# データサイエンス特論
# Data Science

# 画像処理
# Image Processing

1. 画像処理とは
2. 画像データの表現
3. 2値化
4. エッジ検出
5. 周波数フィルタ処理
6. 物体の特徴量抽出
7. オプティカルフロー
8. 顔認識

(C) 創造技術コース　橋本洋志／大久保友幸
{hashimoto, ohkubo-tomoyuki}@aiit.ac.jp
http://hhlab.org/

# 画像データ概要



1. 画像処理の実用例
2. 表色系
3. 標本化と量子化
4. 数値としてのデータ表現

# 画像処理の実用例

## 現代社会で密に実用化されている

- **身の回り**
  - スマホ、ビデオ，カメラ，PC，車載カメラ，インターホン，
- **公共空間**
  - 不審者監視（ATM，マンション，繁華街），身元照合（バイオメトリクス），道路交通観測，Nシステム，河川観測・ダム警報，他
- **科学・工業・医療**
  - 気象予報（(財)日本気象協会），環境観測，製品検査，農作物分別，ロボットビジョン，医療でのレントゲン・CT・MRI検査，他
- **この意義は**
  - 画像から特徴量を抽出し，それを用いて分類，認識を行うことで，価値ある判断が行えるため。
  - 画像データドリブンの考えは，まさしくデータサイエンスの範疇にあるといえる。
- **OpenCV（Open Source Computer Vision Library）**
  - https://opencv.org/
  - フィルタ処理，テンプレートマッチング，物体認識，映像解析などのアルゴリズムが多数提供
  - C/C++をベースにしているが，Python用APIも充実している
  - 世界中の画像処理・認識研究者が良く用いている

# 表色系

## 表色系

- 色を表現するには様々な方法があり，これを表色系と呼ぶ。それぞれの表色系の色要素で構成される空間を色空間と呼ぶ。

| 表色系 | チャネル数 | 色要素 |
|---|---|---|
| RGB | 3 | Red(赤，700nm)，Green(緑，546.1nm)、Blue(青，435.8nm)，この波長はCIE(国際照明委員会)の定めた値。他の分野では色の波長は幅がある。PCのディスプレイで最も採用されている。 |
| HSV | 3 | Hue(色相)，Saturation(彩度)，Value(明度)。CGでよく用いられている。デザイン分野でよく用いられる。 |
| XYZ | 3 | Y(輝度)，Z(青み)，X(それ以外)，光の陰影変化に比較的強く，ロボットビジョンでよく採用されている。 |
| CMYK | 4 | Cyan，Magenta，Yellow，Key Plate，印刷でよく用いられる。 |

## RGB表色系　（以降は，これに限定）

- 光の3原色：　3色交じり合うと白色となる
- 色の3原色：　3色交じり合うと黒色となる

## グレースケール（Gray Scale）

- チャネル数は1，黒-灰色-白と段階的に変わるものをいう。また，8bit階調というのは黒が0，白が255で表される。これをソフト上で規格化して0～1という表現もある。

# RGB表色系

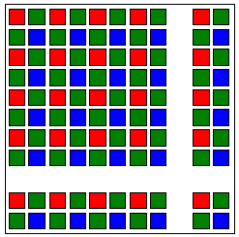
## データ表現

- RGBがそれぞれ1バイト(8ビット, = 0～255)でその強度を表現するとき，全部で256×256×256＝16,777,216≒1670万色の色を表現できる。
- この場合，黒はR=G=B=0，白はR=G=B=255 である。これを255階調ともいう。また，0~255を規格化して0~1で表すものもある。
- しかし，この表現では，全ての色を表現しにくいことから，XYZ表色系など他の表色系が定義されている。

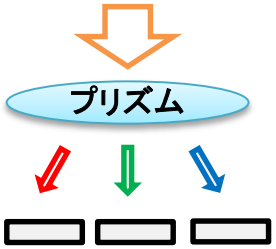
## ピクセル (pixel)

- これは，コンピュータで画像を扱うときの最小単位で、色情報(色調や階調)を持つ画素のことで，pix(pic=写真、画像の意の複数形)+element(要素)の造語。しばしばピクセルと同一の言葉として使われるドット(dot)は単なる点情報である点において区別される。
- 1ピクセルに1ビットの情報しか割り当てなければ、2色しか表現できない。RGBのチャンネルに各8ビット、計24ビットの情報を割り当てれば，約1670万色の色表現が行える。

| イメージセンサ | ディスプレイ |
|---|---|

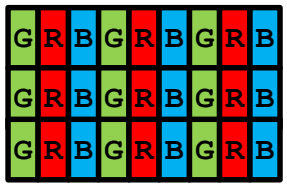


単板式
低コスト，性能(小)




3板式(イメージセンサ3つ)
高コスト，性能(高)


1ピクセル



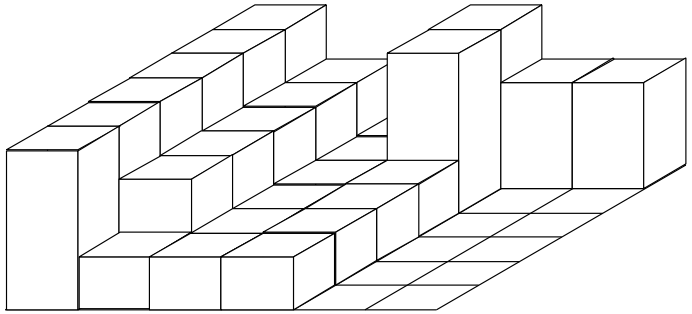

同じ面積ならばグレースケールの方が標本化率は高い(解像度が高い)

# 画像データの表現

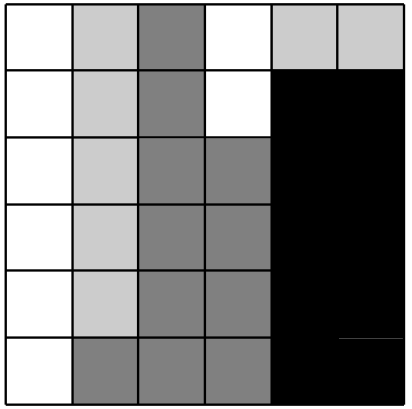
## グレースケール（白–灰色–黒）の場合

| | | | | | |
|---|---|---|---|---|---|
| 3 (11) | 2 (10) | 1 (01) | 3 (11) | 2 (10) | 2 (10) |
| 3 (11) | 2 (10) | 1 (01) | 3 (11) | 0 (00) | 0 (00) |
| 3 (11) | 2 (10) | 1 (01) | 1 (01) | 0 (00) | 0 (00) |
| 3 (11) | 2 (10) | 1 (01) | 1 (01) | 0 (00) | 0 (00) |
| 3 (11) | 2 (10) | 1 (01) | 1 (01) | 0 (00) | 0 (00) |
| 3 (11) | 1 (01) | 1 (01) | 1 (01) | 0 (00) | 0 (00) |

