Pythonを用いて画像処理を行う

OpenCV

インテルが開発・公開したオープンソースのコンピュータビジョン向けライブラリのこと.
画像処理・画像解析および機械学習等の機能を持つC/C++, Java, Python、MATLAB用ライブラリ,
プラットフォームとしてmacOSやFreeBSD等全てのPOSIXに準拠したUnix系OS, Linux, Windows,
Android、iOS等をサポートしている.

- 画像処理 (Image Processing)
  - 勾配、エッジ、コーナー (Gradients, Edges and Corners)
  - サンプリング、補間、幾何変換 (Sampling, Interpolation and Geometrical Transforms)
  - モルフォロジー（英語版）演算 (Morphological Operations)
  - フィルタと色変換 (Filters and Color Conversion)
  - ピラミッドとその応用 (Pyramids and the Applications)
  - 画像分割、領域結合、輪郭検出 (Image Segmentation, Connected Components and Contour Retrieval)
  - 画像と形状のモーメント (Image and Contour Moments)
  - 特殊な画像変換 (Special Image Transforms)
  - ヒストグラム (Histograms)
  - マッチング (Matching)
  - ラベリング (Labeling) : OpenCV 3.0以降

- 構造解析 (Structural Analysis)
  - 輪郭処理 (Contour Processing)
  - 計算幾何 (Computational Geometry)
  - 平面再分割 (Planar Subdivisions)
- モーション解析と物体追跡 (Motion Analysis and Object Tracking)
  - 背景統計量の累積 (Accumulation of Background Statistics)
  - モーションテンプレート (Motion Templates)
  - 物体追跡 (Object Tracking)
  - オプティカルフロー (Optical Flow)
  - 推定器 (Estimators)

- パターン認識 (Pattern Recognition)
  - 物体検出 (Object Detection)
- カメラキャリブレーションと3次元再構成 (Camera Calibration and 3D Reconstruction)
  - カメラキャリブレーション (Camera Calibration)
  - 姿勢推定 (Pose Estimation)
  - エピポーラ幾何 (Epipolar Geometry)

- 機械学習
  - 単純ベイズ分類器 (Naive Bayes Classifier)
  - k近傍法 (K Nearest Neighbors)
  - サポートベクターマシン (SVM)
  - 決定木 (Decision Trees)
  - ブースティング (Boosting)
  - Random forest (Random forest)
  - EMアルゴリズム (Expectation-Maximization)
  - ニューラルネットワーク (Neural Networks)
- ユーザインタフェース
  - シンプルGUI (Simple GUI)
  - 画像の読み込みと保存 (Loading and Saving Images)
  - ビデオ入出力 (Video I/O)
  - OpenGL/Direct3D相互運用

モジュールのセットアップ

以下のコマンドを実行してモジュールのインストールを行います

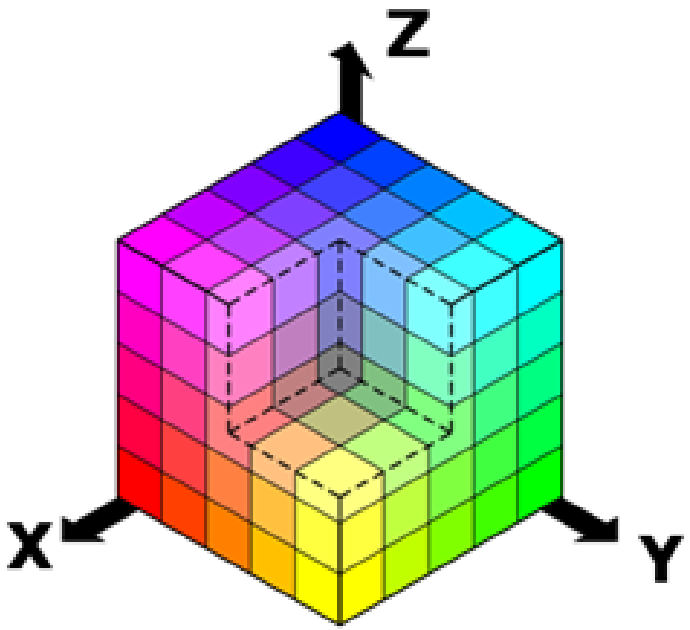
```
$ python3 -m pip install opencv-python
```

