スーパーサイエンスハイスクール事業

滋賀県立膳所高等学校理数科2年AI基礎講座

金沢大学
KANAZAWA UNIVERSITY

本邦初？

実習型AIドローン講義

ドローンを題材とする人工知能体験

金沢大学　軸屋一郎

2022年8月31日 (金)3・4限(10時30分~12時30分)

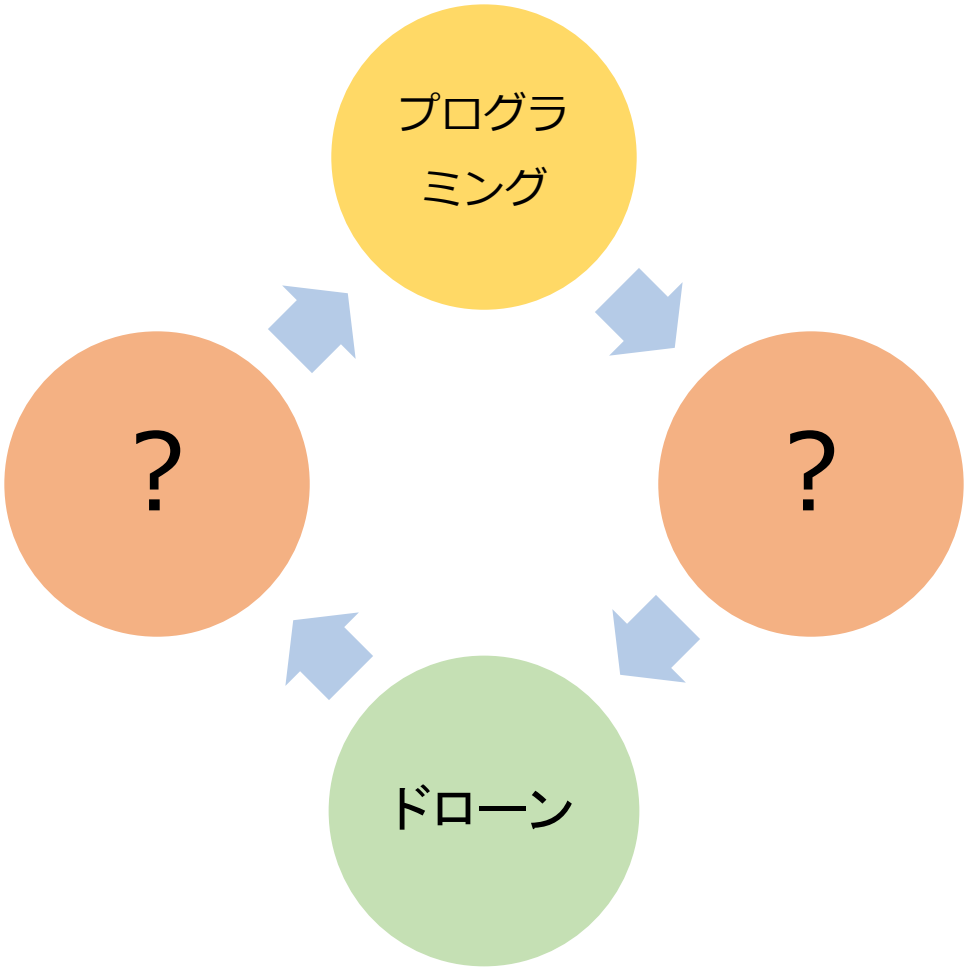
滋賀県立膳所高等学校視聴覚室

TA: 今井祐希，東宗太朗

教材評価: 一方井祐子

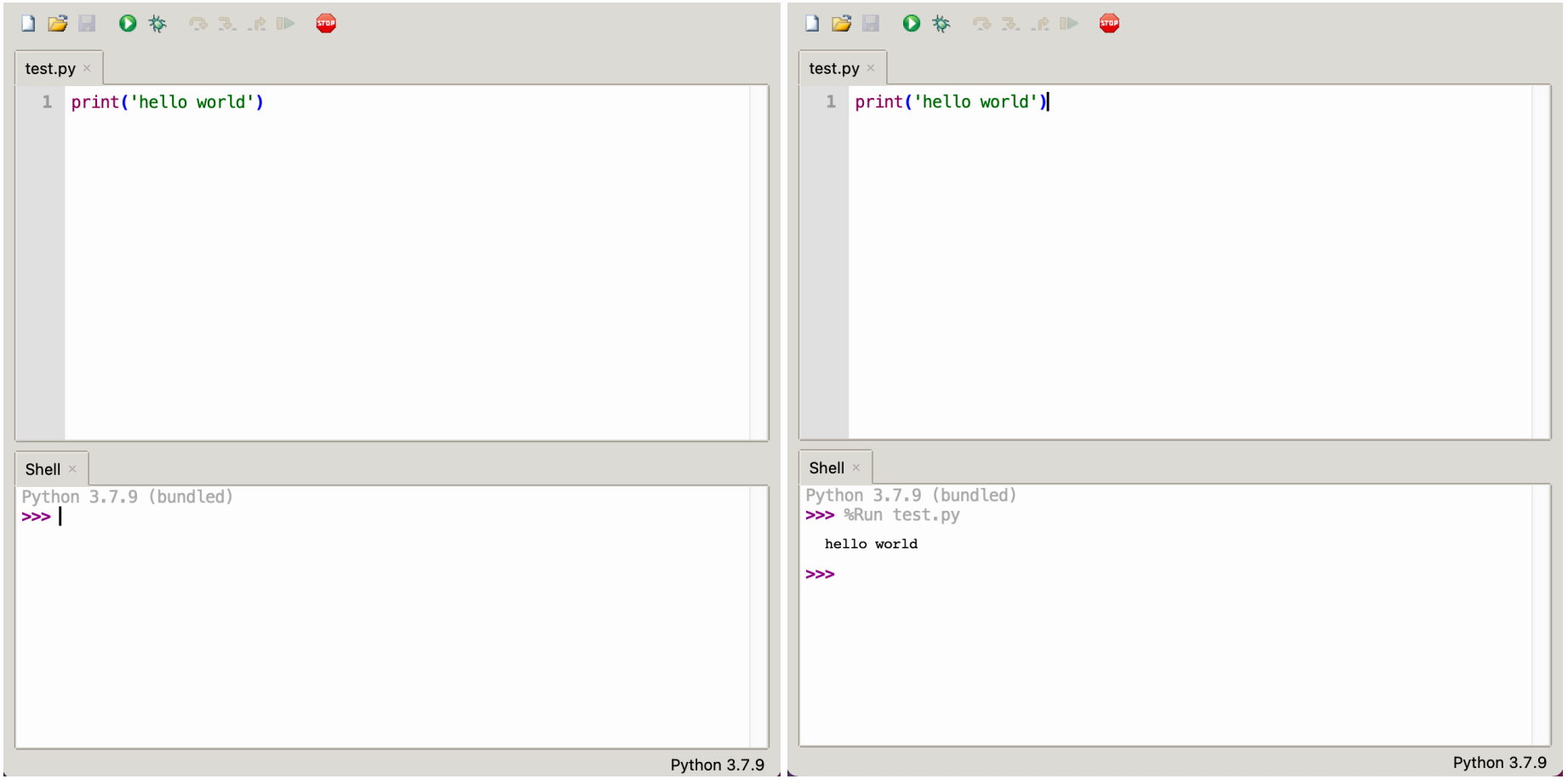
講義の目的

ドローンを使って学びながら遊んでみよう♪



Thonny

- 統合開発環境：



プログラム編集・保存

緑色実行ボタンを押す

実行結果出力

講義概要

3限

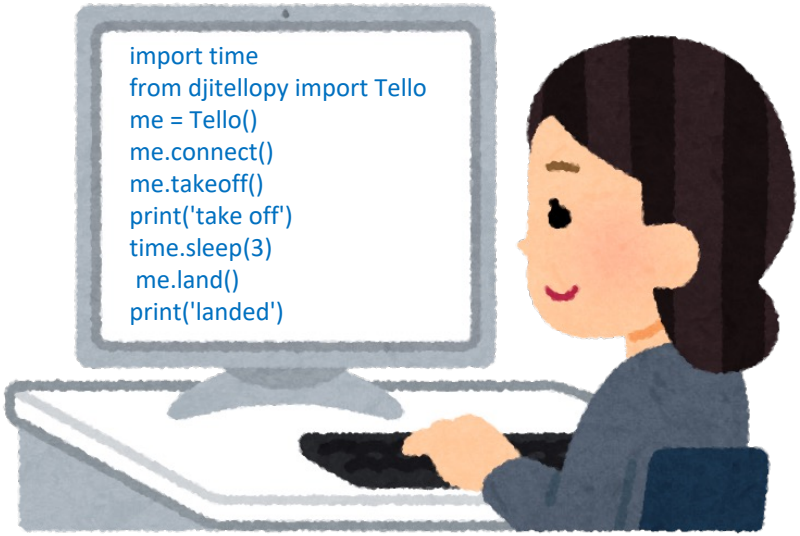
- **ドローン**
- 実習：飛行制御

4限

- デジタル画像
- 実習：画像処理
- 人工知能
- 実習：顔認識
- まとめ

Tello

- 重量87g
  - 国土交通省が定める無人飛行機の飛行ルールの適用外
- 飛行方法
  - スマートフォンアプリ
  - プログラミング：　Scratch、Python



使用上の注意点

- **安全眼鏡 or 眼鏡着用**
  - 視野外からの衝突に注意
- エアコンの風などにより流される危険性
  - 広い場所 & 下が平坦 が基本
  - 変な飛び方をしたら
    本などで優しく落としても構いません
  - 着陸時に机の端にかかるなら
    手乗り着陸も可能です
  - 万一、ドローンが破損しても構いません
    安全を最優先して下さい

先生方：ドローン飛行時・カメラ使用時は**電灯点灯**でお願いします

講義概要

3限

- ドローン
- **実習：飛行制御**

4限

- デジタル画像
- 実習：画像処理
- 人工知能
- 実習：顔認識
- まとめ

バッテリ残量確認

学習3-1: tello_bat.py

```
from djitellopy import Tello

me = Tello()

me.connect()

print('connected')

print(me.get_battery())

me.end()

print('terminated')
```

- 実行してみましょう！
  - 「学習用：飛行制御」からサンプルプログラム tello_bat.py を開く
  - Thonny編集画面にプログラムが表示されていることを確認
- 作業手順の詳細は次ページに記載
  - Shellに充電率が表示されたら動作検証完了です

作業手順

1. Thonnyにおいてプログラムを表示・作成済みとする
2. Telloの電源オン
   - 本体横の電源スイッチを押す
   - 注意：5秒程度長押しするとパスワードリセット

3. PCのwifi接続先から**配布機体のSSID**を選択
   - SSIDはバッテリ下に記載（Tello-******；機体表面に転記済）
   - **パスワード入力**（ID******：ID+6桁英数字）
   - telloを親機，PCを子機として通信経路を確立
4. Thonnyからプログラムを実行
   - 初回実行時にはウィンドウズDefenderにブロックされるかもしれません
   - Pythonに「アクセスを許可する」選択をして再度実行して下さい
   - tello_bat.py ではShellに充電率が表示されたら動作検証完了

5. プログラム実行中のLED点灯パターン

- 充電時は青色点灯
- Connect前は黄色点灯
- Connect後は紫色点灯
- 異常時に赤色点灯
- 電源停止時は消灯