(a) 数値データ



(b) 階調を高さで表現したイメージ



(c) ディスプレイの表示

標本化：縦横6分割
量子化：2ビット（4階調，0～3），黒：0，灰色：1，2，白：3
(a)のカッコ内は2進数

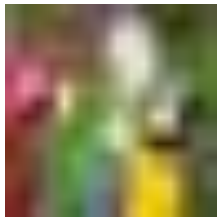
注目点：
（c）で線と見える箇所で，(b)を見るとその境界部分が急峻なものとなだらかなところがあり，この区別が難しい。

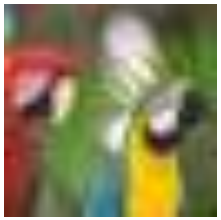
# 標本化と量子化

## 標本化（sampling）

- 画像データを離散的な点(pixel)に分割することを言う。画像に対して縦・横何分割にするか，というイメージ捉えても良い。


8x8 pixel


16x16 pixel


32x32 pixel

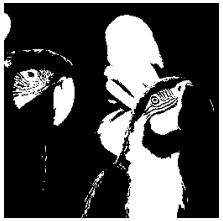

64x64 pixel


128x128 pixel


256x256 pixel

## 量子化（quantization）

- 画像データの各点(pixel)の階調をいう。 1 bitならば2^1=2色，4ビットならば2^4=16色となる。


1bit（2色）


2bit（4色）


3bit（8色）


4bit（16色）


6bit（64色）

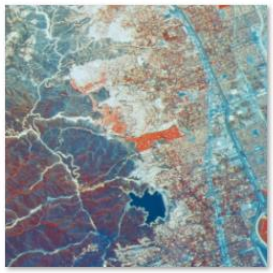

8bit（256色）

注：多くのPCは24bitを採用している。この例では，8bitより多くしても視覚的に変わりなかったので8bitでとどめた。

# 標準画像データ（SIDBA)

標準画像データベースSIDBA(Standard Image Data-BAse)：世界的に標準画像
各人が考案した画像処理・認識プログラムの性能評価に用いられる。入手しやすいサイト

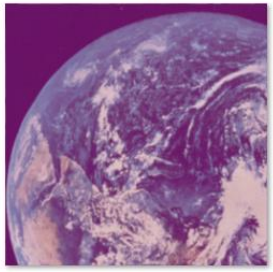
- 南カリフォルニア大学　SIPI　http://sipi.usc.edu/database/
- 検索サイトで”SIDBA Standard Image Data-BAse”を検索


Aerial.bmp


Airplane.bmp


Balloon.bmp


couple.bmp


Earth.bmp


Girl.bmp


lena_std.tif


Mandrill.bmp


milkdrop.bmp


Parrots.bmp


Pepper.bmp


Sailboat.bmp

# OpenCV

## OpenCV

- Document https://opencv.org/ -> “Online documentation”
- OpenCV Wiki https://github.com/opencv/opencv/wiki

## OpenCVをPythonで実行する（一般）

- OpenCVライブラリーがPythonに対応している（C/C++にも対応している）。
- OpenCV-Python Tutorials’s documentation　http://opencv-python-tutroals.readthedocs.io
- OpenCV-Python Tutorials　http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html

## 本授業でOpenCV-Pythonスクリプトの実行

- ファイル形式が”.py”となる（Jupyter Notebook “.ipynb”でないことに注意！）
- 簡単な方法

  コマンドプロンプト　>　python filename.py
- 統合開発環境Sypderを用いる

  参照：　https://sites.google.com/site/datasciencehiro/　→　Python開発環境　→　Pythonスクリプト(.py)の開発方法

**講義では実行することはありませんが、画像データ処理の原理だけを理解してください**

## OpenCV-Python Tutorials

- https://docs.opencv.org/ → Doxygen HTMLで最もバージョンの新しい番号に入り→” OpenCV-Python Tutorials”をクリック, ここに各種画像処理の説明がある

## パッケージのインポート

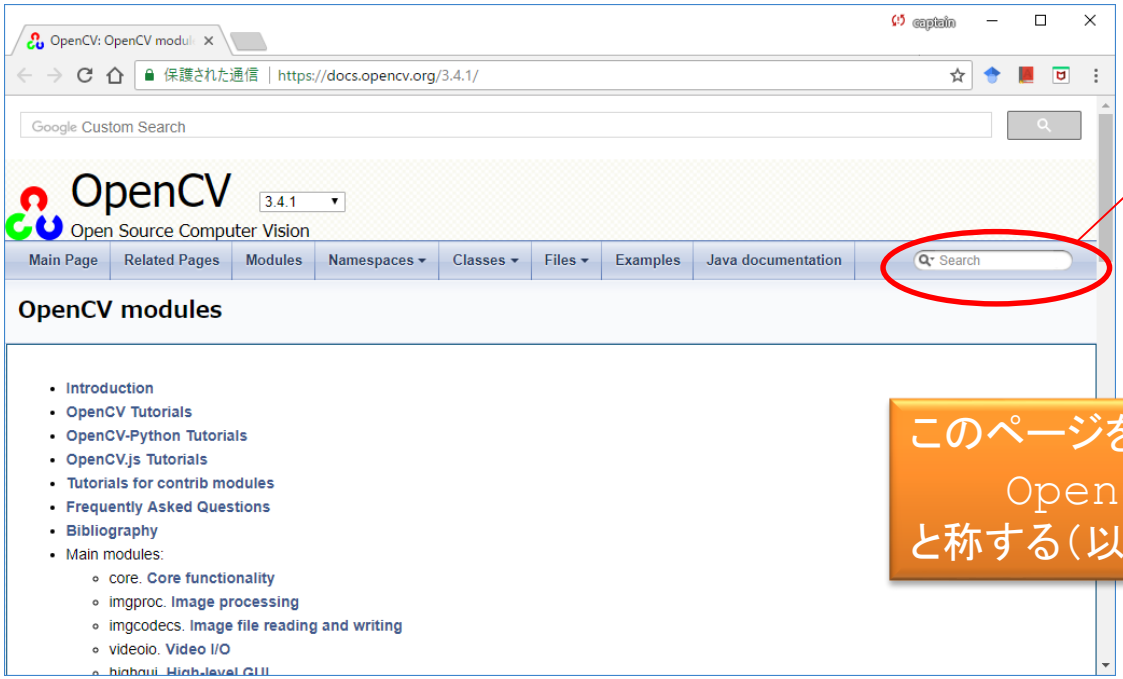
- OpenCV 3であっても次のようにする

  > import cv2

# OpenCV　ドキュメント

## C/C++言語を扱える人

- 公式HP　https://opencv.org/　→　“Online documentation” → ”Doxygen HTML” から最新バージョン番号をクリック




