PythonとOpenCVによる境界検知の実装：詳細解説とコードサンプル

1. 画像処理における境界検知の概要

1.1. エッジの定義とその重要性

画像処理における「エッジ」とは、画像内の輝度値や色が急激に変化する箇所を指します[1]。多くの場合、これは画像内の物体の境界、表面の向きの変化、あるいは材質の違いに対応します[2]。エッジは、画像の内容に関する基本的な構造情報を含んでいます[1]。
エッジ検出は、多くのコンピュータビジョンタスクにおいて重要な最初のステップとなります。なぜなら、処理すべきデータ量を削減しつつ、本質的な構造的特徴を保持できるからです[2]。エッジは物体の輪郭や形状を定義し、視覚的な理解を助けるため、画像処理の多くの応用において不可欠な要素です[2]。エッジを特定することで、画像はピクセルの集合体から、意味のある構造的要素の集合へと変換されます。このデータ削減と特徴抽象化のプロセスは、後続のより高度な処理、例えば物体認識などを、計算的に扱いやすく、かつ形状情報に焦点を当てることでロバストにする基盤となります[2]。

1.2. 境界検知（エッジ検出）の目的と応用分野

エッジ検出の主な目的は、画像から物体の境界線を抽出し[2]、画像の内容理解や解析に必要な主要な特徴を特定することです[2]。これにより、画像認識、物体検出、画像セグメンテーションなどの精度を向上させることができます[2]。エッジ情報は、画像の構造を単純化し、本質的な形状情報を抽出する手段となります[4]。
エッジ検出技術は、非常に多様な分野で応用されています。

- 医療画像処理: CTスキャンやMRI画像から腫瘍、血管、臓器の輪郭を特定し、診断支援に利用されます[2]。早期発見や正確な診断に貢献します[2]。
- 自動運転: 車線、道路境界、障害物、歩行者、交通標識などをリアルタイムで検出し、安全なナビゲーションや意思決定に不可欠です[2]。
- 製造業・品質管理: 製品の外観検査において、傷、欠陥、寸法異常などを自動検出し、品質管理の精度と効率を向上させます[2]。
- セキュリティ: 顔認識システムにおける特徴抽出や、監視カメラ映像の解析による異常検知などに用いられます[8]。
- ロボティクス: ロボットが周囲環境の構造を理解し、物体を把持したり、ナビゲーションを行ったりするためにエッジ情報が利用されます[14]。
- 機械学習の前処理: 画像からエッジ特徴を抽出し、機械学習モデルの入力データとして利用されることがあります[9]。

このように、医療から自動車、産業、セキュリティに至るまで、エッジが基本的な画像特徴として普遍的に重要であることがわかります[2]。しかし、この応用分野の多様性は、同時に最適なエッジ検出戦

略（アルゴリズムの選択、パラメータ設定）が応用ごとに異なることを示唆しています。例えば、自動運転ではリアルタイム性が重視される一方、医療画像では高い精度が求められるでしょう。このため、単一のアプローチが常に最適とは限らず、状況に応じた手法の理解と選択が重要になります。

2. Pythonにおける境界検知のためのOpenCV活用

2.1. OpenCVライブラリの概要

OpenCV (Open Source Computer Vision Library) は、コンピュータビジョン、画像処理、機械学習のタスクを実行するための包括的なオープンソースライブラリです [15]。元々はIntelによって開発され、現在は活発なコミュニティによって維持されています。C++、Java、Pythonなど複数のプログラミング言語に対応していますが、特にPythonはその使いやすさとデータサイエンスエコシステムとの親和性の高さから広く利用されています [15]。学術研究だけでなく商用利用も無料で可能なライセンス形態も普及を後押ししています [15]。

OpenCVは非常に広範な機能を提供しており、エッジ検出はその一部に過ぎません。主な機能としては、画像の読み込み・書き込み・表示 [15]、色空間変換（RGB、グレースケール、HSVなど） [17]、リサイズ・回転・トリミングといった幾何学的変換 [18]、平滑化（ぼかし）などのフィルタリング処理 [17]、線・円・矩形・テキストなどの描画 [15]、コーナー検出やSIFT、ORBなどの特徴点検出 [17]、顔検出や物体追跡 [18]、そして機械学習モデル（特に深層学習モデル）との連携機能などが含まれます [15]。

2.2. 画像処理とエッジ検出におけるOpenCVの主要機能

境界検知の実装においては、OpenCVのいくつかの主要な関数が中心的な役割を果たします。

- 画像入出力と表示:
  - cv2.imread(): 画像ファイルを読み込みます [15]。
  - cv2.imshow(), cv2.waitKey(), cv2.destroyAllWindows(): 処理結果をウィンドウに表示し、ユーザーのキー入力を待機し、ウィンドウを閉じます [15]。
- 前処理:
  - cv2.cvtColor(): 画像の色空間を変換します。特に、cv2.COLOR_BGR2GRAY を用いてグレースケール画像に変換する操作は、エッジ検出の前処理として一般的です [17]。
  - cv2.GaussianBlur(), cv2.medianBlur(): 画像のノイズを低減するための平滑化フィルタを適用します [18]。
- エッジ検出:
  - cv2.Canny(): Cannyアルゴリズムによるエッジ検出を実行します [19]。
  - cv2.Sobel(): Sobelフィルタを用いて画像の勾配（1次微分）を計算し、エッジを検出します [19]。
  - cv2.Laplacian(): Laplacianフィルタを用いて画像の2次微分を計算し、エッジを検出します [19]。

OpenCVの大きな利点の一つは、これらの複雑なアルゴリズムを非常にシンプルな関数呼び出しで利用できる点にあります [16]。例えば、Cannyエッジ検出のような多段階の複雑な処理も cv2.Canny() という一行で実行できます。これは、OpenCVが強力な「抽象化レイヤー」として機能していることを意味します。開発者は、アルゴリズムの低レベルな実装詳細を意識することなく、高度な画像処理技術を利用できます。これにより、開発の敷居が大幅に下がり、迅速なプロトタイピングや応用開発が

可能になります [18]。
さらに、OpenCVはパフォーマンスも考慮して設計されており、多くの関数は効率的なC++で実装され、最適化されています。場合によってはハードウェアアクセラレーション（GPU利用など）も可能であり、リアルタイム処理が求められるアプリケーションにも対応できます [8]。また、Pythonで利用する場合、数値計算ライブラリNumPyとの親和性が高く、画像データをNumPy配列として効率的に操作できる点もメリットです [18]。
ただし、OpenCVが実装の複雑さを隠蔽してくれる一方で、その機能を効果的に活用するためには、背後にある画像処理の概念や各関数のパラメータが結果にどう影響するかを理解することが依然として重要です。例えば、単に cv2.Canny() を呼び出すだけでは、適切な閾値設定や前処理（ノイズ除去など）を行わない限り、望ましい結果が得られないことが多いでしょう [1]。したがって、OpenCVは強力なツールですが、その真価を引き出すためには、コンピュータビジョンの基本原則に関する知識が不可欠となります。

