

# カラーコーン検出アルゴリズムについて

kuma003 (<https://github.com/kuma003>, F.T.E. 14th.)

2025年11月9日

## 目次

1	はじめに	2
2	色を取り扱う上で	2
2.1	色空間	2
2.2	色の演算	4
3	色相を用いる手法	4

# 1 はじめに

CanSat では、最終的にゴールへ至るためにゴールコーンを認識する必要がある。そのため、カメラの入力映像からカラーコーンを何らかの手法で検出することが求められる。

そこで本稿ではカラーコーン検出アルゴリズムについて、古典的な色相を用いる場合、逆投影法を用いる場合、そして機械学習を用いる場合の 3 つの手法を紹介し、それぞれの手法の特徴や精度についてまとめる。主にアルゴリズムについて焦点をあて、プログラムの詳細には立ち入らないこともあるため、適宜リファレンスを参照されたい。

テストデータとして用いるデータについては筆者が撮影したものを用いるが、オープンにしづらい画像もあるため、比較的オープンにしやすい画像について、public\_dataset フォルダにまとめている。以下に用いるサンプルで用いる画像を示す。



図 1: サンプルで用いる画像。

# 2 色を取り扱う上で

## 2.1 色空間

カラーコーン検出において、まず重要なのが色の取り扱いである。一般的に、画像は RGB で表現されるのが一般的であるのは周知の事実である。色を RGB で指定することを **RGB 空間**と呼ぶ。

各色の強度を 0 から 255 までの 256 段階 (8 ビット) で表現することが多いが、この場合、 $(0, 0, 0)$  が黒、 $(255, 255, 255)$  が白、 $(255, 0, 0)$  が赤、 $(0, 255, 0)$  が緑、 $(0, 0, 255)$  が青を表す。もしくは、16 進数を用いて `0x000000`、`0x00FF00` のよ

うに表現することも多い。また、さらにアルファチャンネルという透明度を表す成分を加えた RGBA で表現されることもある。この場合、0x00000000 が完全に透明な黒、0xFFFFFFFF が不透明な白を表す。

画像の処理で頻繁に用いられる OpenCV というライブラリにおいても、画像は基本この三原色で表現される。ただし、OpenCV では画像は BGR の順番で表現されるので注意が必要である<sup>\*1</sup>。そのため、例えば OpenCV で処理した画像を matplotlib で表示すると色が変わって見えることがあるが、これは BGR 表記に起因するものであり、そのときは RGB に変換する必要がある。

RGB 表記は色の三原色に基づくものであるが、あまり直観的ではない。画像処理をする上では、色相 (Hue), 彩度 (Saturation), 明度 (Value) の 3 つの成分で色を表現する HSV 空間がよく用いられる。ペイントエディタなどでも直観的に色を選択できるようにな、図 2 のような HSV の色相環が用いられることが多い。

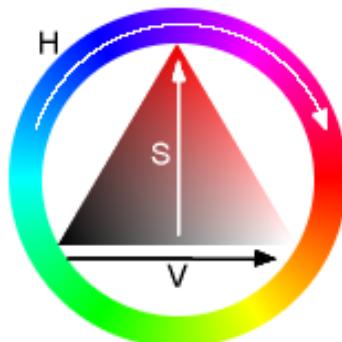


図 2: HSV 色空間のイメージ図。(出典: [https://commons.wikimedia.org/wiki/File:Hsv\\_sample.png](https://commons.wikimedia.org/wiki/File:Hsv_sample.png))

実用上でも、その場の明るさによって明度が変化することはあるが、色相は比較的变化しづらいため、色相を用いた画像認識が有用である。なお、OpenCV では色相のみ範囲が 0 から 179 までの 180 段階で表現されることに注意が必要である<sup>\*2</sup>。

---

<sup>\*1</sup> OpenCV が開発された初期のころ、BGR 形式が一般的だったためらしい。

The reason the early developers at OpenCV chose BGR color format is that back then BGR color format was popular among camera manufacturers and software providers. E.g. in Windows, when specifying color value using COLORREF they use the BGR format 0x00bbggrr. (Why does OpenCV use BGR color format ?より)

<sup>\*2</sup> 参考:[https://docs.opencv.org/4.x/d9d/tutorial\\_py\\_colorspaces.html](https://docs.opencv.org/4.x/d9d/tutorial_py_colorspaces.html)。ソフトウェアによって異なり、0 から 359 までの範囲で表現されることもある。

## 2.2 色の演算

ここについては、本番には不要であるが、画像処理の解析を行う上で知っておくと便利な演算について説明する。読み飛ばしても差し支えない。

簡単のために、各ピクセルごとに黒(0)か白(1)の2値で表現される画像を考える。このような画像を**二値画像**と呼び、このような画像を作成することを**二值化**と呼ぶ。このような二値画像に対しては、AND, OR, NOTなどの論理演算を行うことができる。

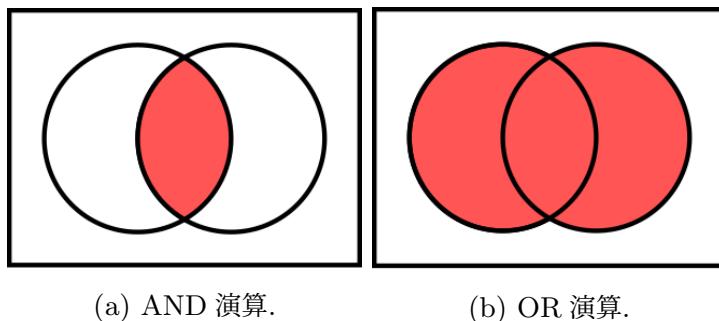


図 3: AND 演算と OR 演算 (出典: <https://commons.wikimedia.org/wiki/File:Venn0001.svg>, <https://commons.wikimedia.org/wiki/File:Venn0111.svg>).

例えば、必要な領域を1、それ以外を0とした二値画像を用意する（このような画像を**マスク画像**と呼ぶ）。このマスク画像と元の画像に対してAND演算を行うことで、必要な領域だけを抽出することができる。また、マスク画像を複数用意し、OR演算を行うことで、複数の領域をまとめて抽出することもできる。本番で用いることは少ないが、処理の結果どこを検出したのかを可視化して解析する際には非常に有用である。

## 3 色相を用いる手法

色相を用いる手法は、カラーコーンの色相の範囲をあらかじめ決めておき、その範囲内にあるピクセルをカラーコーンとして検出することで実現される。非常に実装が容易であり、計算の負荷がとても低いため、堅実に動作する手法である。