6. Telloの電源オフ

- 飛行可能時間：約10分（適宜バッテリ交換）
- 本体が熱くなったら電源オフ
- 長時間使用しない場合も電源オフ
- 再使用時には電源オン・wifi再接続（他機体への誤接続に注意）

バッテリ残量確認（解説１）

学習3-1: tello_bat.py

```
from djitellopy import Tello

me = Tello()

me.connect()

print('connected')

print(me.get_battery())

me.end()

print('terminated')
```

- モジュールdjitellopyからクラスTelloを読み込みます

バッテリ残量確認（解説２）

学習3-1: tello_bat.py

```
from djitellopy import Tello

me = Tello()

me.connect()

print('connected')

print(me.get_battery())

me.end()

print('terminated')
```

- クラスTelloからインスタンスmeを生成します
  - **クラス**とはメンバ変数とメンバ関数を内包したオブジェクトの**雛形**です
  - **インスタンス**とはクラスの**実体化**です
    - 実体化することにより、クラスで設計されたメンバ変数とメンバ関数を利用できるようなります

クラスはオブジェクト指向プログラミングの中核をなす概念です

クラスを理解したら自分でPythonプログラムを書けるようになります

バッテリ残量確認（解説３）

学習3-1: tello_bat.py

```
from djitellopy import Tello

me = Tello()

me.connect()

print('connected')

print(me.get_battery())

me.end()

print('terminated')
```

- メンバ関数connect()を呼び出します
  - PCからtelloにwifi接続した時点で、telloが親機、PCが子機として、通信経路が確立しています（黄色点灯）
  - メンバ変数connect()を実行することにより、様々な**コマンドをtelloに送信可能**となります（紫色点灯）
  - **Print文**により通信確立したことをShellに出力します．動作確認のために記載していますが**省略可能**です．

ドット演算子「.」の解読がPythonプログラム理解の鍵

モジュールからクラスを読み出す，クラスからメンバ変数を呼び出す，クラスからメンバ関数を呼び出す，など

バッテリ残量確認（解説４）

学習3-1: tello_bat.py

```
from djitellopy import Tello

me = Tello()

me.connect()

print('connected')

print(me.get_battery())

me.end()

print('terminated')
```

- メンバ関数get_battery()を呼び出します
  - Print文と組み合わせることにより、充電率をShellに出力します

他のサンプルプログラムにも print(me.get_battery()) を挿入したらバッテリ残量を確認できます

バッテリ残量確認（解説5）

学習3-1: tello_bat.py

```
from djitellopy import Tello

me = Tello()

me.connect()

print('connected')

print(me.get_battery())

me.end()

print('terminated')
```