ここに，調べたい関数名を入れて検索する

このページを
OpenCVサイト
と称する（以降，これを用いる）

- 日本語サイト：http://opencv.jp/　　→　“REFERENCE MANUAL” → ”C”または”C++”リファレンス日本語訳を見る，または，このHP→”SAMPLE　CODE”も役に立つ

## Pythonの立場から調べたい人

- OpenCV-Python Tutorials　https://docs.opencv.org/3.4.1/d6/d00/tutorial_py_root.html
- http://opencv-python-tutroals.readthedocs.io/

# スクリプト　“.py”の実行

## 今回は，Jupyter Notebookを用いません。

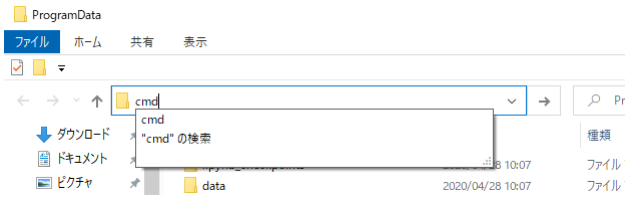
- リアルタイム処理が一部入るためです。

## Pythonスクリプトである”.py”を実行します。

- 実行手順（Windowsの例）

1. フォルダProgramDataを開き
2. コマンドプロンプトを開く（PowerShellの場合もあり）→カレントフォルダ（カレントディレクトリ）
   - 方法1：SHIFTキー＋右クリックで開いたサブウィンドウから選択
   - 方法2：フォルダ内のアドレスバーに直接”cmd”を入力。
3. コマンドプロンプトから次を入力すると実行します。

   ¥> python　filename.py




## スクリプトの編集

- 適当なエディタを用いてください（ワードなどの文章ソフトは用いない）
- 大学PCでは，”.py”をダブルクリックすると，SciTEが立ち上がり，編集できます。
  - SciTEでは，編集の後に，F5を押すと実行できます。

SciTE: Windows上、Python Scriptを開発・実行できる統合エディタ、軽快なためよく用いられる。
https://www.scintilla.org/SciTE.html

# 各処理の説明



1. 2値化
2. エッジ検出
3. 物体形状の特徴量抽出
4. 周波数フィルタ処理
5. オプティカルフロー
6. 顔認識

# 2値化

## RGB－＞グレースケールへの変換

- 変換方法は幾つかある（https://en.wikipedia.org/wiki/Grayscale ）
- OpenCVは次を採用している。（OpenCVサイトで”Color conversions”を検索）

$$Y = 0.299R + 0.587G + 0.114B$$

## 2値化

- グレースケール（例：0～255）で，設定したしきい値より大きければ255（白），小さければ0（黒）とすること。[0, 1]に規格化した考え方ならば次の表現となる。

$$Y \in [0,1] = \begin{cases} 0 & if\ Y < thresh \\ 1 & f\ Y \geq thresh \end{cases}$$

IMG_Gray2Bin.py

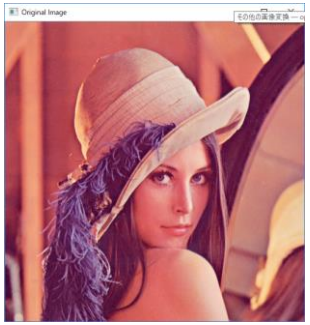
## トラックバー

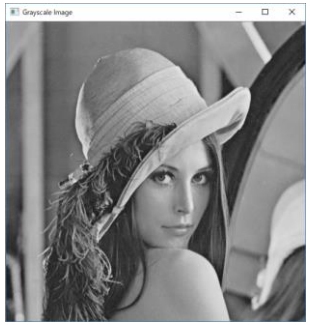
- しきい値をトラックバーで与えて，視覚で2値化の確認をしやすいようにする

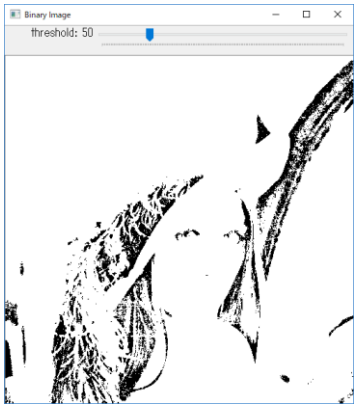
```
#RGB -> GrayScale
img_org = cv2.imread('data/lena_std.tif') # オリジナル画像の入力
img_gry = cv2.cvtColor(img_org, cv2.COLOR_BGR2GRAY) # グレイスケールに変換
#2値化
retVal, img_bin = cv2.threshold(img_gry, threshVal, MAX_VAL, type=cv2.THRESH_BINARY)

#トラックバー
cv2.createTrackbar('threshold', WIN_BIN, DEFAULT_THRESH_VAL, MAX_VAL, update)
```

# 2値化

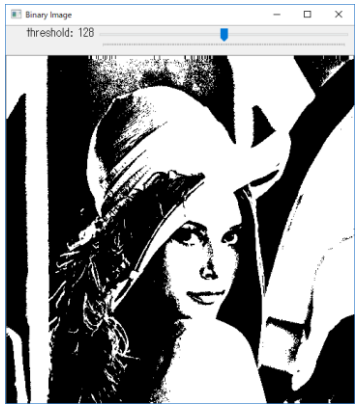


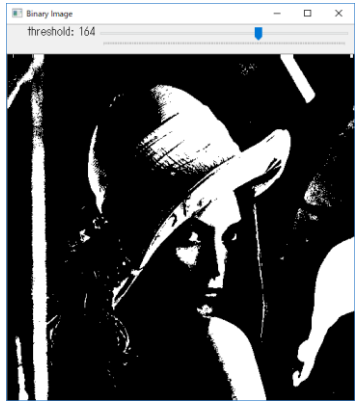




threshold=50


threshold=80




threshold=128




threshold=164

**2値化の用途:**
白地に黒の文字が光線の影響でぼけているものをくっきりと抽出する，などに用いられる。
この例に見るように，視覚の陰影と画像処理の陰影結果が異なる。
また，しきい値が画像全体に対し一定であることも問題と考えられ，背景によりしきい値を適応的に変化させるという考え方もある。