3. OpenCVにおける主要な境界検知アルゴリズム

3.1. 概要：Canny、Sobel、Laplacian

OpenCVには、境界（エッジ）を検出するための複数のアルゴリズムが実装されています。本レポートでは、その中でも特に代表的で広く利用されている3つの手法、Canny、Sobel、Laplacianに焦点を当てます。これらは、画像の輝度勾配（輝度値の変化率）に基づいてエッジを検出する古典的な手法です [3]。

- **Canny (キャニー):** John F. Cannyによって開発された、多段階からなる洗練されたエッジ検出アルゴリズムです。ノイズ除去、勾配計算、非極大値抑制、ヒステリシス閾値処理というステップを経てエッジを検出します [27]。精度とノイズ耐性のバランスに優れており、古典的な手法の中ではしばしば最良の結果をもたらすと評価されています [33]。OpenCVでは cv2.Canny() 関数で実装されています [20]。
- **Sobel (ソーベル):** 画像の1次微分を近似することでエッジを検出する手法です [1]。特定の畳み込みカーネル（Sobelオペレータ）を用いて、画像の水平方向（x方向）と垂直方向（y方向）の輝度勾配を個別に計算できます [24]。比較的シンプルで高速なアルゴリズムです。OpenCVでは cv2.Sobel() 関数で実装されています [20]。
- **Laplacian (ラプラシアン):** 画像の2次微分に基づいてエッジを検出する手法です [24]。輝度変化が急峻な箇所（2次微分のゼロ交差点）を検出します。細かいディテールやシャープなエッジの検出に優れていますが、ノイズに非常に敏感であるという特徴があります [34]。OpenCVでは cv2.Laplacian() 関数で実装されています [20]。

これらのアルゴリズムは、それぞれ異なる原理と特性を持っています。Cannyは高精度を目指した複雑な手法、SobelとLaplacianはより単純で高速な微分演算に基づいた手法と位置づけられます [34]。このアルゴリズムの多様性は、エッジ検出が一つの「正解」を持つ単純な問題ではないことを示唆しています。画像の特性（ノイズレベル、コントラスト、対象物の複雑さなど）やアプリケーションの要求（速度、精度など）に応じて、適切なアルゴリズムを選択する必要があります。OpenCVのような標準的なライブラリに複数の選択肢が用意されていること自体が、この問題の文脈依存性を反映していると言えるでしょう。次章以降で、これらのアルゴリズム、特にCannyについて、その仕組みと使い方を詳しく見ていきます。

4. Cannyエッジ検出アルゴリズム：詳細なステップ解説

4.1. Cannyアルゴリズムの紹介

Cannyエッジ検出アルゴリズムは、1986年にJohn F. Cannyによって提案された、非常に影響力のあるエッジ検出手法です [27]。しばしば「最適なエッジ検出器」として言及されるのは、以下の3つの主要な設計基準を満たすことを目指しているためです [28]。

1. 低いエラー率: 存在するエッジのみを良好に検出し、存在しないエッジ（ノイズなど）を検出しない。
2. 良好な局在性: 検出されたエッジピクセルが、実際の物体のエッジに可能な限り近い位置にある。
3. 最小応答: 1つのエッジに対して、検出器の応答（検出されるエッジピクセル）が1つだけである（エッジが太くならない）。

これらの基準を達成するために、Cannyアルゴリズムは複数の段階（ステップ）を経て処理を実行します [1]。以下に各ステップを詳述します。

4.2. ステップ1：ノイズ除去（ガウシアン平滑化）

エッジ検出処理、特に微分演算を含む処理は、画像中のノイズに非常に敏感です [27]。ノイズが存在すると、それが誤ってエッジとして検出されてしまう可能性があります。そのため、Cannyアルゴリズムの最初のステップでは、入力画像に対してガウシアンフィルタ（通常は5x5のカーネルサイズ）を適用し、ノイズを抑制します [27]。

ガウシアンフィルタは、畳み込み演算の一種であり、中心ピクセルとその近傍のピクセル値に対して、ガウス分布に基づいた重み付けを行い、加重平均を計算します [1]。これにより、画像が滑らかになり、特に高周波成分であるノイズが効果的に低減されます。この前処理は、後続の勾配計算ステップで、ノイズによる偽のエッジが生成されるのを防ぐために極めて重要です [1]。

4.3. ステップ2：輝度勾配の計算（Sobelオペレータ）

ノイズが除去された滑らかな画像に対して、次に輝度の勾配（変化率）とその方向を計算します [27]。エッジは輝度が急激に変化する箇所で発生するため、勾配の大きさが大きいピクセルがエッジ候補となります。

この計算には、多くの場合Sobelオペレータ（あるいはそれに類する微分フィルタ）が用いられます。Sobelオペレータは、水平方向（x方向）と垂直方向（y方向）の輝度変化を検出するための畳み込みカーネルです [1]。これらのカーネルを入力画像に適用することで、各ピクセルにおけるx方向の勾配 (G_x) とy方向の勾配 (G_y) が得られます。

これらの値から、各ピクセルの勾配の大きさと方向を計算します。

勾配の大きさ (G) は、通常、以下の式で計算されます：

$$ G = \sqrt{G_x^2 + G_y^2} $$

計算負荷を軽減するために、より単純な近似式 (G = |G_x| + |G_y|) が用いられることもあります 27。

勾配の方向 (\theta) は、以下の式で計算されます：

$$ \theta = \arctan\left(\frac{G_y}{G_x}\right) $$

この勾配方向は、エッジの接線方向に対して常に垂直になります 27。

4.4. ステップ3：非極大値抑制 (Non-Maximum Suppression)

勾配計算の結果、エッジ周辺では勾配の大きい領域が帯状に広がることがあります。しかし、Cannyの設計基準である「最小応答」を満たすためには、エッジを1ピクセルの幅に細線化する必要があります [27]。非極大値抑制は、この細線化を行うためのステップです。
この処理では、画像内の各ピクセルについて、その勾配方向における近傍のピクセルと比較を行います [27]。具体的には、ピクセルの勾配方向（ステップ2で計算）を調べ、その方向に沿った隣接する2つのピクセルの勾配の大きさと比較します。もし、注目しているピクセルの勾配の大きさが、勾配方向上の両隣のピクセルよりも大きい（つまり、局所的な最大値である）場合、そのピクセルはエッジ候補として保持されます。そうでなければ、そのピクセルはエッジではないと見なされ、抑制（勾配の値を0に設定）されます [27]。
勾配方向は連続的な値ですが、効率化のために通常、最も近い4つの方向（水平、垂直、および2つの対角線方向：0度、45度、90度、135度）のいずれかに丸められます [27]。この処理により、太いエッジの峰（リッジ）部分のみが残り、結果として細い線状のエッジ候補が得られます [27]。これはCannyの「最小応答」基準を達成するために不可欠なステップです。