- インスタンスmeの終了処理
  - 何故か処理時間を要します
  - 省略可能です

離陸・着陸

学習3-2: tello_fly.py

```
import time

from djitellopy import Tello

me = Tello()

me.connect()

me.takeoff()

print('take off')

time.sleep(3)

me.land()

print('landed')

me.end()
```

- 実行してみましょう！
  - 広い場所 & 下が平坦 が基本
  - 着陸場所が微妙なら手乗り着陸も可

プログラムを解読してしてみましょう

DJITellopyというラッパーライブラリを適用することにより可読性が高くなっています

GitHub上のソースコードを読むと具体的な処理内容を理解できます

Tello公式サイトの SDK 2.0 User Guide を読むとより原理的な枠組みを理解できます

離陸・着陸（解説）

学習3-2: tello_fly.py

```
import time

from djitellopy import Tello

me = Tello()

me.connect()

me.takeoff()

print('take off')

time.sleep(3)

me.land()

print('landed')

me.end()
```

- メンバ関数takeff()を呼び出します
- モジュールtimeのメンバ関数sleep()を読み込みます
  - 引数は待機時間[秒]を表します
- メンバ関数land()を呼び出します

takeoff()とland()の間に
関数を挿入すると飛行中に様々な機能を実現できます

飛行中に15秒間コマンド受信が無ければ自動着陸します

プログラムは順次実行 → 適切な場所に関数を挿入すると**機能拡張**できます

離陸・前進・後退・着陸

学習3-3: tello_move.py

```
import time

from djitellopy import Tello

me = Tello()

me.connect()

me.takeoff()

me.move_forward(50)

time.sleep(3)

me.move_back(50)

me.land()

me.end()
```

- 実行してみましょう！

離陸・着陸との差分を確認してみましょう

離陸・前進・後退・着陸（解説）学習3-3: tello_move.py

```
import time

from djitellopy import Tello

me = Tello()

me.connect()

me.takeoff()

me.move_forward(50)

time.sleep(3)

me.move_back(50)

me.land()

me.end()
```

- メンバ関数move_forward()を呼び出します
  - 引数は20から500の間の整数
  - 移動距離[cm]を表します
- メンバ関数move_back()を呼び出します
  - 引数は20から500の間の整数
  - 移動距離[cm]を表します

離陸・回転・着陸

学習3-4: tello_rotatate.py

```
import time

from djitellopy import Tello

me = Tello()

me.connect()

me.takeoff()

me.rotate_clockwise(90)

time.sleep(3)

me.rotate_counter_clockwise(90)

me.land()

me.end()
```

- 実行してみましょう！

離陸・前進・交代・着陸との差分を確認してみましょう

離陸・回転・着陸（解説）

学習3-4: tello_rotate.py

```
import time

from djitellopy import Tello

me = Tello()

me.connect()

me.takeoff()

me.rotate_clockwise(90)

time.sleep(3)

me.rotate_counter_clockwise(90)

me.land()

me.end()
```

- メンバ関数rotate_clockwise()を呼び出します
  - 引数は1から360の間の整数
  - 回転角度[deg]を表します
- メンバ関数rotate_counter_clockwise()を呼び出します
  - 引数1から360の間の整数
  - は回転角度[deg]を表します

構造化定理

- **順次**
  - プログラムは上から下に順番に実行される
- **選択**
  - If文により処理を分岐できる

```
x=3
if x > 0:
    print('xは正')
elif x == 0:
    print('xはゼロ')
else:
    print('xは負')
```

```
xは正
```

- **繰返し**
  - For文により処理を繰り返せる

```
for i in range(3):
    print(i)
```

```
0
1
2
```

選択・繰り返しに馴染みのない人は

順次の考え方に基づき課題に取り組んで下さい

課題

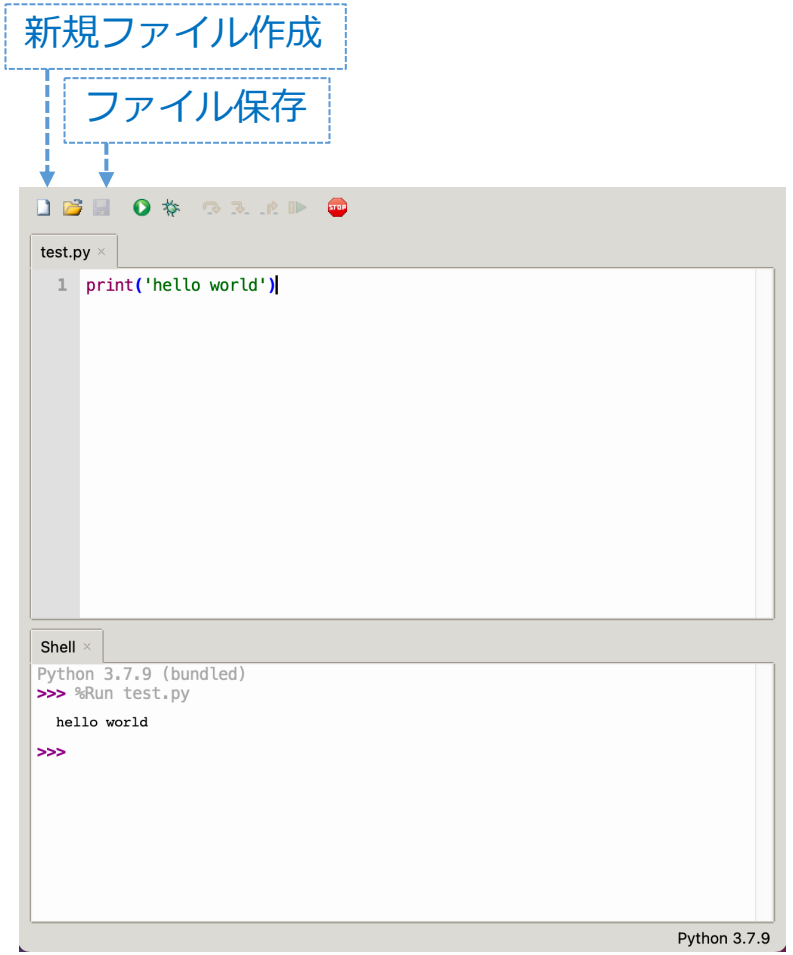
配布資料を参考に課題に取り組んで下さい

- **課題3-5：　　往復運動**
- **課題3-6：　　反復横跳び**

  右移動・左移動を行う関数？

  DJITellopyの関数リストを参照
- **課題3-7：　　正三角形**
- 課題3-8：　　自由課題

  DJITellopyの関数リストを参照



プログラムが正しく記載されていても通信不良のために記載通りに実行しないことがあります

→ Shellに赤字エラーが出力されなければStopボタンで停止してから**再度実行**して下さい

Table of contents

講義概要

3限

- ドローン
- 実習：飛行制御

4限

- **デジタル画像**
- **実習：画像処理**
- 人工知能
- 実習：顔認識
- まとめ

カメラ画像撮影

学習4-1: camera_photo.py

```
import cv2

from djitellopy import Tello

me = Tello()

me.connect()

me.streamon()

print('camera on')

frame_read = me.get_frame_read()

cv2.imwrite('camera_photo.png',frame_read.frame)

print('photo recorded')

me.streamoff()

print('camera off')

me.end()
```