演習:照明条件が悪いところで，白地に黒文字を書いた写真の2値化を行ってみる

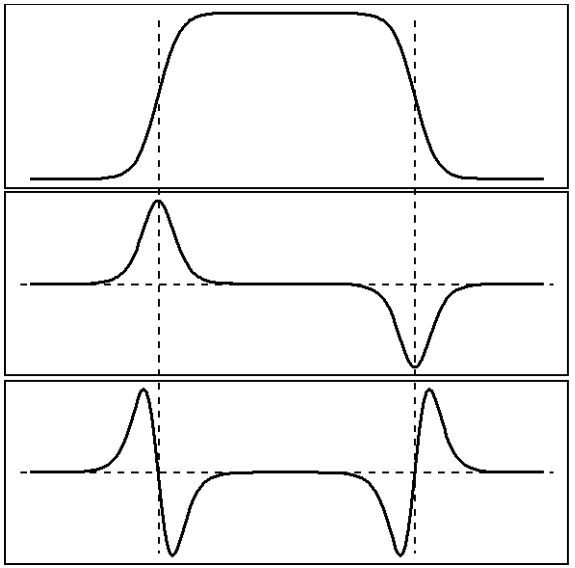
# エッジ検出

## エッジ(edge)とは

- 境界，へり，(二つの線の接する)縁，(刃物の)刃，(刃の)鋭利さ
- 画像では，輝度が大きく変化している箇所(境界)
- 人間は画像データの輝度変化を捉えることができる→人間の視覚細胞で特定のエッジ方向に発火するニューロンが見つかっている。

## 画像データの微分

- 画像データを微分すれば，変化しているところがエッジとして見なせる
- 離散データゆえ，微分の代わりに差分が用いられる。
- 2階微分が振れの間のゼロとなる箇所をエッジとして抽出してもよい




## 問題点（一部）

- 画像データの離散的な段々の変化と差分の性質から，抽出したエッジは，一般に，幅がある。したがって，抽出したエッジをそのまま輪郭線として見なしてよいか否かは，用途に関わる。太さのあるエッジを１ピクセルまでにする細線化処理がある（他書を参照）。
- 微分はノイズに敏感であり，ノイズに反応する。
- 画像データは2次元である。一方，単純差分は一方向（x軸，または，y軸のみ）であり，2次元に対応できるエッジ抽出器を選ぶ。

# エッジ検出

## Sobel法

- cv2.Sobel
- 1次微分ゆえ，エッジ検出に1方向性がある

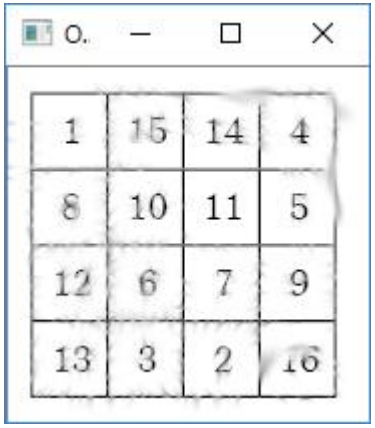
## Laplacian

- cv2.Laplacian
- 2次微分で，4方向（縦横の上下）がある。

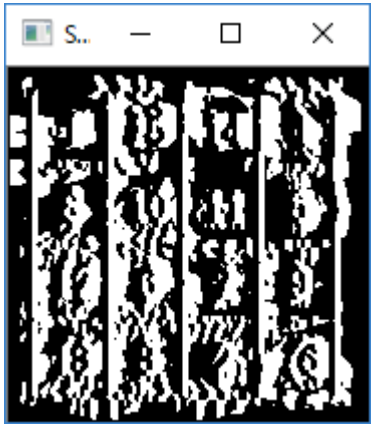
## Canny法

- cv2.Canny()
- 比較的性能が良く，よく用いられる方法。
- 画像中のノイズ影響を低減するため，ガウシアンフィルタを用いた平滑化を行った後に，複数ステップによるエッジ抽出を行う。
- https://en.wikipedia.org/wiki/Canny_edge_detector

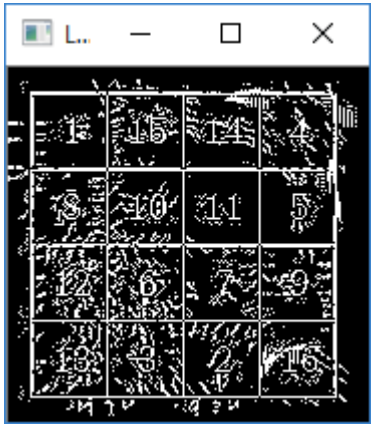
# エッジ検出



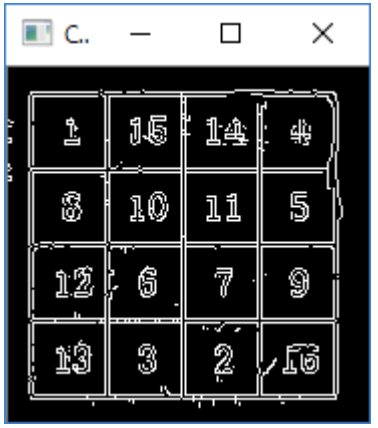

(a) 画像データ




(b) Sobel（x方向）




(c) Laplacian




(d) Canny

## 結果

- 画像データ：若干のノイズが重畳し，かつ，にじみを与えている
- Soble（x方向）：x方向の微分を行っているので縦線の検出はできる。一方，横線の検出はできていない。
- Laplacian：縦横の枠線はよく検出できているが，ノイズに敏感に反応して数字の検出が不十分
- Canny：ノイズに頑強で，かつ，縦横方向の検出もできている。ただし，線分の抽出はできておらず，枠および数字は線でなく2つの境界線として検出されている。

IMG_EdgeDetection.py

SobelとLaplacianは次を参照 http://opencv.jp/opencv-2svn/cpp/imgproc_image_filtering.html
Cannyは次を参照 http://opencv.jp/opencv-2.1/cpp/feature_detection.html

```
edge_sob_x = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
edge_lapl = cv2.Laplacian(img, cv2.CV_64F)
edge_cann = cv2.Canny(img, 80, 120)
```