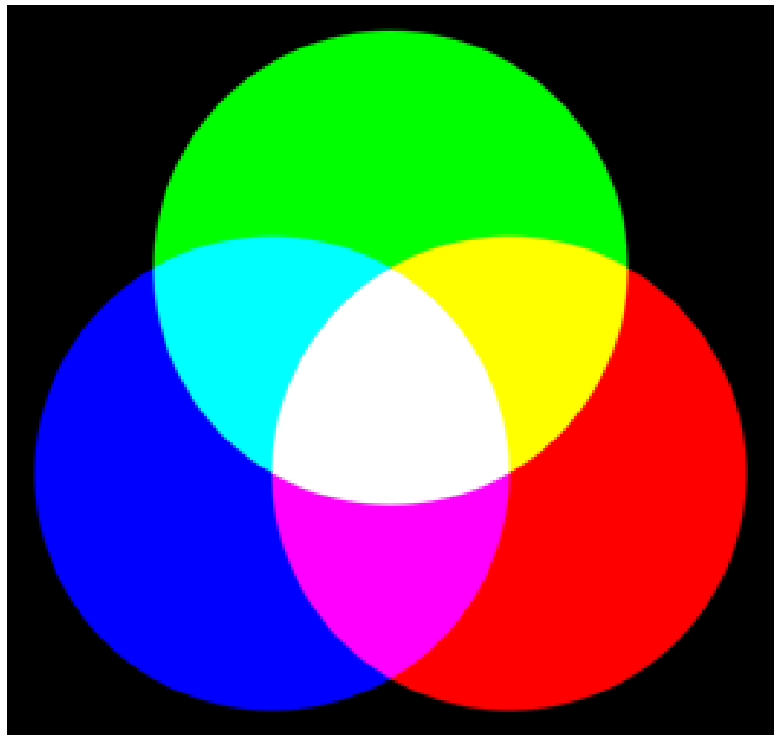
画像データの構造と色空間の話

コンピュータ上で画像ファイルはそれぞれのピクセルごとに配列で格納されていることが多い.
一般的なものはRGB画像で，R（赤），G（緑），B（青）それぞれの色についてその濃さを8bitづつデータとして定義し，1ピクセルあたり24bit長のデータを保持する．これを画像の幅x高さの要素をもつ配列に格納する.
したがって画像の大きさが大きいとデータサイズが大きくなるのに加えて，画像処理における処理時間も長くなる.

RGB色空間

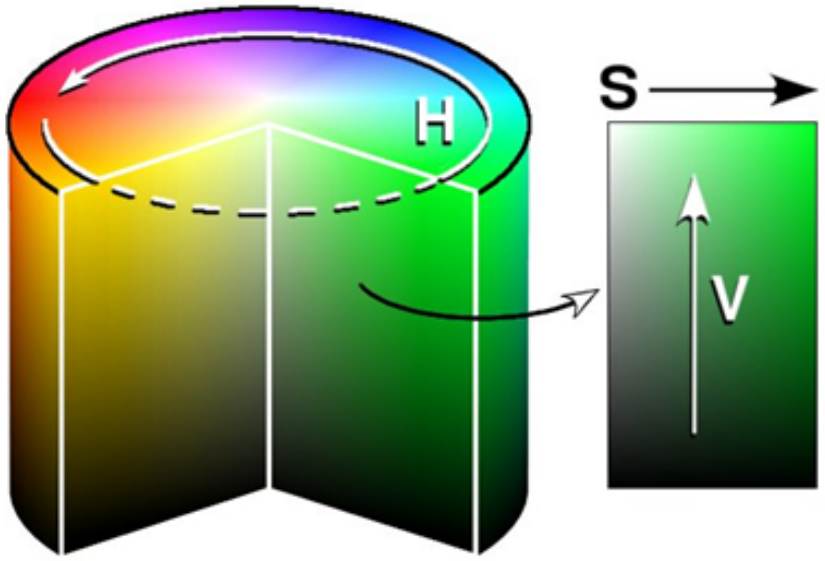
1画素に定義される色をRGBの3要素により表現する.
光の三原色で定義を行い，全部足すと白になる（黒は光がないことを意味する）
1ch/8bitの場合は0～255(0x00～0xFF）の数値で表現する（RGBの3chで24bit）