- 実行してみましょう！

自分達の顔の写真を保存してみよう！

撮影係，モデル，PC操作担当を分担しましょう．複数人モデルも可

なるだけ顔の全体が明るく映るように光の加減に工夫して撮影しましょう

画像データをバックアップする場合はファイル名を書き換えて下さい

カメラ画像撮影（解説）

学習4-1: camera_photo.py

```
import cv2

from djitellopy import Tello

me = Tello()

me.connect()

me.streamon()

print('camera on')

frame_read = me.get_frame_read()

cv2.imwrite('camera_photo.png',frame_read.frame)

print('photo recorded')

me.streamoff()

print('camera off')

me.end()
```

streamon()とstreamof()の間に関数を挿入すると
撮影中に様々な機能を実現できます

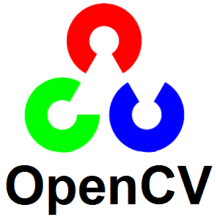
プログラムは**順次**実行 → 適切な場所に関数を挿入すると**機能拡張**できます

カメラ画像撮影（解説）

学習4-1: camera_photo.py

```
import cv2

from djitellopy import Tello

me = Tello()

me.connect()

me.streamon()

print('camera on')

frame_read = me.get_frame_read()

cv2.imwrite('camera_photo.png',frame_read.frame)

print('photo recorded')

me.streamoff()

print('camera off')

me.end()
```

- オブジェクト frame_read に瞬時的なカメラ画像が保存されている
- モジュールopencv_pythonのメンバ関数imwrite()を用いて画像ファイルを保存



画像処理

学習4-3: camera_RGB_0.py

```
import cv2
import numpy as np


def colorChange0(img):
    img_B = img.copy()
    img_B[:, :, (1, 2)] = 0
    img_G = img.copy()
    img_G[:, :, (0, 2)] = 0
    img_R = img.copy()
    img_R[:, :, (0, 1)] = 0
    img_BGR = np.concatenate((img_B, img_G, img_R), axis=1)
    return img_B, img_G, img_R, img_BGR


face_src = cv2.imread('camera_photo.png')
face_src_BGR = colorChange0(face_src)
cv2.imwrite('camera_photo_0_B.png', face_src_B )
cv2.imwrite('camera_photo_0_G.png', face_src_G )
cv2.imwrite('camera_photo_0_R.png', face_src_R )
cv2.imwrite('camera_photo_0_BGR.png', face_src_BGR )
```