# Lennaのエッジ検出

- 試してみて，何が言えるか？

# 周波数フィルタリング



時間領域の波形のフーリエ変換は周波数領域に移され，周波数成分で表現された。

画像データは２次元ゆえ空間領域にあると言える。これに２次元フーリエ変換を行い周波数領域に移されたものは空間周波数で表現される。

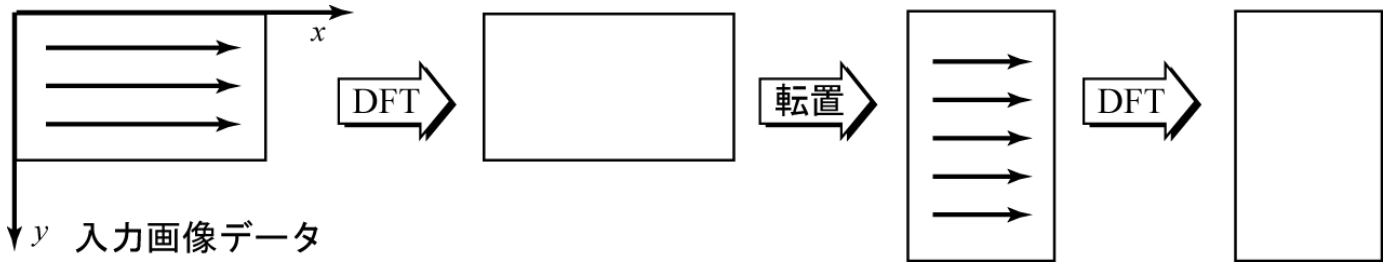
空間周波数で表すことも画像の特徴量となる。

空間周波数に対するローパスフィルタ，ハイパスフィルタを施すことで，画像がどのように変化するかを見る。

なお，FFT（DFT)は，numpyとOpenCVの両方が提供しており，両方の使用を示す。

# 2次元フーリエ変換の概要

## 2D-DFT




2D-DFT

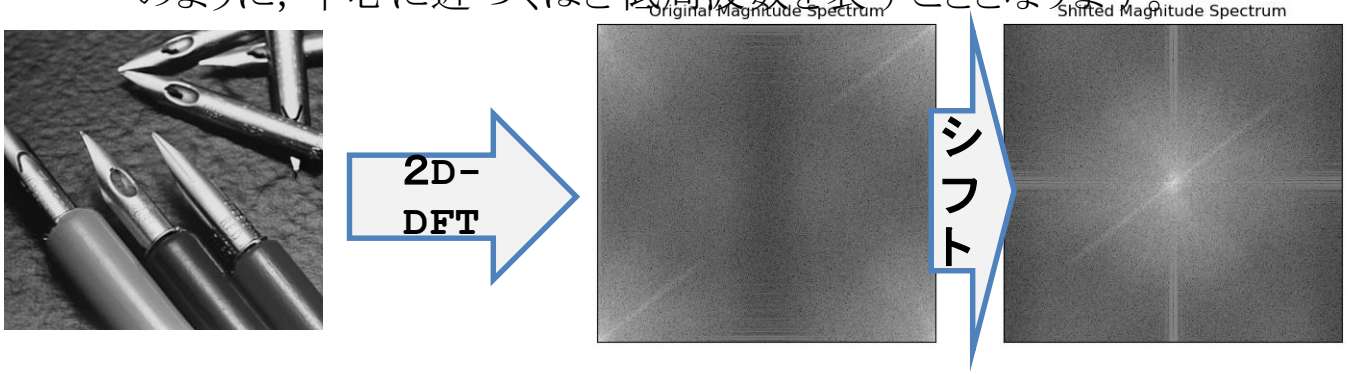
$$F(u,v)=\frac{1}{NM}\sum_{y=0}^{M-1}\sum_{x=0}^{N-1}f(x,y)\exp\{-j2\pi xu/N\}\exp\{-j2\pi yv/M\}$$

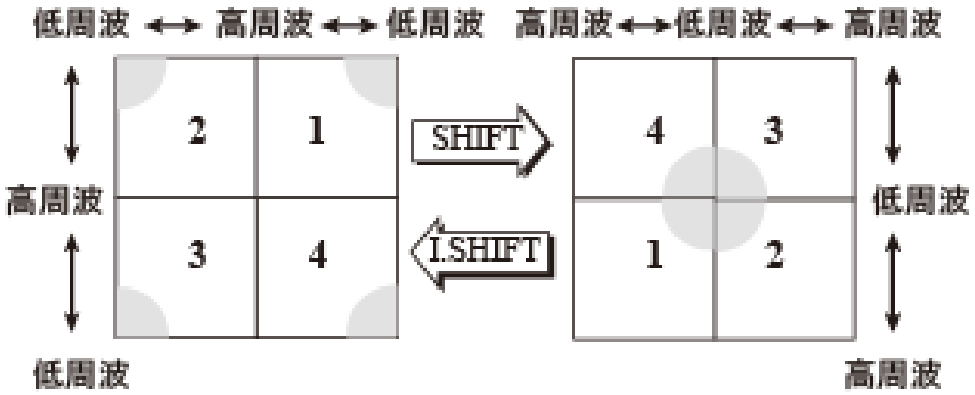
2D-IDFT

$$f(x,y)=\sum_{v=0}^{M-1}\sum_{u=0}^{N-1}F(u,v)\exp\{j2\pi(xu)/N\}\exp\{j2\pi(yv)/M\}$$

# 振幅スペクトルの表現

- 図1の画像の2次元DFTは，図2(a)のようになる。
- この見方は図3(a)のように，中心を原点として，中心から離れるほど低周波数となる。
- これでは扱いにくいので，N点DFTのとき，N/2だけをシフトします。このシフトの意味が図3(b)のように，中心に近づくほど低周波数を表すこととなります。

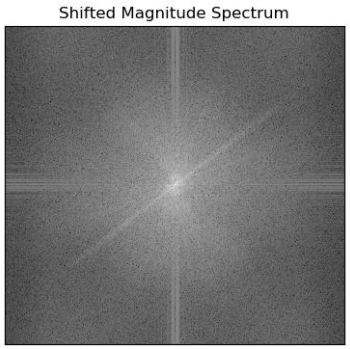


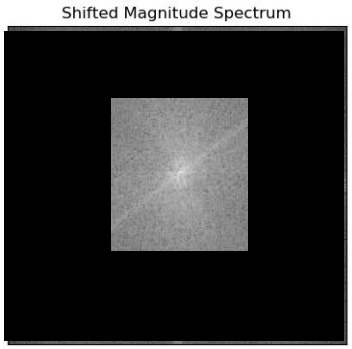



# フィルタ

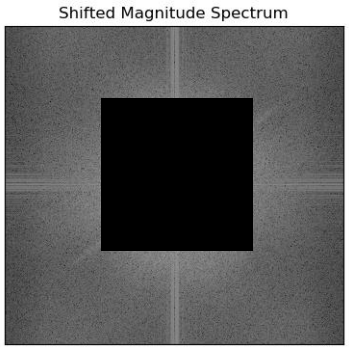
## フィルタの表現

- サンプリング間隔(標本化)，量子化の問題はここでは考えないものとして
- フィルタを次のように表現する。




ローパスフィルタ




ハイパスフィルタ

・フィルタをmaskとして表現する。maskは0，1で表現できるため。
・maskを矩形窓にすると，もともとは無い波がのような画像となるリンギング現象が生じる。
・maskの形状には，他に円やGaussian窓が用いられる。

