック内の finally 節に release() が正しく配置されていることを確認する。
  - ループ内に print 文を追加し、フレームが処理され write() が呼び出されていることを確認する。
  - 実行中にコンソールに出力されるエラーメッセージを確認する。
  -

## 7.3 writer.write(frame) 中にエラーが発生する

- 症状: writer.write(frame) の行でPythonの例外が発生する。
- 一般的な原因:
  - フレームサイズの不一致: frame の次元 (高さ、幅) が VideoWriter に指定された frameSize と一致しない。
  - 色の不一致: フレームはカラーだが isColor=False が指定されている、またはその逆。
  - 無効なフレームデータ: frame が None であるか、データが破損している。
- デバッグ手順:
  - ループ内で writer.write(frame) を呼び出す直前に print(frame.shape) を実行し、その結果を VideoWriter に渡した frameSize (幅、高さ) と比較する。
  - frame.shape のチャンネル数を確認し、isColor の設定と一致しているか確認する (カラーなら通常3、グレースケールなら通常1またはチャンネル次元なし)。
  - 書き込む前にフレームが正しく読み込まれたか、または生成されたかを確認する。一時的に cv2.imshow() を使用して、書き込もうとしているフレームを視覚化する。

## 7.4 コーデック/再生の問題

- 症状: 動画ファイルは作成されるが、ターゲットシステムやプレーヤーで正しく再生できない、映像が乱れるなど。
- 一般的な原因:
  - コーデックの欠如: 再生システムに、エンコードに使用された特定のコーデックがインストールされていない。
  - 非互換のプレーヤー: 使用しているメディアプレーヤーが、そのコーデックやコンテナフォーマットをサポートしていない。
  - コーデックのバグ: まれに、コーデックの実装自体に問題がある場合がある。
- デバッグ手順:
  - 別のメディアプレーヤー（例: VLC media player は多くのコーデックを内蔵している）で再生を試す。
  - より一般的で互換性の高いFourCC（例: AVIなら 'MJPG'、FFmpegバックエンドが利用可能ならMP4の一般的なオプション）を使用して動画を再エンコードしてみる。
  - 再生システムに必要なコーデックパックがインストールされているか確認する（ただし、特定のコーデックパックのインストールを要求するソフトウェアの配布は、ユーザーにとって負担となる可能性がある）。
  - 問題を切り分けるために、作成した動画ファイルを別のマシンや他の人に提供して再生可能か確認する。

## 7.5 トラブルシューティング概要表

以下の表は、一般的な問題とその診断・解決策をまとめたものです。

表 7.1: 一般的な動画作成エラーと解決策

| 症状 | 考えられる原因 | デバッグ/解決手順 |
|---|---|---|

| | | |
|---|---|---|
| VideoWriter オブジェクト初期化失敗 (isOpened() が False) | 1. 無効/非対応のFourCC <br> 2. OpenCVバックエンドの問題/不足 <br> 3. 出力先への書き込み権限不足 <br> 4. 無効なファイルパス/名前 | 1. 別のFourCC (例: 'MJPG') を試す。Windowsなら -1 を試す。 <br> 2. cv2.getBuildInformation() でバックエンドを確認。必要ならOpenCV再インストールやFFmpeg等を追加。<br> 3. パスと権限を確認。<br> 4. パス文字列を確認。 |
| ファイルは作成されるが0バイトまたは再生不可 | 1. **writer.release()** が未呼び出し <br> 2. フレームが一度も write() されていない <br> 3. try...finally なしで途中でエラー終了 | 1. コードを確認し、finally 節で release() が確実に呼ばれるようにする。<br> 2. ループ内で write() が呼ばれているか print で確認。<br> 3. コンソールエラーを確認し、try...finally を使用する。 |
| writer.write(frame) でエラー発生 | 1. フレームサイズが frameSize と不一致 <br> 2. フレームの色空間が isColor と不一致 <br> 3. frame が None または破損 | 1. write() 直前に frame.shape を出力し、(height, width) が期待通りか確認。必要なら cv2.resize()。<br> 2. frame.shape のチャンネル数と isColor を確認。必要なら cv2.cvtColor()。<br> 3. フレーム取得/生成コードを確認。 |
| 作成された動画が再生できない/乱れる | 1. 再生環境に必要なコーデックがない <br> 2. メディアプレーヤーが非対応 <br> 3. コーデック自体の問題 | 1. 別のプレーヤー (VLC等) で試す。<br> 2. より互換性の高い FourCC (例: 'MJPG') で再作成する。<br> 3. 再生環境にコーデックをインストール (注意が必要)。<br> 4. 別の環境で再生テストする。 |