HSV色空間

色相(Hue)、彩度(Saturation)、明度(Value・Lightness・Brightness)の3要素で色を表現する方法で,
円筒または円錐状の座標系に定義される
円筒座標系を鉛直上方向（V軸側）から見て色相を選択することで色抽出フィルタの実装などが用意になる.



OpenCVで画像を開く

`cv2.imread()` で画像の読み込みを行うことができる.
また，ユーザーインターフェースの機能も用いてウィンドウを開くことができる.

```
import cv2

srcimgFilename = 'sample2.png'
img = cv2.imread(srcimgFilename)     # 画像を読み込んでimgに格納する

cv2.imshow('sample',img)     # 画像を表示するウィンドウを開く
cv2.waitKey(0)               # キー入力を待ち受ける
cv2.destroyAllWindows()      # すべてのウィンドウを閉じる
```

OpenCVで色抽出処理を行う

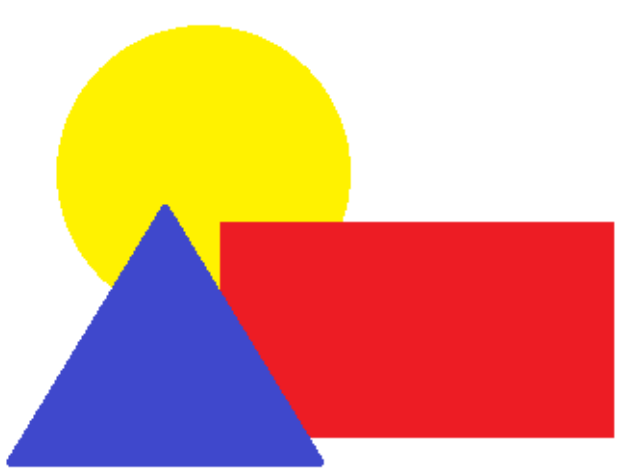
HSV色空間において抽出を行う.
画像処理で色抽出を行う場合，一旦「マスク」と呼ばれる画像を作成して元画像と重ね合わせを行い，
出力画像を得る場合が多い.