- 実行してみましょう！

学習4-1で撮影した写真を使って実験をしよう！

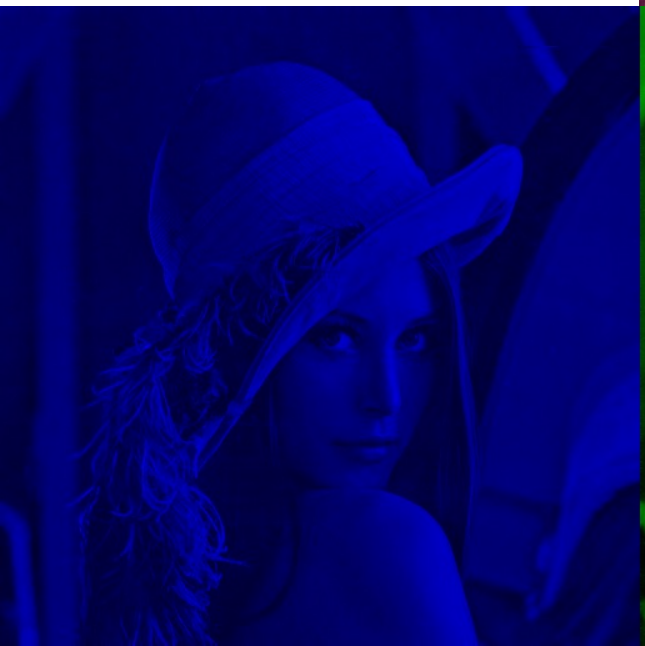
camera_photo_0_BGR.pngを開いて画像処理を確認してみよう

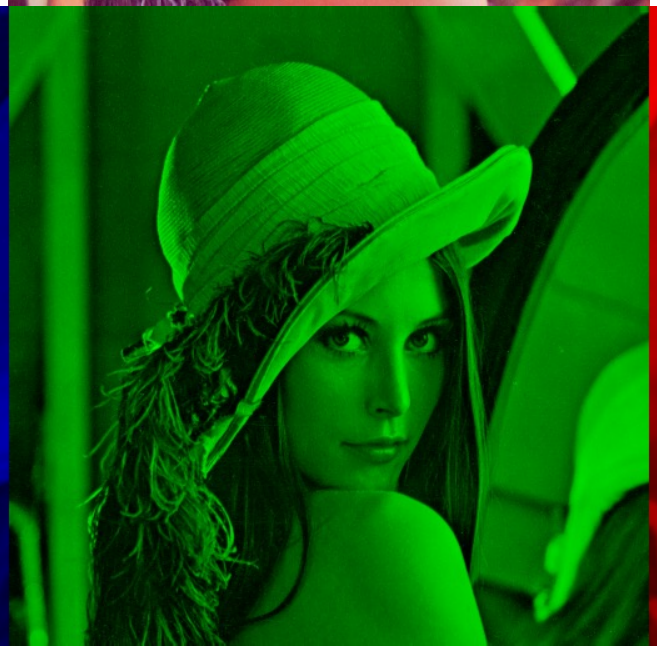
学習4-4: camera_RGB_255.py も動かしてみよう

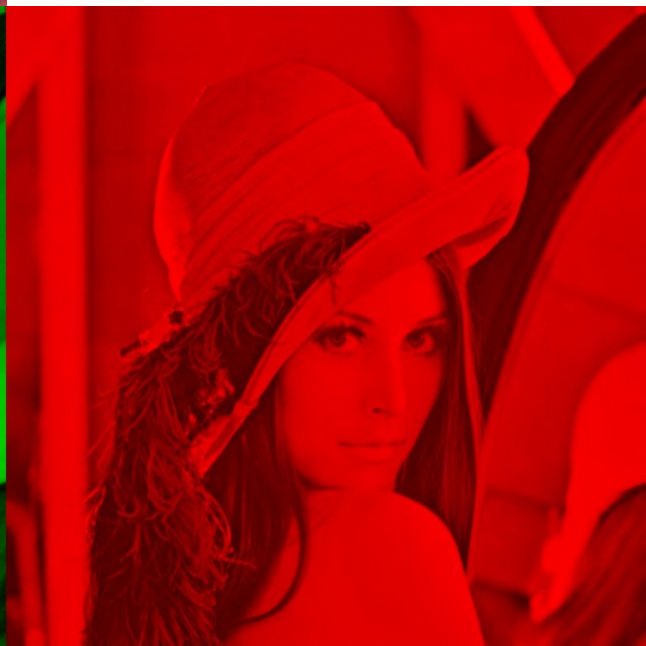
画像処理

OpenCV

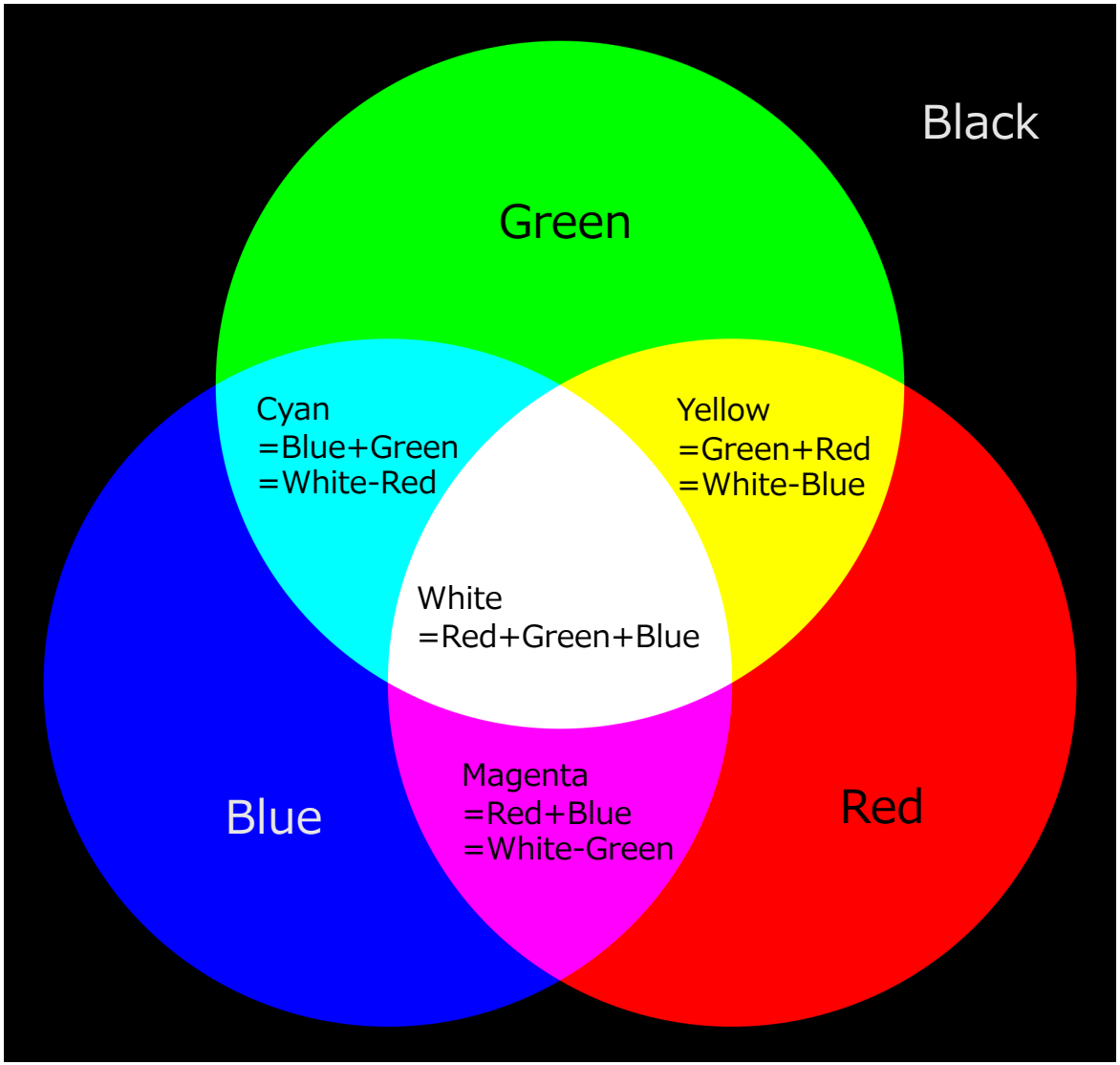
デジタル画像データを直接操作





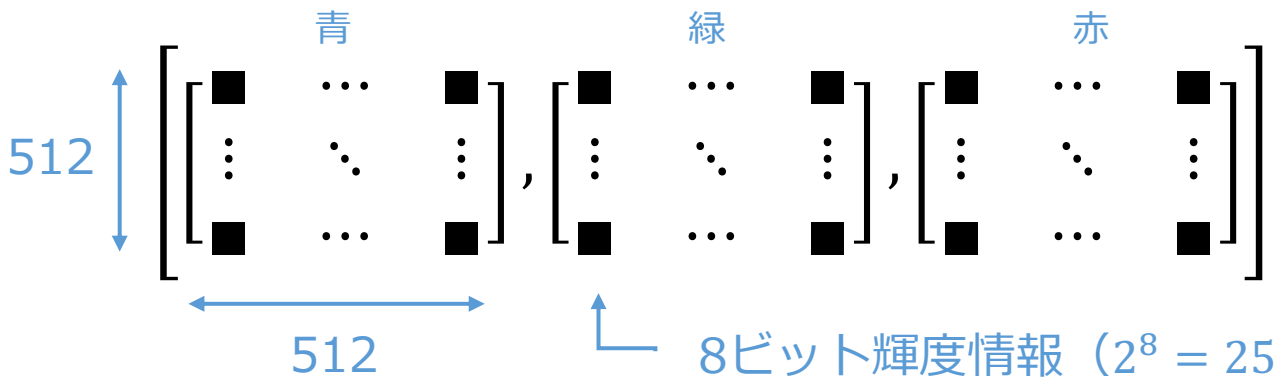


光の三原色


Black
Green
Cyan
=Blue+Green
=White-Red
Yellow
=Green+Red
=White-Blue
White
=Red+Green+Blue
Magenta
=Red+Blue
=White-Green
Blue
Red