- 逆シフト操作
- 離散フーリエ逆変換

- フィルタリングされた画像

参照

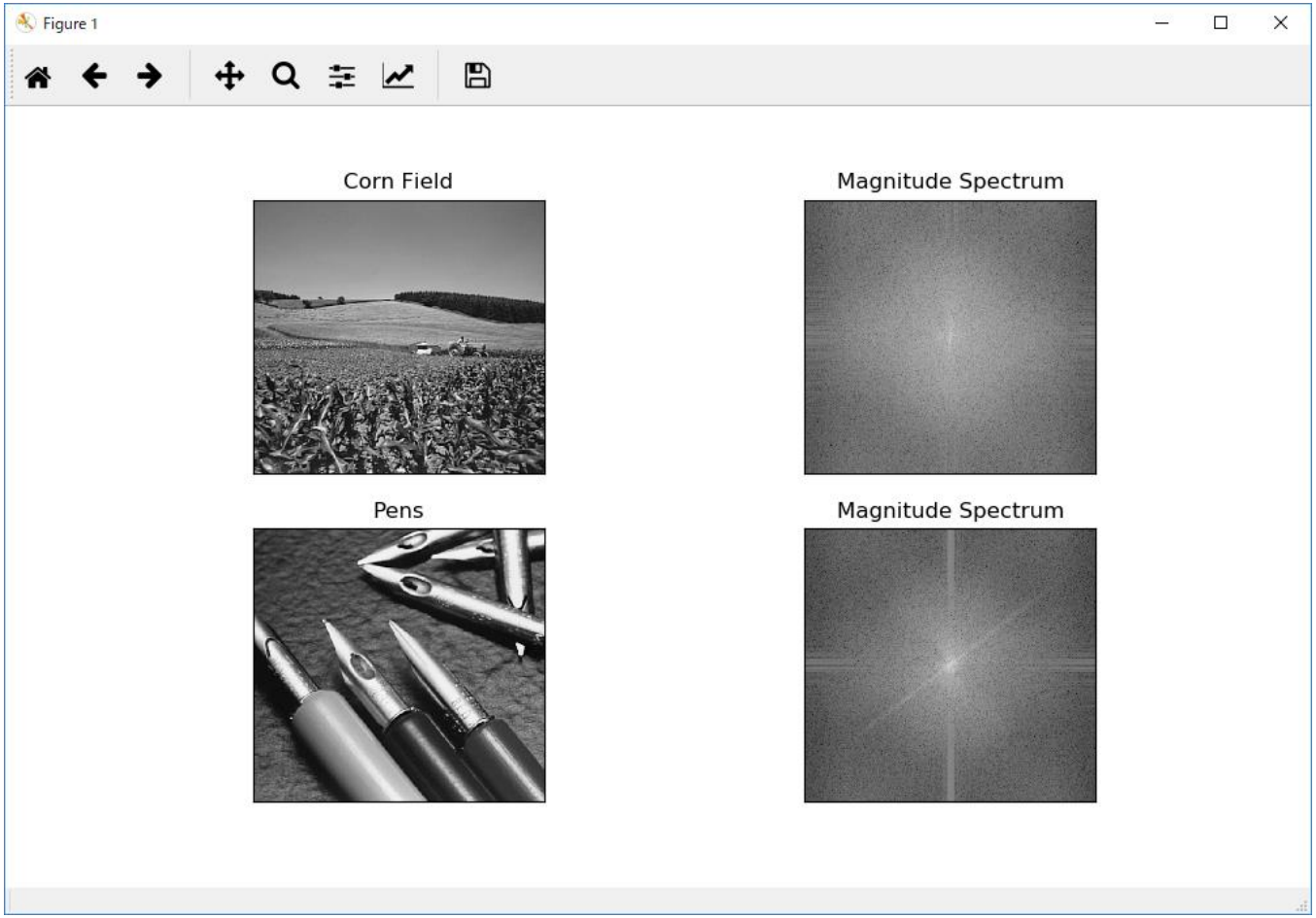
https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_transforms/py_fourier_transform/py_fourier_transform.html

http://lang.sist.chukyo-u.ac.jp/classes/OpenCV/py_tutorials/py_imgproc/py_transforms/py_fourier_transform/py_fourier_transform.html