4.5. ステップ4：ヒステリシス閾値処理

非極大値抑制によって得られた細いエッジ候補には、真のエッジだけでなく、ノイズなどによって生じた偽のエッジも含まれている可能性があります。ヒステリシス閾値処理は、これらの候補の中から真のエッジを区別し、最終的なエッジマップを生成するステップです [27]。単一の閾値を用いるのではなく、2つの閾値を使うことで、エッジの連結性を考慮し、よりロバストな判定を行います。
このステップでは、2つの閾値、threshold1（下限閾値、minVal）とthreshold2（上限閾値、maxVal）を使用します [27]。

1. 強いエッジの判定: ピクセルの勾配の大きさ (G) が threshold2 よりも大きい場合、そのピクセルは「強いエッジ」ピクセルと見なされ、最終的なエッジマップに確実に含まれます [27]。
2. 非エッジの判定: ピクセルの勾配の大きさ (G) が threshold1 よりも小さい場合、そのピクセルはエッジではないと見なされ、破棄されます [27]。
3. 弱いエッジの判定と連結性評価: ピクセルの勾配の大きさ (G) が threshold1 と threshold2 の間にある場合、そのピクセルは「弱いエッジ」ピクセルと見なされます [27]。弱いエッジピクセルが最終的なエッジマップに含まれるかどうかは、その連結性によって決まります。具体的には、弱いエッジピクセルが、8近傍（周囲8ピクセル）で強いエッジピクセルに接続されている（直接または他の弱いエッジピクセルを介して間接的に）場合にのみ、最終的なエッジとして採用されます [27]。

この「エッジトラッキング」または「連結性解析」と呼ばれるプロセスにより、弱いながらも真のエッジの一部である可能性のあるピクセルを保持しつつ、ノイズによる孤立した弱いエッジ候補を効果的に除去することができます。これにより、Cannyの「低いエラー率」基準が達成され、途切れにくく連続的な強いエッジが得られます [27]。
Cannyアルゴリズム全体を見ると、ノイズ除去から始まり、勾配計算、非極大値抑制による細線化、そしてヒステリシス閾値処理による検証という一連の洗練されたステップを経ていることがわかります。各ステップが前のステップの結果を改善し、最終的に高品質なエッジ検出を実現するように設計されています。この段階的な改善プロセスこそが、Cannyが他の単純な手法と比較して優れた性能

を発揮する理由です。

5. PythonとOpenCVによるCannyエッジ検出の実装

5.1. cv2.Canny 関数の解説

OpenCVでは、前章で説明したCannyエッジ検出の全ステップ（ノイズ除去、勾配計算、非極大値抑制、ヒステリシス閾値処理）を cv2.Canny() という単一の関数で実行できます。
関数の書式（シグネチャ）:

Python

```
edges = cv2.Canny(image, threshold1, threshold2, apertureSize=3, L2gradient=False)
```

[27]

パラメータ:

- image: 入力画像。8ビットのシングルチャンネル（グレースケール）画像である必要があります[24]。カラー画像を使用する場合は、事前に cv2.cvtColor() でグレースケールに変換しておく必要があります。
- threshold1: ヒステリシス閾値処理で使用される1番目の閾値（下限閾値）[24]。
- threshold2: ヒステリシス閾値処理で使用される2番目の閾値（上限閾値）[24]。慣例的に、threshold2 の方が threshold1 よりも大きい値に設定されます。OpenCVの実装では、内部的に値の大小で役割を判断するため、引数の順序はどちらでも等価です[36]。
- apertureSize (オプション): 内部で使用されるSobelオペレータのカーネルサイズ（アパーチャサイズ）。デフォルトは3です[27]。3, 5, 7 のいずれかの奇数を指定する必要があります。
- L2gradient (オプション): 勾配の大きさを計算する方法を指定するブール値フラグ。True の場合、より精度の高いL2ノルム ($G = \sqrt{G_x^2 + G_y^2}$) を使用します。False (デフォルト) の場合、計算が高速なL1ノルム ($G = |G_x| + |G_y|$) を使用します[27]。

戻り値:

- edges: 出力されるエッジマップ。入力画像と同じサイズ、同じ型（通常は uint8）のバイナリ画像です。エッジ部分は白（または非ゼロ値）、それ以外の背景部分は黒（ゼロ値）で表現されます[24]。

この cv2.Canny 関数は、Cannyアルゴリズムの複雑な内部処理をカプセル化し、ユーザーが主要なパラメータ（特に閾値）を指定するだけで簡単にエッジ検出を実行できるようにします。

5.2. コードサンプル：ステップ・バイ・ステップ実装

以下に、PythonとOpenCVを用いてCannyエッジ検出を行う具体的なコードサンプルを示します。一般的な実装フローに従い、各ステップにコメントを付与しています。

Python

```
# 必要なライブラリをインポート
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os  # ファイル存在確認のため

# --- ステップ 1: 画像ファイルのパスを指定 ---
image_path = 'lena.png'  # ここに処理したい画像ファイルのパスを指定してください

# --- ステップ 2: 画像の読み込み ---
# ファイルが存在するか確認
if not os.path.exists(image_path):
    print(f"エラー: 画像ファイルが見つかりません - {image_path}")
else:
    # カラー画像として読み込む
    img_color = cv2.imread(image_path)

    if img_color is None:
        print(f"エラー: 画像ファイルを読み込めませんでした - {image_path}")
    else:
        # --- ステップ 3: グレースケール変換 ---
        # Cannyアルゴリズムはグレースケール画像を対象とするため変換する
        img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)
        # [27, 29, 30, 36]

        # --- ステップ 4: ノイズ除去（ガウシアンブラー）---
        # Canny関数内部でもある程度のノイズ除去は行われるが、
        # 事前に適用することでより良い結果が得られる場合が多い (推奨)
        # (5, 5) はカーネルサイズ、0 はsigmaX（標準偏差）を指定
        img_blur = cv2.GaussianBlur(img_gray, (5, 5), 0)
        # [1, 27, 33, 37, 42]

        # --- ステップ 5: Cannyエッジ検出の適用 ---
        # threshold1 (下限閾値) と threshold2 (上限閾値) を指定
        # これらの値は画像によって調整が必要
        threshold1 = 50
        threshold2 = 150
        edges = cv2.Canny(img_blur, threshold1, threshold2)
        # [24, 27, 33]

        # --- ステップ 6: 結果の表示 ---
        # matplotlibを使用して元画像とエッジ画像を表示
```

```
    plt.figure(figsize=(10, 5)) # 表示ウィンドウのサイズを設定

    plt.subplot(1, 3, 1) # 1行3列の1番目
    plt.imshow(cv2.cvtColor(img_color, cv2.COLOR_BGR2RGB)) # OpenCVはBGR、matplotlib
はRGBのため変換
    plt.title('Original Color Image')
    plt.axis('off') # 軸を非表示

    plt.subplot(1, 3, 2) # 1行3列の2番目
    plt.imshow(img_gray, cmap='gray')
    plt.title('Grayscale Image')
    plt.axis('off')

    plt.subplot(1, 3, 3) # 1行3列の3番目
    plt.imshow(edges, cmap='gray') # エッジ画像はグレースケールで表示
    plt.title(f'Canny Edges (T1={threshold1}, T2={threshold2})')
    plt.axis('off')

    plt.tight_layout() # サブプロット間のスペースを調整
    plt.show()

    # OpenCVのウィンドウで表示する場合 (matplotlibを使わない場合)
    # cv2.imshow('Original Color', img_color)
    # cv2.imshow('Grayscale', img_gray)
    # cv2.imshow('Canny Edges', edges)
    # cv2.waitKey(0) # 何かキーが押されるまで待機
    # cv2.destroyAllWindows() # 全てのウィンドウを閉じる
    # [24, 27, 33]
```