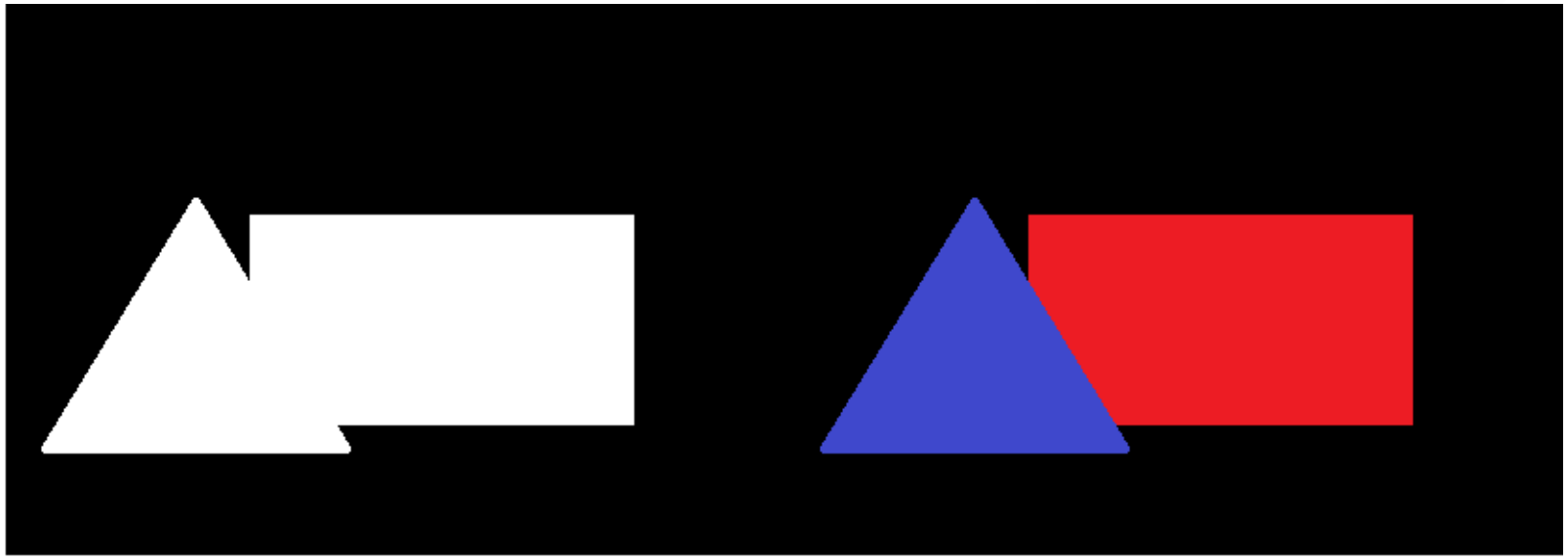
```
import cv2
import numpy as np #numpyを利用する

srcimgFilename = 'sample2.png'
img = cv2.imread(srcimgFilename)     # 画像を読み込んでimgに格納する
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)   # フレームをHSVに変換

lower_blue = np.array([80, 100, 20])
upper_blue = np.array([180, 255, 255])

# 指定した色に基づいたマスク画像の生成
img_mask_blue = cv2.inRange(hsv, lower_blue, upper_blue)
```

マスク処理

マスク処理は特定の部分のみを表示（抽出）し，それ以外の部分を表示しない（黒色画像または白色画像）ようにする画像処理のことを指す．マスク処理を行うためには、表示（抽出）したい部分は、どこなのかを表すマスクをあらかじめ用意する必要があり，1チャネルの画像を用いることが多い．

元画像とマスク，マスク画像





OpenCVによるカメラ制御

- `cv2.VideoCapture()` でカメラキャプチャインスタンスを作成
- `cap.read()` でカメラ画像を読み込み
- `cv2.imshow()` でウィンドウに画像を表示
- `cv2.imwrite()` で画像の書き出しを行う

※teraterm + WinSCPなどで開発している場合にはGUIを利用できないので `cv2.imshow()` は利用できない．VNCから利用するか，ファイルなどに画像を書き出して確認するなど行う．

Raspberry piでのカメラデバイスの扱い方

UVCタイプのカメラはセットアップ不要で利用可能

UVC(USB Video Class)のカメラであればドライバなどのセットアップを行うことなく利用できる.
（古いWebカメラではUVCタイプでないカメラも多く，OSごとにドライバを必要とした．）

昨今販売されているWebカメラでは，Windowsに接続した際にドライバのインストールが不要なカメラであれば多くがLinuxでも利用できる.
※購入時にUVCカメラと明記されていることも多い