# 画像の振幅スペクトル

## numpy のFFTを用いる

IMG_FFT.py




見方：振幅スペクトル，明るい方が強度が強い。（白の方が黒よりも値が大きい）

Corn Fieldは，低周波成分から高周波成分までを一様に有している。これは，膨大な数の葉の影響と考えられる。

Pensは，低周波成分を多くふくんでいる。刃先の数はCornの数より非常に少なく，また，ペンはおよそ直線的であるためと考えられる。

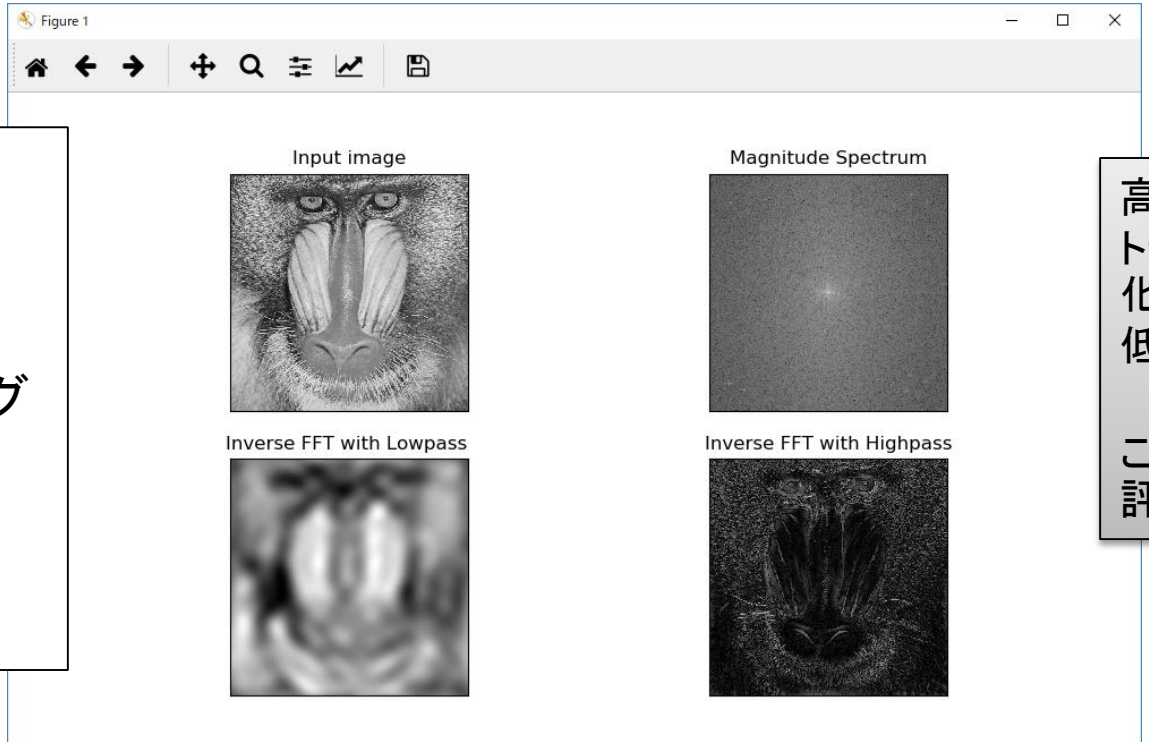
# ローパスフィルタ，ハイパスフィルタ

## 対象と設定

IMG_DFT_Filter.py

- 512 x 512 ピクセルの画像データ
- OpenCV提供のDFTで周波数領域に変換
- この領域で，原点中心で1辺20ピクセル（=isize×2）の正方形窓をマスクとして設ける
- **ローパスフィルタ**:この正方領域内のデータだけをフィルタリング（通過させる）
- **ハイパスフィルタ**:この正方領域外のデータだけをフィルタリング（通過させる）
- 両者の結果を見比べてみよう。何が言えるか？
- 見方を変えて:isizeを適当に大きく取ると入力イメージと見た目で変わらなくなる。このisizeは512よりも十分に小さい。**画像データ量の低減化**を図ることが可能となる。

- 2D-DFT
  ↓
- シフト
  ↓
- フィルタリング
  ↓
- 逆シフト
  ↓
- 2D-inv DFT




高周波数成分:コントラストが細かく変化している箇所
低周波成分:この逆

これを基に，結果の評価を試みる

# 問題

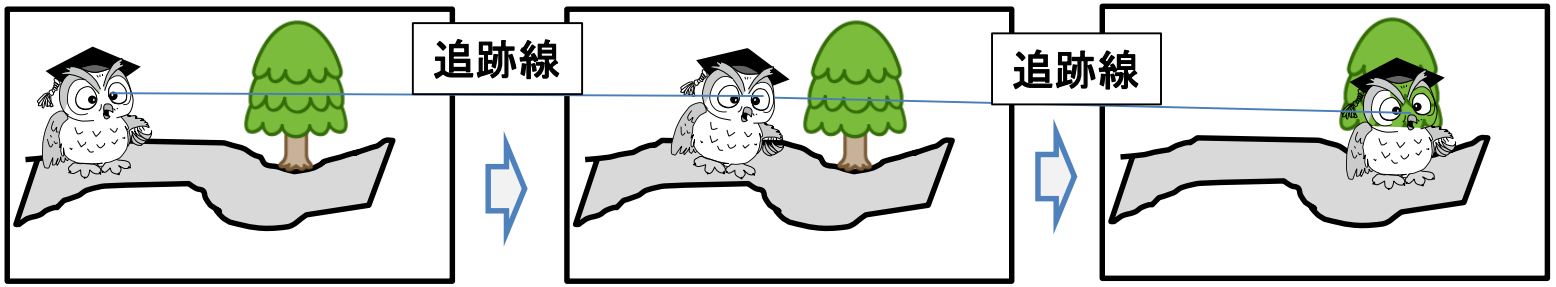
- 肌表面画像は，低周波数成分と高周波数成分どちらを多くふくむであろうか？それに対するフィルタリングの結果はどうなる？
- 他に，このような例はあるのか？

# 特徴点抽出



# 特徴点抽出

## 物体の認識や追跡に用いる




木のイラスト：http://flode-design.com/?page_id=1452

- OpenCV 3.0.0-dev documentation → OpenCV-Python Tutorials → Feature Detection and Description → Understanding Features
- 動画で紹介（OpenCV GSoC 2015）：https://youtu.be/OUbUFn71S4s

見てみましょう

Ref.: GSoC: Google Summer Of Code, https://summerofcode.withgoogle.com/

## 特徴点（feature point）とは

- 画像の特徴；画像の色、輝度、輪郭、コーナー，固有値、固有ベクトルなど
- これら複数（特徴ベクトル）を有する点

# 特徴点抽出法

## OpenCVが提供する特徴点抽出法

- FAST, ORB, BRISK, AKAZE, SHIFTなど
- https://docs.opencv.org/3.0-beta/modules/features2d/doc/feature_detection_and_description.html
- https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_table_of_contents_feature2d/py_table_of_contents_feature2d.html

## AKAZEを採用

- 有力な方法のうち，AKAZE(Accelerated Kaze)を採用する。KAZEはSIFTやSURFの欠点を解決したアルゴリズムである。この更なるロバスト性の向上と高速化を図ったのがAKAZEである。
  - Pablo F. Alcantarilla, Jesús Nuevo and Adrien Bartoli:Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces, In British Machine Vision Conference (BMVC). Bristol, UK. September, 2013,
  - 著者のHP: http://www.robesafe.com/personal/pablo.alcantarilla/kaze.html
  - OpenCV 3.0.0-dev documentation は次 https://docs.opencv.org/3.0-beta/index.html
  - OpenCV 3.0.0-dev documentation » OpenCV Tutorials » feature2d module. 2D Features framework » AKAZE local features matching
  - OpenCV 3.0.0-dev documentation » OpenCV Tutorials » feature2d module. 2D Features framework » AKAZE and ORB planar tracking

# 特徴点抽出法

## OpenCVが提供する特徴点抽出法

- FAST, ORB, BRISK, AKAZE, SHIFTなど
- https://docs.opencv.org/3.0-beta/modules/features2d/doc/feature_detection_and_description.html
- https://qiita.com/hitomatagi/items/62989573a30ec1d8180b
- http://lang.sist.chukyo-u.ac.jp/classes/OpenCV/py_tutorials/py_feature2d/py_table_of_contents_feature2d/py_table_of_contents_feature2d.html （この英語版は？）
- どれを使う？
  - https://qiita.com/shu223/items/fa3cf693296e5641f771
  - これによると，AKAZEがいいみたい。一手法だけでよい。
  - http://lang.sist.chukyo-u.ac.jp/classes/OpenCV/py_tutorials/py_feature2d/py_table_of_contents_feature2d/py_table_of_contents_feature2d.html