このコードは、画像ファイルを読み込み、グレースケール変換、ガウシアンブラーによるノイズ除去、そして cv2.Canny 関数によるエッジ検出という一連の標準的なワークフローを実行します [27]。最終的に、元のカラー画像、グレースケール画像、そして検出されたエッジ画像を並べて表示します。

5.3. 結果の視覚化

上記のコードを実行すると、指定した画像に対するエッジ検出結果が視覚的に表示されます。例えば、一般的なテスト画像（例：lena.png）を使用した場合、Cannyアルゴリズムは人物の輪郭、帽子の飾り、背景の構造などを線として抽出します。閾値 threshold1 と threshold2 の値を変更すると、検出されるエッジの量や連続性が変化する様子を観察できます。適切なパラメータ設定により、ノイズが少なく、対象物の形状を的確に捉えたエッジマップが得られます [24]。
cv2.Canny 関数は非常に便利ですが、その内部で行われている4つのステップ（ノイズ除去、勾配計算、非極大値抑制、ヒステリシス閾値処理）を1行のコードに集約しています。この高度な抽象化は実

装を容易にする反面、内部プロセスを直接制御したり、中間結果（例：勾配マップ）を確認したりすることが難しくなります。そのため、期待通りの結果を得るためには、関数のパラメータ、特に閾値の意味と影響を正確に理解し、適切に設定することが極めて重要になります。

6. Cannyパラメータの理解と調整

Cannyエッジ検出の結果は、cv2.Canny 関数に渡されるパラメータ、特に2つの閾値 threshold1 と threshold2 に大きく依存します。これらのパラメータを適切に調整することが、望ましいエッジ検出結果を得るための鍵となります。

6.1. threshold1 と threshold2 の役割

これらの閾値は、Cannyアルゴリズムの最終段階であるヒステリシス閾値処理（セクション 4.5 参照）で使用されます。

- **threshold2 (上限閾値):** この閾値は、「強い」エッジピクセルを識別するために使用されます。勾配の大きさが threshold2 を超えるピクセルは、確実にエッジの一部であると判定されます [27]。これらの強いエッジピクセルは、エッジ追跡の開始点となります。
- **threshold1 (下限閾値):** この閾値は、「弱い」エッジピクセルを候補として識別するために使用されます。勾配の大きさが threshold1 より大きく threshold2 以下であるピクセルが弱いエッジ候補となります。これらの弱いエッジ候補は、それ自体では最終的なエッジとは判定されませんが、強いエッジピクセルに（直接または他の弱いエッジピクセルを介して）接続されている場合に限り、最終的なエッジマップに含まれます [27]。threshold1 未満の勾配を持つピクセルは、ノイズとして破棄されます。

簡単に言えば、threshold2 は「これは間違いなくエッジだ」と判断する基準であり、threshold1 は「強いエッジにつながっていれば、これもエッジかもしれない」と判断する基準と考えることができます。

6.2. 閾値が検出結果に与える影響

閾値の設定は、検出されるエッジの量や質に直接的な影響を与えます。

- 両方の閾値が低い場合: より多くのピクセルがエッジ候補（強いエッジまたは弱いエッジ）として認識されます。これにより、微細なエッジやコントラストの低いエッジも検出されやすくなりますが、同時にノイズもエッジとして拾ってしまう可能性が高まります [28]。結果として、エッジが多く、ノイズの多いエッジマップになる傾向があります。
- 両方の閾値が高い場合: 強い勾配を持つピクセルのみがエッジとして認識されるため、ノイズの影響を受けにくく、クリーンなエッジマップが得られやすくなります。しかし、弱いエッジや細部のエッジが見逃される可能性があります [28]。また、threshold1 が高すぎると、弱いエッジ部分での連結が途切れ、エッジが断片的になるリスクがあります。
- **threshold2 と threshold1 の差（ギャップ）:**
    - ギャップが小さい場合: ヒステリシス閾値処理の効果が薄れ、単一閾値処理に近い挙動になります。ノイズに対する耐性が低下する可能性があります。
    - ギャップが大きい場合: より多くの弱いエッジが強いエッジに接続される機会が増えるため、ノイズに対する耐性が向上し、輝度変化の大きいエッジと小さいエッジが混在する場合でも、エッジの連結性が保たれやすくなります。ただし、意図しない弱いエッジが強いエッジに接続されてしまい、余分なエッジが含まれる可能性もあります。

これらの影響を理解するために、同じ入力画像に対して異なる閾値の組み合わせ（例：(30, 90),

(100, 200), (50, 200) など）を適用し、その結果を比較することが非常に有効です [28]。

6.3. 閾値設定の指針

最適な閾値は画像の内容や目的に依存するため、万能な値は存在しません [27]。しかし、以下の指針が調整の助けになります。

- 推奨比率: Canny自身は、上限閾値 (threshold2) と下限閾値 (threshold1) の比率を 2:1 から 3:1 の間に設定することを推奨しています [37]。これは、パラメータ調整の出発点として有効です。
- 調整戦略: まず推奨比率（例：3:1）で閾値を設定します。次に、threshold2 を調整して、検出したい最も重要な（強い）エッジが捉えられるようにします。その後、threshold1 を調整して、弱いエッジ部分の連結性を制御します。threshold1 を下げるとより多くの弱いエッジが接続され、上げると接続が途切れやすくなります。このプロセスは試行錯誤を伴うことが多く、結果を視覚的に確認しながら最適な値を見つけることが重要です [24]。
- 画像依存性: 照明条件、ノイズレベル、画像のコントラスト、対象物の複雑さなどによって最適な閾値は大きく変動します [27]。特定のアプリケーションで安定した性能を得るためには、対象となる画像の特性を考慮した上で閾値を決定する必要があります。
- 自動閾値設定（参考）: 基本的な cv2.Canny 関数では手動で閾値を設定しますが、より高度な応用では、画像のヒストグラムや統計的性質に基づいて閾値を自動的に決定する手法（例：大津の二値化で用いられる考え方を応用するなど）も研究されています [5]。しかし、Cannyの2つの閾値を完全に自動化するのは単純ではなく、手動調整が依然として一般的なアプローチです。