この表は、問題発生時に症状から原因を推測し、具体的な対策を講じるための迅速な診断ガイドとして機能します。

# 8. 基本を超えて: 高度な技術（概要）

OpenCVの基本的な動画作成機能を理解した上で、さらに高度な応用も可能です。

## 8.1 動画への音声の追加

OpenCVの VideoWriter は、現時点ではビデオフレームのみを扱い、音声ストリームを直接処理・統

合する機能は提供していません。OpenCVで作成した映像ファイルに音声を追加するには、外部のツールやライブラリを使用する必要があります。

- **MoviePy:** Pythonでビデオ編集を行うための高レベルライブラリ。音声ファイルの読み込み、OpenCVで作成した動画との結合、エフェクト追加などが比較的容易に行えます。内部的にFFmpegを使用していることが多いです。
- **FFmpeg ( subprocess 経由):** 強力なマルチメディアフレームワークであるFFmpegのコマンドラインツールを、Pythonの subprocess モジュールを使って直接呼び出す方法。OpenCVで作成した映像ファイル（例: video_only.mp4）と別の音声ファイル（例: audio.mp3）を結合（mux）して最終的な動画ファイル（例: final_output.mp4）を作成できます。

概念的な手順は以下のようになります。

1. OpenCVで映像のみの動画ファイルを作成する。
2. 別途、使用する音声ファイルを用意する。
3. MoviePyライブラリまたは subprocess + FFmpeg を使用して、映像ファイルと音声ファイルを結合し、最終的な出力ファイルを作成する。

### 8.2 リアルタイム処理/エフェクトの適用

書き込みループ内で取得または生成されたフレーム frame は、本質的にはNumPy配列です。したがって、writer.write(frame) を呼び出す前に、この frame に対してOpenCVや他の画像処理ライブラリの任意の関数を適用することができます。

例:

- フィルタ適用: cv2.GaussianBlur(), cv2.Canny()
- オーバーレイ描画: 物体検出のバウンディングボックス、テキスト注釈 (cv2.rectangle(), cv2.putText())
- 色空間変換: cv2.cvtColor()
- 幾何学的変換: cv2.warpAffine(), cv2.resize()

これにより、コンピュータビジョンの結果（例: 物体追跡）、特殊効果、注釈などを直接埋め込んだ動画を作成することが可能です。

### 8.3 パフォーマンスに関する考慮事項

複雑なフレームごとの処理、低速なディスクI/O、非効率的なコーデックの選択などは、動画作成のボトルネックになる可能性があります。特に、高解像度・高フレームレートでのリアルタイム処理と書き込みは、相応の計算リソースを要求します。パフォーマンスが問題となる場合は、処理の最適化、効率的なコーデックの選択、非同期処理の導入などを検討する必要があります。

## 9. 結論

本稿では、PythonとOpenCVライブラリを用いて動画ファイルを作成するための技術的な側面を解説しました。主要な要素である cv2.VideoWriter クラスの初期化、FourCCコーデック、フレームレート、フレームサイズといった重要なパラメータの設定、様々なソースからのフレーム取得方法、そしてフレームを書き込むループ処理について詳述しました。
特に、動画ファイルを正しく完成させ、リソースリークを防ぐために writer.release() メソッドを try...finally ブロック内で確実に呼び出すことの重要性を強調しました。また、コーデックの互換性、パラメータの正確性、リソース管理に関連する一般的な問題とそのトラブルシューティング方法についても触れました。
OpenCVによる動画作成は、パラメータと環境依存性を正しく理解すれば、強力なツールとなります。本稿で概説した基本原則とベストプラクティスに基づき、さらなる実験とOpenCV公式ドキュメントの参照を通じて、より高度な動画処理アプリケーションの開発が可能となるでしょう。