UVCカメラの接続確認方法について

USB接続したデバイスを列挙する `lsusb` コマンドが用意されており，USBデバイスが認識されているかどうかの確認を当該コマンドにより確認できる.

```
$ lsusb
Bus 002 Device 002: ID 0bda:0411 Realtek Semiconductor Corp.
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 003: ID 046d:0825 Logitech, Inc. Webcam C270
Bus 001 Device 002: ID 0bda:5411 Realtek Semiconductor Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

また，`/dev/` 以下に `videoX` （`X` は0から始まる数字）でデバイスファイルが作成されるのでこのファイルの有無を確認することでも可能.

カメラを接続する順番によりデバイスファイルのインデックスが変わってしまうため，複数のカメラを利用する場合には都合が悪い．そこでポート番号などでデバイスを識別する方法もある.

> https://dev.classmethod.jp/articles/opencv-device-id-by-port-number/ ”

カメラキャプチャ：Raspberry Piにモニタを繋いで操作をしている場合

VNCの場合も下記のコードでチェックできる

```
import cv2

if __name__ == '__main__':
    cap = cv2.VideoCapture(-1) # 任意のカメラ番号に変更する
    while True:
        ret, frame = cap.read()
        cv2.imshow("camera", frame)

        k = cv2.waitKey(1) # キー入力を待つ
        if k == ord('q'):
            cv2.imwrite('frame.png', frame)
            break     #「q」キーが押されたら終了する

    # キャプチャをリリースして、ウィンドウをすべて閉じる
    cap.release()
    cv2.destroyAllWindows()
```

カメラキャプチャ：SSH/SCP経由で操作を行っている場合

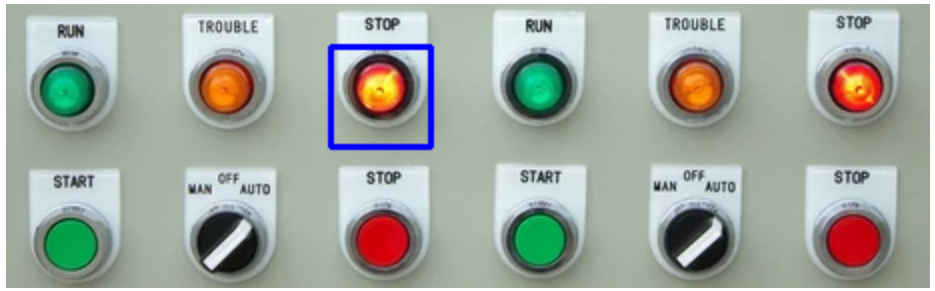
画像が表示されるウィンドウを表示することができないので下記のコードを利用する.

```python
import cv2
import time

if __name__ == '__main__':
    cap = cv2.VideoCapture(-1) # 任意のカメラ番号に変更する
    image = None
    try:
        print("capture start")
        while True:
            ret, frame = cap.read()
            time.sleep(0.01)
            image = frame
    except KeyboardInterrupt:
        cv2.imwrite('frame.png', image)
        pass
    cap.release()
    print("done")
```

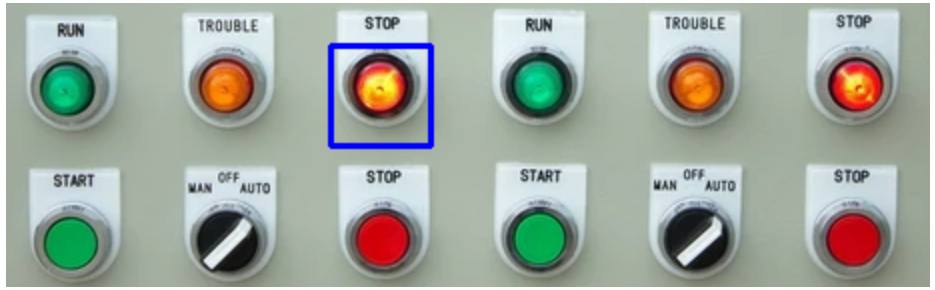
特定の領域を抽出（切り出し）する

画像データのうち操作や注目する対象として選ぶ領域であるROI(Region of Interest）を設定する.

上記の元画像から，青枠で囲った領域を切り出す