パラメータ調整、特に閾値の選択は、理論的なガイドラインを参考にしつつも、最終的には対象画像に対する実験と評価を通じて行われる経験的なプロセスです [24]。この試行錯誤の必要性は、特に照明条件などが変動する実世界のアプリケーションにおいて、固定閾値を用いた基本的なCannyアルゴリズムの限界を示唆することもあります。そのような状況では、適応的な閾値決定手法や、あるいは深層学習ベースのエッジ検出器など、より高度なアプローチが必要になる場合があります。

6.4. その他のパラメータ (apertureSize, L2gradient)

cv2.Canny 関数には閾値以外にも調整可能なパラメータがあります。

- **apertureSize:** 内部で使用されるSobelカーネルのサイズ（3, 5, または 7）を指定します [27]。値が大きいほど、より広い範囲のピクセルを考慮して勾配を計算するため、ノイズの影響をいくらか受けにくくなる可能性がありますが、エッジの局在性（位置精度）がわずかに低下することがあります [31]。通常はデフォルト値の 3 で十分な場合が多いです。
- **L2gradient:** 勾配の大きさを計算する際に、より精度の高いL2ノルム ($\sqrt{G_x^2 + G_y^2}$) を使うか（True）、計算が速いL1ノルム ($|G_x| + |G_y|$) を使うか（False）を指定します [27]。L2ノルムの方が理論的には正確ですが、計算コストが若干高くなります。多くの場合、L1ノルム（デフォルトの False）でも視覚的な結果に大きな差はなく、速度面で有利です。

これらのパラメータは閾値ほど結果に劇的な影響を与えることは少ないですが、特定の状況下では調整が有効な場合があります。

7. CannyとSobel、Laplacianフィルタの比較

Cannyアルゴリズムは強力ですが、常に唯一の選択肢ではありません。状況によっては、よりシンプルなSobelフィルタやLaplacianフィルタが適している場合もあります。ここでは、これら3つの手法を比較し、それぞれの特性と使い分けについて解説します。

7.1. Sobelフィルタ：概念とユースケース

- 概念: Sobelフィルタは、画像の輝度値の1次微分を近似することでエッジを検出します [1]。具体的には、3x3（またはそれ以上）の畳み込みカーネル（Sobelオペレータ）を用いて、画像の水平方向（Gx）と垂直方向（Gy）の勾配を計算します [24]。これらの勾配の大きさを組み合わせることで、エッジの強度を表現します。
- 特性: アルゴリズムが比較的単純で、計算が高速です [34]。Laplacianフィルタよりはノイズの影響を受けにくいですが、Cannyほど頑健ではありません [34]。検出されるエッジは、Cannyに比べて太くなる傾向があります [36]。X方向とY方向の勾配を別々に得られるため、エッジの方向に関する情報が必要な場合に有用です。
- ユースケース: 処理速度が重要視され、最高の精度が要求されない場面に適しています [34]。また、勾配の方向情報が後続の処理で必要な場合にも利用されます。Cannyアルゴリズム内部でも勾配計算のためにSobelオペレータが使われていることからもわかるように、他の高度なアルゴリズムの構成要素としても重要です [27]。
- **OpenCV**での実装: cv2.Sobel() 関数を使用します [24]。勾配計算では負の値も生じるため、出力画像のデータ型を cv2.CV_64F のような符号付きの浮動小数点型に設定し、cv2.convertScaleAbs() を用いて絶対値を取り、表示可能な uint8 型に変換するといった配慮が必要です [25]。

7.2. Laplacianフィルタ：概念とユースケース

- 概念: Laplacianフィルタは、画像の輝度値の2次微分（ラプラシアン）を計算することでエッジを検出します [24]。輝度変化が最も急峻な点（2次微分のゼロ交差点）をエッジとして捉えます。方向性を持たない（等方的）ため、あらゆる方向のエッジを均等に検出します。
- 特性: 細かいディテールやシャープなエッジを強調する能力に長けています [34]。しかし、2次微分はノイズ成分を強く増幅するため、ノイズに対して非常に敏感です [34]。計算自体は高速です [34]。検出されるエッジは細いですが、ゼロ交差の性質上、時に二重線として現れることもあります。
- ユースケース: ノイズが少ない画像で、細かいエッジやコーナーを検出したい場合に適しています。画像の鮮鋭度（シャープネス）を評価する指標としても利用されるため、オートフォーカス機能などにも応用されます。通常、適用前に強力なノイズ除去（例：ガウシアンブラー）を行うことが推奨されます [34]。
- **OpenCV**での実装: cv2.Laplacian() 関数を使用します [24]。Sobelと同様に、計算結果を適切に扱うためにデータ型の管理（例：cv2.CV_64F への一時的な変換と cv2.convertScaleAbs() による uint8 への変換）が必要です [25]。

7.3. 比較分析：Canny vs. Sobel vs. Laplacian

これらの3つの手法は、精度、ノイズ耐性、計算コストなどの面で異なるトレードオフを持っています。

- 精度と局在性: 一般的に、Cannyが最も高い精度と良好なエッジ局在性（エッジ位置の正確さ）

を提供します。これは、非極大値抑制による細線化とヒステリシス閾値処理による偽エッジ除去の効果です[1]。Sobelはそれなりに良好ですが、Cannyほどではありません。Laplacianはシャープなエッジには強いですが、ノイズが多いと精度が低下します。

- ノイズ感度: Laplacianが最もノイズに敏感です。Sobelは中程度、Cannyは初期のガウシアンブラーとヒステリシス閾値処理により、最もノイズに強い（耐性がある）と言えます[1]。この関係性は、アルゴリズムの複雑さと計算コストとの間に逆相関があることを示唆しています。最も単純なLaplacian（2次微分）は最速ですがノイズに最も弱く、最も複雑なCanny（多段階処理）は最も遅いですがノイズに最も強いです。
- 計算コスト: Cannyは多段階の処理を含むため、計算コストが最も高くなります[34]。SobelとLaplacianは、単純な畳み込み演算が中心であるため、大幅に高速です[34]。
- エッジの太さ: Cannyは設計上、1ピクセル幅の細いエッジを生成します[27]。Sobelは比較的太いエッジを生成する傾向があります[36]。Laplacianはシャープで細いエッジを生成しますが、二重線になることもあります。
- パラメータ調整: Cannyは2つの閾値を適切に設定する必要があり、調整がやや複雑です[34]。SobelとLaplacianは、カーネルサイズや出力データ型といった、比較的調整が容易なパラメータが主です。