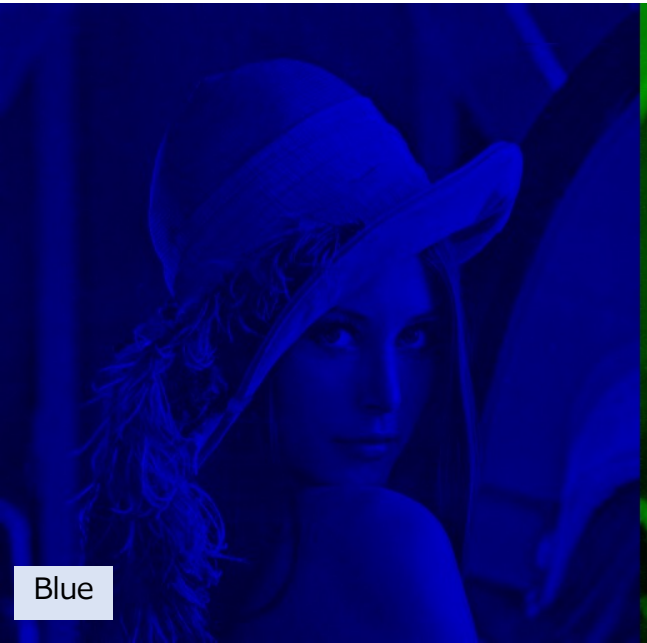
デジタル画像

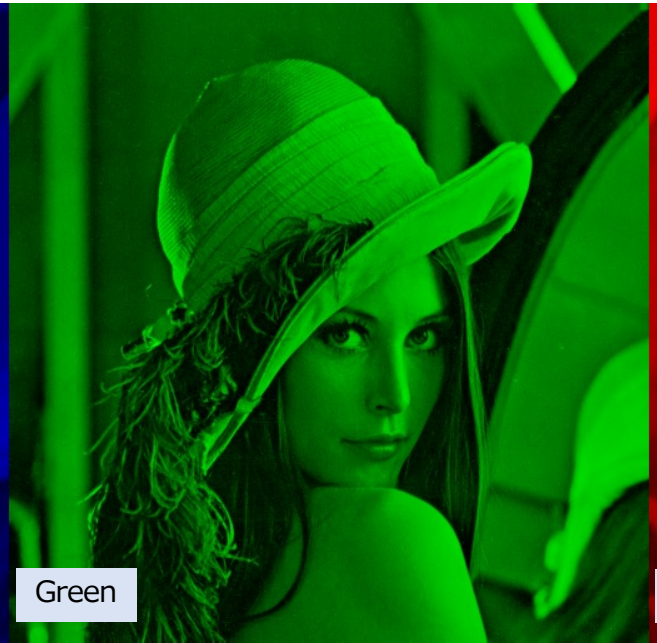
Numpy.ndarray型配列: サイズ 512× 512× 3, 要素 uint8

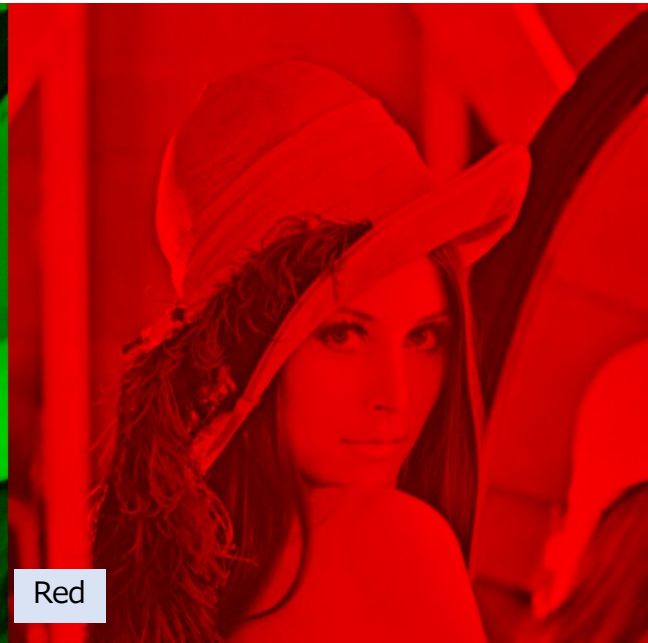


8ビット輝度情報（$2^8 = 256$）

0,1,2,…,255のいずれかの整数が入る





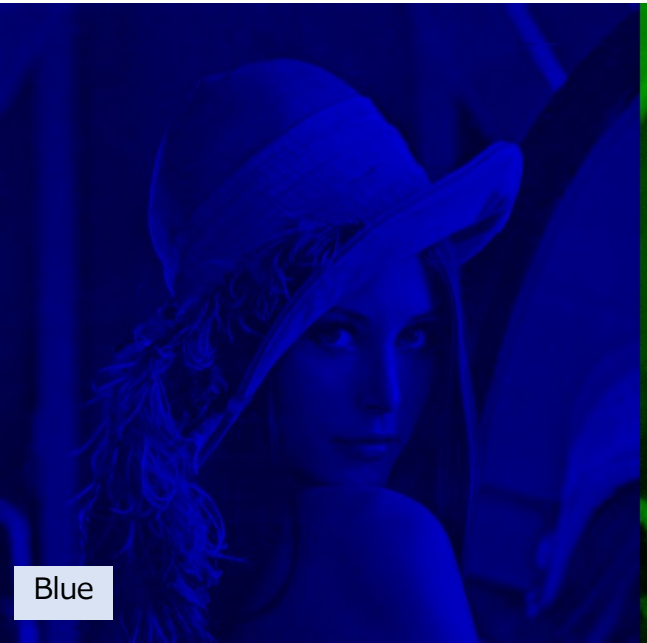


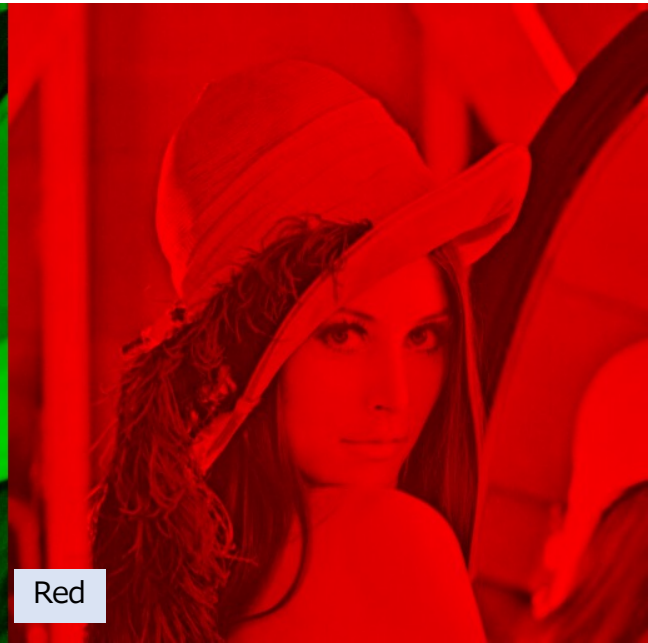
デジタル画像

Numpy.ndarray型配列: サイズ 512× 512× 3, 要素 uint8

$$\left[\left[\begin{matrix}\blacksquare & \cdots & \blacksquare \\ \vdots & \ddots & \vdots \\ \blacksquare & \cdots & \blacksquare\end{matrix}\right]_{\text{青}},\left[\begin{matrix}0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0\end{matrix}\right]_{\text{緑}},\left[\begin{matrix}0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0\end{matrix}\right]_{\text{赤}}\right]$$