## 特徴点抽出と対応付けの例

- https://docs.opencv.org/3.0-beta/doc/tutorials/features2d/table_of_content_features2d/table_of_content_features2d.html

## 発展例

- 動画を入力として，移動する物体を抽出することが行われている。この事例を次に紹介する。
- https://www.youtube.com/watch?v=OUbUFn71S4s

- https://docs.opencv.org/master/da/df5/tutorial_py_sift_intro.html

# 特徴点のマッチング

## マッチング

- 特徴点から複数の特徴量が得られる → パターン認識でいう特徴ベクトル
- 二つの画像から類似の特徴ベクトルを見出し，これを対応付ける（マッチング）すること

## 考え方

- 総当たり探索（Brute-Force）
  - 全探索空間を総当たり探索のため時間がかかるが，確実に最近傍の特徴量を検索することができる。
  - https://en.wikipedia.org/wiki/Brute-force_search
- 高速近似最近傍探索（FLANN: Fast Library for Approximate Nearest Neighbors）
  - 探索対象の特徴点と近い空間のみを探索する。探索空間が減るので高速に検索することができる。ただし，探索のパラメータの指定が適切でないと，探索空間の選択を誤ることとなり，最近傍の特徴点が含まれていない探索空間だけを探索することとなる。
  - https://www.cs.ubc.ca/research/flann/

## Brute-ForceのkNNマッチングを採用

- kNN（k-Nearest Neighbor algorithm）は，探索空間から最近傍のラベルをk個選択し，多数決でクラスラベルを割り当てるアルゴリズム。k=2を指定すれば，画像間それぞれ一つずつを選択することとなる。

## 参考

- https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html
- http://lang.sist.chukyo-u.ac.jp/classes/OpenCV/py_tutorials/py_feature2d/py_matcher/py_matcher.html を

# AKAZE+BF（kNN）の例

## スクリプト

IMG_FeatureDetectionAKAZE.py

- 特徴点は幾つ生成されるかわからないので，ratioで対応線の数を調整する。

```
# image 1
img1 = cv2.imread("data/book.jpg")
#cv2.imshow('img1', img1)
# image2
img2 = cv2.imread("data/book_in_scene.jpg")

# A-KAZE検出器の生成
# https://docs.opencv.org/3.4.1/d8/d30/classcv_1_1AKAZE.html
akaze = cv2.AKAZE_create()

# 各画像から特徴量の検出と特徴量ベクトルの計算
kp1, des1 = akaze.detectAndCompute(img1, None)
kp2, des2 = akaze.detectAndCompute(img2, None)

# Brute-Force Matcher生成
bf = cv2.BFMatcher()

# 特徴量ベクトル同士をBrute-Force+kNNでマッチング
matches = bf.knnMatch(des1, des2, k=2)
```

# AKAZE+BF（kNN）の例

## スクリプト

IMG_FeatureDetectionAKAZE.py

- 特徴点は幾つ生成されるかわからないので，ratioで対応線の数を調整する。

```
# データを間引きする， ratioと，特徴点同士を結ぶ対応線の数は比例する
ratio = 0.6
good = []
for m, n in matches:
    if m.distance < ratio * n.distance:
        good.append([m])

# 対応する特徴点を直線で引く(対応線)
img3 = cv2.drawMatchesKnn(img1, kp1, img2, kp2, good, None, flags=2)
```

# AKAZE+BF（kNN）の例

## 結果の説明

- 左図：img1，右図：img2
- 右図の複数の本から，自身を良く選んでいることが認められる。
- 人物や物体の移動追跡に用いることができることがわかるであろう。
- 対応線の数はratioで調整できる。

# 他



1. Optical Flow
2. 顔認識

# Optical Flow

- 画像の特徴点の動く速度ベクトル（大きさと方向）に合わせたベクトル表示（これがOptical Flow）することで，動きの流れ（flow）を可視化したもの
- Lucas-Kanade法の例
  - カメラ接続が条件

IMG_OpticalFlow.py

参照：

OpenCV https://docs.opencv.org/3.3.1/d7/d8b/tutorial_py_lucas_kanade.html

この日本語：http://labs.eecs.tottori-u.ac.jp/sd/Member/oyamada/OpenCV/html/py_tutorials/py_video/py_lucas_kanade/py_lucas_kanade.html

OpenCV GSOC 2015 https://youtu.be/OUbUFn71S4s

Wiki https://en.wikipedia.org/wiki/Lucas%E2%80%93Kanade_method

# 顔認識

## ここでの顔認識の意味

- 人の顔が画像上どこにあるかを認識する。
- 人の同定を行うわけではない。

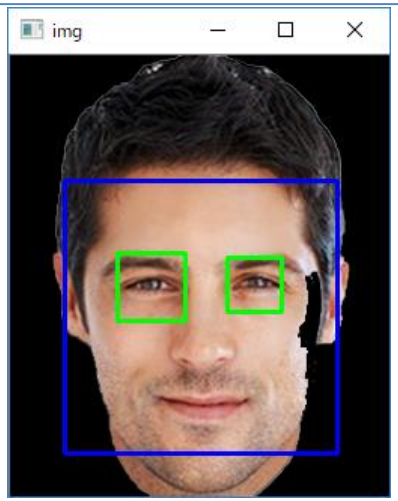
## 手法

- Haar-like特徴量を用いて，眼，鼻，顔(正面)など個別の特徴分類器を用意し，それらをカスケード型に組合わせて認識を行う。

## スクリプトの実行

IMG_FaceDetection.py

認識した顔と眼を矩形で囲む




IMG_FaceDetectionFromCamera.py

リアルタイムで，カメラで撮影した顔の認識を行う

参考

Wiki https://en.wikipedia.org/wiki/Haar-like_feature

OpenCV-Python https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html

1. 金谷 健一，空間データの数理―3次元コンピューティングに向けて，朝倉書店，1995
2. G. Bradski，A. Kaehler（著），松田 晃一，他（翻訳）：詳解 OpenCV 3 ―コンピュータビジョンライブラリを使った画像処理・認識，オライリージャパン,2018
3. 原田達也：画像認識（機械学習プロフェッショナルシリーズ），講談社，2017
4. 堀越力，森本正志，三浦康之，澤野弘明:IT Text画像工学，オーム社，2016
5. Pablo F. Alcantarilla, Jesús Nuevo and Adrien Bartoli:Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces, In British Machine Vision Conference (BMVC). Bristol, UK. September, 2013, http://www.bmva.org/bmvc/2013/Papers/paper0013/paper0013.pdf

# END