7.4. アルゴリズム比較表

特徴	Canny	Sobel	Laplacian
原理	多段階 (1次微分 + NMS + ヒステリシス)	1次微分近似	2次微分近似
長所	高精度、ノイズ耐性良好	単純、高速、勾配方向情報	シャープなエッジ検出、高速
短所	複雑、低速、パラメータ調整が必要	精度はCannyに劣る、エッジが太め	ノイズに非常に敏感
計算コスト	高	低	低
ノイズ耐性	良い	中程度	悪い
主な用途	一般的な高品質エッジ検出	高速な検出、勾配方向解析	細部検出、焦点評価

この比較から、SobelやLaplacianが単にCannyの「劣った」代替品ではないことがわかります。それぞれに独自の利点と適切な応用場面が存在します。特にSobelはCannyアルゴリズムの内部でも利用されており[27]、これらの基本的なフィルタを理解することは、より高度な手法を理解する上でも価値があります。

8. 境界検知における前処理の重要性

エッジ検出アルゴリズム、特に勾配ベースの手法を効果的に適用するためには、適切な前処理が不可欠です。前処理は、元の画像に含まれるノイズを除去したり、エッジ検出に適した形式に画像を変換したりすることで、最終的な結果の品質と信頼性を大幅に向上させます。

8.1. 前処理の重要性

実世界の画像は、センサノイズ、圧縮によるアーティファクト、照明のむら、不要なテクスチャなど、エッジ検出を妨げる可能性のある様々な要素を含んでいます [7]。エッジ検出アルゴリズム、特に微分演算を用いるものは、これらのノイズ成分に非常に敏感であり、ノイズを誤ってエッジとして検出したり、逆に本来のエッジがノイズに埋もれて検出できなかったりする原因となります [1]。
したがって、前処理は単なるオプションの改善ステップではなく、多くの場合、信頼性の高いエッジ検出を実現するための必須のステップです [4]。画像処理パイプラインにおいて、エッジ検出の前段階でこれらの不要な情報を取り除き、エッジの特徴を強調することが、後続の処理の成功に繋がります [29]。

8.2. グレースケール変換

多くのエッジ検出アルゴリズムは、輝度（明るさ）の変化に基づいて動作します。カラー画像の場合、RGBの3つのチャンネルそれぞれで輝度変化を計算することも可能ですが、通常は色情報はエッジ検出には不要であり、処理を複雑にする可能性があります [30]。
そのため、一般的には、エッジ検出を行う前にカラー画像をグレースケール画像（単一の輝度チャンネルを持つ画像）に変換します [17]。これにより、処理するデータ量が1/3になり、計算が単純化され、効率が向上します。OpenCVの cv2.Canny などの多くのエッジ検出関数は、入力としてグレースケール画像を期待しています [27]。
グレースケール変換は、OpenCVの cv2.cvtColor() 関数を用いて容易に行うことができます。

Python

```
img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)
```

[17]

8.3. ノイズ除去技術

前述の通り、ノイズはエッジ検出の精度を著しく低下させる要因です [4]。特に、勾配（微分）計算はノイズ成分を増幅する性質があるため、ノイズ除去は非常に重要です [1]。OpenCVには、いくつかのノイズ除去フィルタが用意されています。

- ガウシアンブラー **(Gaussian Blur):** Cannyエッジ検出の前処理として最も一般的に使用されるフィルタです [1]。画像を滑らかにし、主に高周波ノイズを低減します [21]。カーネルサイズ（例：(5, 5)）と標準偏差（sigmaX）でぼかしの度合いを調整できます [21]。カーネルサイズが大きいほど、より強くぼかされますが、細部のエッジも失われる可能性があります。cv2.GaussianBlur() 関数で適用します [21]。
- メディアンブラー **(Median Blur):** 特に「ごま塩ノイズ」（salt-and-pepper noise）のような、突発的なピクセル値を持つノイズに対して効果的です [21]。注目ピクセルの値を、その近傍領域内のピクセル値の中央値で置き換えます。平均値フィルタやガウシアンフィルタと比較して、エッジを比較的よく保持しながらノイズを除去できる場合があります [21]。cv2.medianBlur() 関数で

適用します [21]。

- バイラテラルフィルタ **(Bilateral Filter):** ノイズを除去しつつ、エッジを保持する能力が高いフィルタです [21]。ピクセル間の距離だけでなく、輝度値の差も考慮して重み付けを行うため、エッジ部分をぼかさずに平滑化できます。ただし、他のフィルタに比べて計算コストが高いという欠点があります [21]。cv2.bilateralFilter() 関数で適用します [21]。

どのノイズ除去フィルタを選択するかは、画像に含まれるノイズの種類や、エッジ保持の重要度、許容される計算コストによって異なります。実際にノイズのある画像に対して、フィルタ適用前後でエッジ検出を行い、その結果を比較することで、ノイズ除去の効果を確認できます [42]。

8.4. 画像の種類に応じたアルゴリズムとパラメータの選択

最適なエッジ検出アルゴリズムとパラメータ設定は、処理対象の画像の特性に大きく依存します [28]。また、アプリケーションが要求する速度と精度のトレードオフも考慮する必要があります。

- アルゴリズム選択の考慮事項:
  - ノイズが多い画像: Cannyアルゴリズムが、組み込みのノイズ除去とヒステリシス閾値処理により、一般的に最もロバストです [1]。Sobelも前処理で適切にブラーをかければ使用可能ですが、Laplacianはノイズに非常に弱いため、通常は避けるべきです [34]。
  - 細かいディテールが重要な画像: ノイズが少なければ、Laplacianがシャープなエッジや細部を捉えるのに適している場合があります [34]。Cannyでも、apertureSize を小さくし、閾値を慎重に調整することで対応できる可能性があります [31]。
  - 速度が最優先されるアプリケーション: SobelやLaplacianはCannyよりも大幅に高速です [34]。
- パラメータ調整の考慮事項:
  - ノイズが多い画像: ガウシアンブラーのカーネルサイズを大きくする必要があるかもしれません。Cannyの閾値は、ノイズを拾わないように比較的高めに設定する必要がある場合があります [28]。
  - コントラストが低い画像: エッジの輝度変化が小さいため、Cannyの閾値を低めに設定しないと、エッジが検出されない可能性があります [28]。
  - テクスチャが複雑な画像: 布地や木目のようなテクスチャを持つ画像では、意図しないテクスチャのエッジが多数検出されることがあります。この場合、より強力なブラーを適用するか、エッジ検出ではなくセグメンテーションなど他のアプローチを検討する必要があるかもしれません。
- その他の前処理: 上記以外にも、画像のコントラストを改善するためにヒストグラム平坦化を適用したり、特定の形状のノイズを除去するためにモルフォロジー演算（膨張・収縮）[47] を行ったりすることが有効な場合があります。また、まれに、グレースケール変換では失われてしまう色情報がエッジ検出の手がかりとなる場合は、HSV色空間など他の色空間を利用することも考えられます [48]。