青画像＝緑･赤の輝度情報がゼロ





色抽出（解説）

学習4-3: camera_RGB_0.py

```
import cv2
import numpy as np

def colorChange0(img):
    img_B = img.copy()
    img_B[:, :, (1, 2)] = 0
    img_G = img.copy()
    img_G[:, :, (0, 2)] = 0
    img_R = img.copy()
    img_R[:, :, (0, 1)] = 0
    img_BGR = np.concatenate((img_B, img_G, img_R), axis=1)
    return img_B, img_G, img_R, img_BGR

face_src = cv2.imread('camera_photo.png')
face_src_BGR = colorChange0(face_src)
cv2.imwrite('camera_photo_0_B.png', face_src_B )
cv2.imwrite('camera_photo_0_G.png', face_src_G )
cv2.imwrite('camera_photo_0_R.png', face_src_R )
cv2.imwrite('camera_photo_0_BGR.png', face_src_BGR )
```

- img_B[:, :, (1, 2)]=0により緑画像成分と赤画像成分にアクセスし，0を代入
  - img_B[:, :, 0]が青画像
  - img_B[:, :, 1]が緑画像
  - img_B[:, :, 2]が赤画像

講義概要

3限

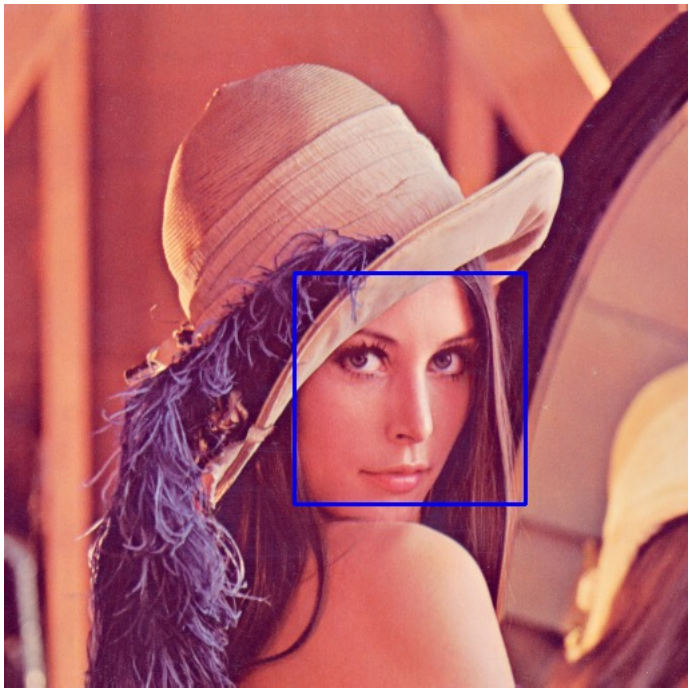
- ドローン
- 実習：飛行制御

4限

- デジタル画像
- 実習：画像処理
- **人工知能**
- **実習：顔認識**
- まとめ

顔認識

- Haar特徴ベースのCascade型分類器を使った物体検出
  - 学習済分類用データ haarcascade_frontalface_default.xml
  - 分類機 detectMultiScale()



顔認識

学習4-5: camera_detect.py

```
import cv2

face_detect_classifier = 'haarcascade_frontalface_default.xml'

face_cascade = cv2.CascadeClassifier(face_detect_classifier)


def findFace(img):

    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    faces = face_cascade.detectMultiScale(imgGray)

    for x, y, w, h in faces:

        cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)

    return img


face_src = cv2.imread('camera_photo.png')

face_src_rev, face_info = findFace(face_src)

cv2.imwrite('face_detect.png', face_src_rev )
```

- 実行してみましょう！

自分達の顔の写真に顔認識を適用してみよう！

haarcascade_frontalface_default.xml は正面から見た顔画像を検出

顔全体が明るく映るように光の加減に気を配って下さい

眉毛とか口元が映っていないと顔認識は厳しいかもしれません

顔認識（解説）

学習4-5: camera_detect.py

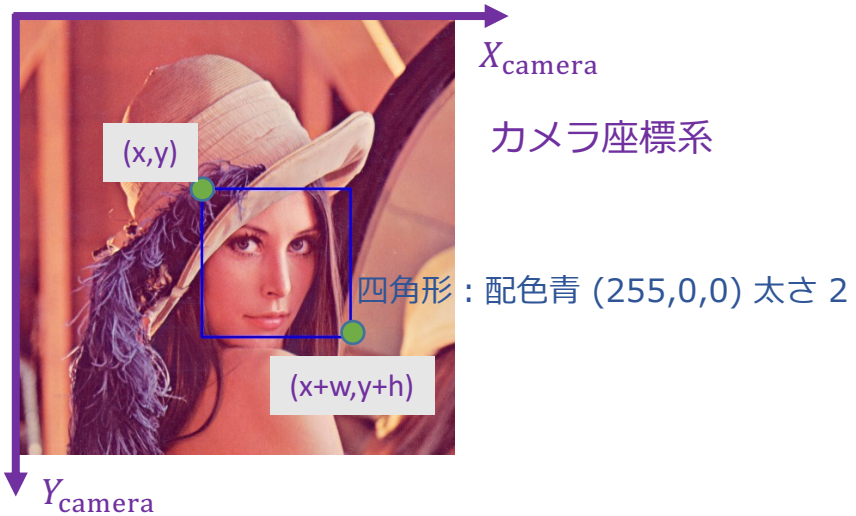
```
import cv2

face_detect_classifier = 'haarcascade_frontalface_default.xml'

face_cascade = cv2.CascadeClassifier(face_detect_classifier)


def findFace(img):

    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    faces = face_cascade.detectMultiScale(imgGray)

    for x, y, w, h in faces:

        cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)

    return img


face_src = cv2.imread('camera_photo.png')

face_src_rev = findFace(face_src)

cv2.imwrite('face_detect.png', face_src_rev )
```

- 顔認識用分類機を構成
- 前処理として白黒画像に変換してから分類機を適用
- 画像に検出顔位置を書き込む

オプションを追加可能

face_cascade.detectMultiScale(imgGray, scaleFactor=1.1, minNeighbors=3)