```python
import cv2
img = cv2.imread("pilotlamp.png") # 画像の読み込み

x, y = 162, 21   # ROIの左上座標
w, h = 50, 50    # ROI画像の幅・高さ

roi = img[y:y+h, x:x+w] # 入力画像からROI領域を切り取り
cv2.imwrite("pilotlamp_roi_output.png", roi)   # ROI画像の保存

cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), thickness=2)
cv2.imwrite("pilotlamp_output.png", img)   # 元画像の保存
```



切り出した画像: 

画像の画素（ピクセル）操作を行う

画素（ピクセル）の値を参照あるいは変更する処理について実装する.
OpenCVではピクセルのデータは `(R,G,B)` の順ではなく，その逆順の `(B,G,R)` で格納されてるため注意する.

```
import cv2
import numpy as np

img = cv2.imread('sample.jpg')

pixelValue = img[10, 20] # y=10, x=20の座標のピクセルの値を取得する.
print('pixelValue = ' + str(pixelValue))

img[10, 20] = (255,255,255) # y=10, x=20の座標のピクセルの値に白を設定

cv2.imwrite('sample_dst.jpg')
```

画像にテキストや図形を描画する

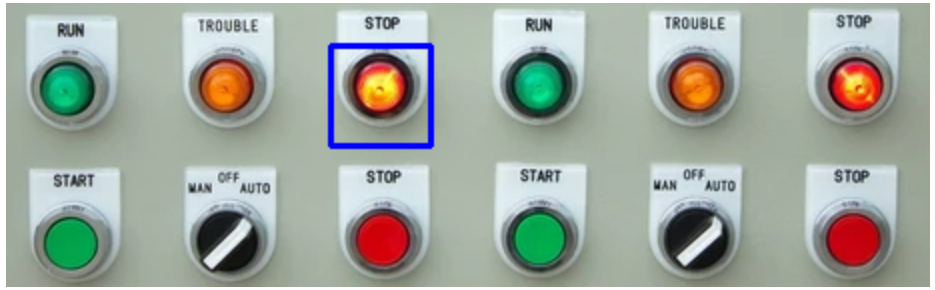
- 線分を描画: `cv2.line()`
- 矢印を描画: `cv2.arrowedLine()`
- 長方形を描画: `cv2.rectangle()`
- 円を描画: `cv2.circle()`
- 楕円を描画: `cv2.ellipse()`
- 円弧を描画: `cv2.ellipse()`
- マーカーを描画: `cv2.drawMarker()`
- 折れ線、多角形を描画: `cv2.polylines()`, `cv2.fillPoly()`, `fillConvexPoly()`
- 文字列を描画: `cv2.putText()`

`cv2.rectangle()` で長方形（矩形）を描画する

```python
import cv2

img = cv2.imread("pilotlamp.png") # 画像の読み込み

x, y = 180, 35   # ROIの左上座標
w, h = 15, 15    # ROI画像の幅・高さ

cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), thickness=2)
cv2.imwrite("pilotlamp_output.png", img)  # ROI画像の保存
```

元画像：

出力画像：



cv2.putText() で画像に文字（時刻）を描画する

```python
import cv2
import datetime

img = cv2.imread("pilotlamp.png") # 画像の読み込み

x, y = 50, 100  # 文字を描画する左上座標

cv2.putText(img,
                text=datetime.datetime.now().strftime('%Y/%m/%d %H:%M:%S'),
                org=(x,y),
                fontFace=cv2.FONT_HERSHEY_SIMPLEX,
                fontScale=1.0,
                color=(0, 0, 0),
                thickness=2
        )
cv2.imwrite("pilotlamp_text_output.png", img)  # ROI画像の保存
```

> cv::putText()の引数について ”

元画像：

出力画像：