重要なのは、前処理の選択とそのパラメータ設定が、後続のエッジ検出アルゴリズムとそのパラメータ設定と密接に関連しているという点です。例えば、前処理で強いブラーをかけると、ノイズは減りますがエッジも鈍化するため、Cannyの閾値を低めに設定する必要が生じるかもしれません。逆に、ブラーが弱い場合は、ノイズを避けるために閾値を高めに設定する必要があるでしょう。このように、最適な結果を得るためには、前処理とエッジ検出のステップを個別にではなく、パイプライン全体として

捉え、相互の影響を考慮しながら調整していく必要があります。

9. 結論と推奨事項

9.1. 主要技術のまとめ

本レポートでは、画像処理における境界検知（エッジ検出）の基本的な概念から、PythonとOpenCVを用いた具体的な実装方法までを解説しました。

- エッジは画像の構造を理解する上で基本的な特徴であり、物体認識や品質検査、自動運転など多様な分野で応用されています。
- OpenCVは、Canny、Sobel、Laplacianといった主要なエッジ検出アルゴリズムを含む、豊富な画像処理機能を提供する強力なライブラリです。
- Cannyアルゴリズムは、ノイズ除去、勾配計算、非極大値抑制、ヒステリシス閾値処理という複数のステップを経て、高精度かつノイズに強いエッジ検出を実現します。
- SobelフィルタとLaplacianフィルタは、それぞれ1次微分と2次微分に基づいてエッジを検出する、より高速でシンプルな手法ですが、精度やノイズ耐性ではCannyに劣る場合があります。
- エッジ検出の成功には、グレースケール変換やノイズ除去といった適切な前処理と、アルゴリズムのパラメータ（特にCannyの閾値）の慎重な調整が不可欠です。

9.2. 実装に向けた推奨事項

これからPythonとOpenCVを用いて境界検知を実装する際には、以下の点を考慮することをお勧めします。

1. **Canny**から始める: 一般的な用途には、まずCannyアルゴリズム (cv2.Canny) を試すのが良いでしょう。その精度とノイズ耐性は多くの場合、良好なベースラインとなります。
2. 前処理を必ず行う: 画像をグレースケールに変換し (cv2.cvtColor)、ガウシアンブラー (cv2.GaussianBlur) を適用することを標準的な手順とします。
3. 閾値は実験的に調整する: Cannyの閾値 (threshold1, threshold2) は、まず推奨される比率（2:1～3:1）から始め、処理したい画像で実際に結果を見ながら、試行錯誤を通じて最適な値を見つけます。
4. 他のアルゴリズムも検討する: 処理速度が極めて重要である場合や、特定のエッジ特性（例：勾配方向、微細なディテール）を捉えたい場合は、Sobel (cv2.Sobel) やLaplacian (cv2.Laplacian) の利用も検討します。ただし、これらの手法を用いる場合もノイズ除去は重要です。
5. 画像と目的に合わせた最適化: 対象とする画像の種類（ノイズレベル、コントラストなど）やアプリケーションの最終的な目的に応じて、前処理手法（メディアンブラーなど他のフィルタも試す）、アルゴリズム、パラメータの組み合わせを実験的に探求します。

9.3. さらなる探求

本レポートで扱った手法は古典的ながら強力ですが、エッジ検出の分野は進化を続けています。より高度な技術や関連分野を探求することで、さらに複雑な課題に対応できる可能性があります。

- 高度なエッジ検出手法: 特に複雑な自然画像に対しては、ランダムフォレストを用いたStructured Edge Detection (SED) や、深層学習（ディープラーニング）に基づくHolistically-Nested Edge Detection (HED) などの手法が、古典的な手法よりも優れた性能

を示す場合があります。
- 適応的閾値設定: 画像の内容に応じて閾値を自動的に調整するアルゴリズムについて学ぶことで、様々な条件下で安定した結果を得る助けになります。
- 輪郭検出: エッジ検出の後処理として、検出されたエッジピクセルを連結し、閉じた輪郭として抽出する cv2.findContours などの関数を探求することは、物体領域の特定や形状分析に繋がります [6]。

境界検知はコンピュータビジョンの基礎であり、ここで解説した技術と概念を理解することは、より高度な画像解析タスクに取り組むための重要な一歩となるでしょう。

引用文献

1. Edge Detection Using OpenCV | LearnOpenCV #, 4月 14, 2025にアクセス、https://learnopencv.com/edge-detection-using-opencv/
2. 画像処理の基礎｜Pythonで学ぶエッジ検出手法 - Hakky Handbook, 4月 14, 2025にアクセス、https://book.st-hakky.com/data-science/edge-detection-in-image-processing-python/
3. Laplacian Sobel and Canny Edge Detection using OpenCV Python - YouTube, 4月 14, 2025にアクセス、https://www.youtube.com/watch?v=a7m-9EFEfU0
4. 「基礎から学ぶ医療・科学分野で活用される、OpenCVによる画像認識技術」第2回 Canny法によるエッジ検出の仕組みについて実例を用いて解説 - eSOL, 4月 14, 2025にアクセス、https://blog.esol.co.jp/embedded/image-recognition-technology_02
5. 【初心者向け】画像処理｜領域分割と特徴抽出の基本を徹底解説 - Hakky Handbook, 4月 14, 2025にアクセス、https://book.st-hakky.com/data-science/image-processing-region-segmentation/
6. 【Python】画像のノイズを減らして二値化し、輪郭を抽出する - LabCode, 4月 14, 2025にアクセス、https://labo-code.com/python/drawcontours-image/
7. 画像処理のエッジ検出とは？前処理や検査、事例10選 - FAプロダクツ, 4月 14, 2025にアクセス、https://jss1.jp/column/column_76/
8. OpenCV.jsとは？JavaScript版OpenCVの概要とその特徴 - 株式会社一創, 4月 14, 2025にアクセス、https://www.issoh.co.jp/tech/details/3608/
9. エッジ検出を完全理解！主要3フィルタとCanny法の実装手順を解説 - Tech Teacher, 4月 14, 2025にアクセス、https://www.tech-teacher.jp/blog/edge-detection/
10. 画像認識とは何か？しくみや応用分野、利点と課題を解説 - トリプルアイズ, 4月 14, 2025にアクセス、https://www.3-ize.jp/triple-blog/list/3581/
11. 画像認識とは？AI活用の仕組みや画像解析との違い、精度・種類・活用方法・注意点を徹底解説！, 4月 14, 2025にアクセス、https://ai-market.jp/purpose/image-recognition/
12. AI外観検査とは？画像処理の仕組みや事例・メリット、導入費用相場まで徹底解説, 4月 14, 2025にアクセス、https://products.sint.co.jp/aisia-ad/blog/ai-visual-inspection
13. AIの物体検出とは？YOLO・CNNなど機械学習による画像認識・最新事例徹底解説！ - AI Market, 4月 14, 2025にアクセス、https://ai-market.jp/purpose/image-recognition-object_detection/