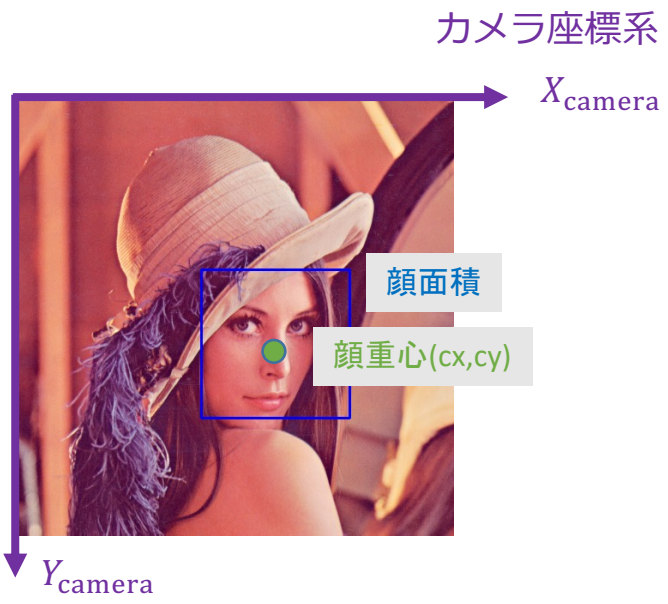
scaleFactorは1.01以上，数値を大きくすると高速化される傾向



学習4-5: camera_detect_max.py

```
import cv2
face_detect_classifier = 'haarcascade_frontalface_default.xml'
face_cascade = cv2.CascadeClassifier(face_detect_classifier)

def findBiggestFace(img):
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(imgGray)
    faceLocations = []
    faceSizes = []
    for x, y, w, h in faces:
        cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
        cx = x + w // 2
        cy = y + h // 2
        area = w * h
        faceLocations.append([cx, cy])
        faceSizes.append(area)
    if len(faceSizes) != 0:
        i = faceSizes.index(max(faceSizes))
        return img, [faceLocations[i], faceSizes[i]]
    else:
        return img, [[0, 0], 0]
face_src = cv2.imread('camera_photo.png')
face_src_rev, face_info = findBiggestFace(face_src)
print('size of face_src_rev =',face_src_rev.shape)
print('info =',face_info)
cv2.imwrite('face_detect.png', face_src_rev )
```

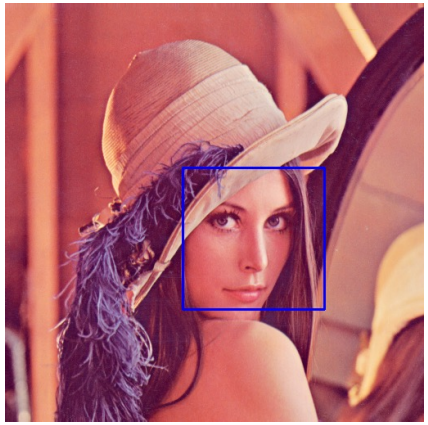
- 顔の中心位置 (cx,cy) を計算
- 顔の面積 area を計算
- 複数の顔が検出された場合に最大の顔を検出
- 顔の中心位置 (cx,cy) と面積 area をまとめて出力
  face_info = [[cx,cy],area]



計測データを追加取得 → 飛行制御と組み合わせ可能

飛行制御＋**顔認識**

ちょっこと左上にずれて見えるし遠いし
近づいた方が良いかな



飛行制御＋顔認識

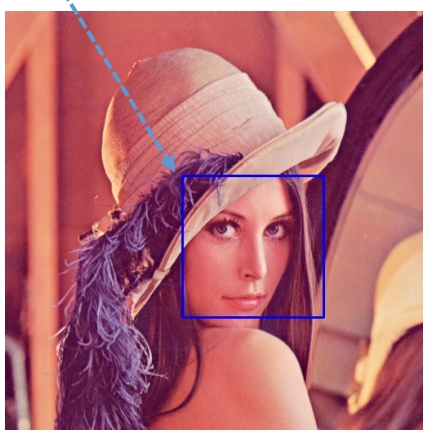
良い感じ♪

何か近づいてきたわ？

飛行制御＋**顔認識**

飛行制御＋顔認識

良い感じ♪



計測（顔認識）と制御（飛行制御）を組み合わせると・・・

ドローンで鬼ごっこ♪

課題

配布資料を参考に課題に取り組んで下さい

- **課題4-6：　　ドローンを飛ばして自撮り♪**
  - 必要に応じて move_up() と move_down()を用いて高度調整
- **課題4-7：　　課題4-6＋顔認識♪**

- 学習4-2：　　ビデオ撮影
- 課題4-8：　　作業4-2のビデオデータに顔認識を実施

- **課題4-9：　　鬼ごっこ飛行のサンプルプログラムの動作検証と解読**
  - 解読の鍵は自作関数 trackface() の中の組込み関数 send_rc_control()
  - ビデオウィンドウ上で「q」を押したら自動着陸

プログラムが正しく記載されていても通信不良のために記載通りに実行しないことがあります

→ Shellに赤字エラーが出力されなければ**再度実行**して下さい

解読：鬼ごっこ飛行

@djitellopy API reference

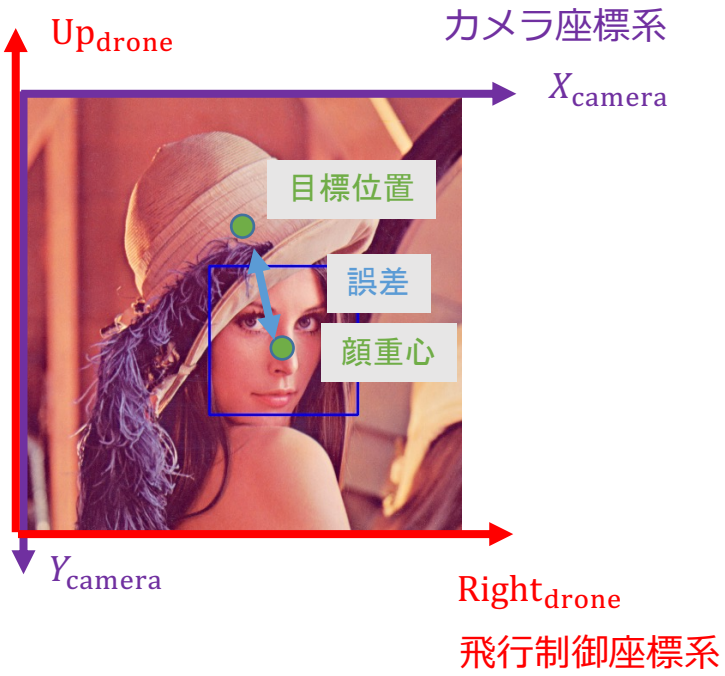
```
send_rc_control(self, left_right_velocity,
forward_backward_velocity, up_down_velocity,
yaw_velocity)
```

Send RC control via four channels. Command is sent every self.TIME_BTW_RC_CONTROL_COMMANDS seconds.

Parameters:

Name	Type	Description	Default
`left_right_velocity`	`int`	-100~100 (left/right)	*required*
`forward_backward_velocity`	`int`	-100~100 (forward/backward)	*required*
`up_down_velocity`	`int`	-100~100 (up/down)	*required*
`yaw_velocity`	`int`	-100~100 (yaw)	*required*

関数 send_rc_control() により定期的に速度・角速度指令値を送信し続けることができる



要点：顔画像が目標位置に来るように定期的に速度・角速度指令値を送信

自作関数 trackface() の中の me.send_rc_control() により

顔面積 area と顔重心 (cx,cy) の理想値からの誤差を解消している

カメラ座標系と飛行制御座標系の座標軸の違いを吸収するために座標変換が必要

講義概要

3限

- ドローン
- 実習：飛行制御

4限

- デジタル画像
- 実習：画像処理
- 人工知能
- 実習：顔認識
- **まとめ**

まとめ

- **Pythonを用いた実習型AIドローン講義**
  - プログラミングによって様々な基本機能を実現でき、さらにそれらの**組み合わせにより高機能**を実現できる
  - 工夫次第では様々な課題研究を設定できて**楽しい♪**