14. OpenCVを使うスケルトンプログラム #Python - Qiita, 4月 14, 2025にアクセス、https://qiita.com/hsgucci/items/4bf1ef324837665f6b60
15. 【Pythonで画像処理をはじめよう】OpenCVの使い方を解説 - Udemy メディア, 4月 14, 2025にアクセス、https://udemy.benesse.co.jp/development/python-work/opencv.html
16. OpenCVを入門者向けに解説！OpenCVでできることやPythonでの画像処理にオススメの理由を紹介, 4月 14, 2025にアクセス、https://freelance.shiftinc.jp/column/open-cv/
17. Pythonで簡単画像加工！OpenCVでできる画像処理入門, 4月 14, 2025にアクセス、https://yyoshiblog.com/python-opencv-intro/
18. PythonのOpenCVとは？画像加工に必須の知識・テクニックを徹底解説 - NEUTRAL, 4月 14, 2025にアクセス、https://saas.n-works.link/programming/python/whatispythonsopencv
19. 【Python×OpenCV】初めてのOpen CV(画像処理ライブラリ)ガイド - ケイエルブイ, 4月 14, 2025にアクセス、https://www.klv.co.jp/corner/python-opencv-what-is-opencv.html
20. Python画像認識とは？役割や使い方まで解説 - AIsmiley, 4月 14, 2025にアクセス、https://aismiley.co.jp/ai_news/what-is-a-python/
21. 【Python・OpenCV】4種類のノイズ除去フィルターの使い方と特徴について - codevace, 4月 14, 2025にアクセス、https://www.codevace.com/py-opencv-smoothing/
22. OpenCVとは？できることや特徴、開発に役立つ使い方とメリット・デメリットをわかりやすく解説, 4月 14, 2025にアクセス、https://gen-ai-media.guga.or.jp/glossary/opencv/
23. OpenCV のノイズ除去について (Blur) - メカトロニクスエンジニアのブログ, 4月 14, 2025にアクセス、https://iryachi.stars.ne.jp/opencv-blur/
24. OpenCVを用いたエッジ検出 #Python - Qiita, 4月 14, 2025にアクセス、https://qiita.com/kakuteki/items/55c00e33cad63f9e440f
25. 画像処理初心者がOpenCVを使ってみた(2) #Python - Qiita, 4月 14, 2025にアクセス、https://qiita.com/HIsui0921/items/4c99f1c167efc9ae9149
26. OpenCVとは？基本概念から応用まで解説 - TechSuite AI Blog, 4月 14, 2025にアクセス、https://techsuite.biz/opencv%E3%81%A8%E3%81%AF%EF%BC%9F%E5%9F%BA%E6%9C%AC%E6%A6%82%E5%BF%B5%E3%81%8B%E3%82%89%E5%BF%9C%E7%94%A8%E3%81%BE%E3%81%A7%E8%A7%A3%E8%AA%AC/
27. Canny Edge Detection - OpenCV, 4月 14, 2025にアクセス、https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html
28. OpenCV Edge Detection (cv2.Canny) - PyImageSearch, 4月 14, 2025にアクセス、https://pyimagesearch.com/2021/05/12/opencv-edge-detection-cv2-canny/
29. 画像処理入門講座 : OpenCVとPythonで始める画像処理 | POSTD, 4月 14, 2025にアクセス、https://postd.cc/image-processing-101/
30. 機械学習のためのOpenCV入門 #Python - Qiita, 4月 14, 2025にアクセス、https://qiita.com/icoxfog417/items/53e61496ad980c41a08e
31. OpenCVの画像処理（アナログタコメーターのデジタル化） - 株式会社アルゴ, 4月 14, 2025にアクセス、

https://www.argocorp.com/software/sdk/ICImagingControl/Sample_program/dotnet_35/distance-and-angle-measurements.html
32. 【図解】画像処理の6つの種類をご紹介。画像処理メーカー3社 - FAプロダクツ, 4月 14, 2025にアクセス、https://fa-products.jp/column/image-processing-types/
33. 【Python・OpenCV】Cannyエッジ検出器による輪郭抽出(cv2 ..., 4月 14, 2025にアクセス、https://www.codevace.com/py-opencv-canny/
34. 【Python・OpenCV】ラプラシアン フィルターによるエッジ検出 ..., 4月 14, 2025にアクセス、https://www.codevace.com/py-opencv-laplacian/
35. Sobel Derivatives - OpenCV Documentation, 4月 14, 2025にアクセス、https://docs.opencv.org/4.x/d2/d2c/tutorial_sobel_derivatives.html
36. エッジ検出（Sobel，Laplacian，Canny）| OpenCV.jp, 4月 14, 2025にアクセス、http://opencv.jp/opencv2-x-samples/edge_detection
37. Canny Edge Detector - OpenCV, 4月 14, 2025にアクセス、https://docs.opencv.org/3.4/da/d5c/tutorial_canny_detector.html
38. Canny Edge Detector — OpenCV Documentation, 4月 14, 2025にアクセス、https://vovkos.github.io/doxyrest-showcase/opencv/sphinx_rtd_theme/page_tutorial_canny_detector.html
39. Canny Edge Detector - OpenCV Documentation, 4月 14, 2025にアクセス、https://docs.opencv.org/4.x/da/d5c/tutorial_canny_detector.html
40. Canny Edge Detection - OpenCV Documentation, 4月 14, 2025にアクセス、https://docs.opencv.org/3.4/d7/de1/tutorial_js_canny.html
41. 特徴検出 — opencv v2.1 documentation, 4月 14, 2025にアクセス、http://opencv.jp/opencv-2.1/cpp/feature_detection.html
42. Detect Edges with OpenCV and Python | Computer Vision Tutorial - YouTube, 4月 14, 2025にアクセス、https://www.youtube.com/watch?v=6cXV8dhNu50
43. OpenCV (Python3) を利用したエッジ検出, 4月 14, 2025にアクセス、https://axa.biopapyrus.jp/ia/opencv/edge.html
44. Edge Detection Using OpenCV Explained. - YouTube, 4月 14, 2025にアクセス、https://www.youtube.com/watch?v=J1KGGsvxY0w
45. Pythonで綺麗に輪郭抽出する方法【OpenCVでエッジ検出、画像処理】- MONOwith, 4月 14, 2025にアクセス、https://monowith.com/image-processing/opencv-extractedge/
46. Python+OpenCVを利用した二値化処理 - ドローンBiz, 4月 14, 2025にアクセス、https://dronebiz.net/tech/opencv/binarize
47. OpenCV 入門：画像処理・画像認識・機械学習の実装を徹底解説（全実装コード公開）, 4月 14, 2025にアクセス、https://www.codexa.net/opencv_python_introduction/
48. Python で OpenCV 備忘録 (575) - 中級編 #ChatGPT - Qiita, 4月 14, 2025にアクセス、https://qiita.com/maskot1977/items/ed4c5ca60bd2fd0a30da