石川県立金沢泉丘高等学校

金沢大学
KANAZAWA UNIVERSITY

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12月21日 (水) 13時00分~16時00分

プログラミング指導: 久保守・今井祐希

アンケート

- 教材の評価・改善のために事前・事後アンケートご協力をお願いします
  - 強制ではありません。ご協力頂けると嬉しいです
  - 解析時の照合のために**出席番号**と**PC番号**のご記入をお願いします
  - 事後アンケートにおける**自由記述**は是非色々とご記入ください
    大変有益な情報として参考にさせて頂きます

配布資料

- 講義資料　映写資料の写し
- 実習用サンプルプログラム
  - 1_学習_飛行制御　指示に従い実行
  - 2_課題_飛行制御　課題が難しければ参照
  - 3_学習_人工知能　指示に従い実行
  - 4_課題_人工知能　課題が難しければ参照
  - 5_参考_瞳検出　興味ある人は顔認識とともに試して下さい

フォルダをダブルクリックして
順番に指定ファイルを開きます

講義の流れ

わからなくて当然
気軽に質問してみよう

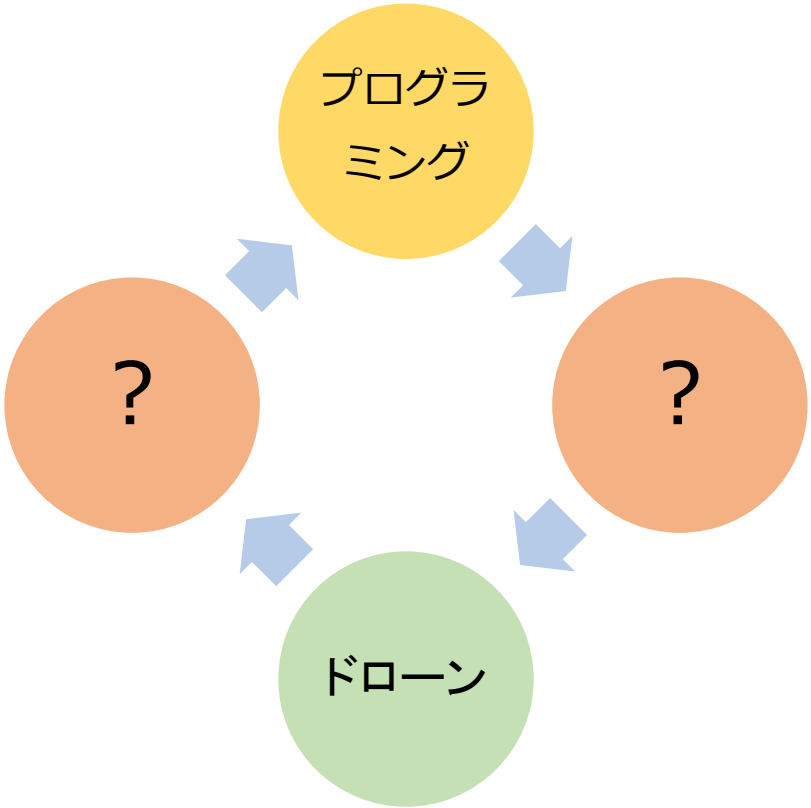
パソコンの画面を皆で
見ながら相談しよう

パソコン作業する人と
資料を見ながら作戦を練る人に
分かれると効率良いかも

講義の目的

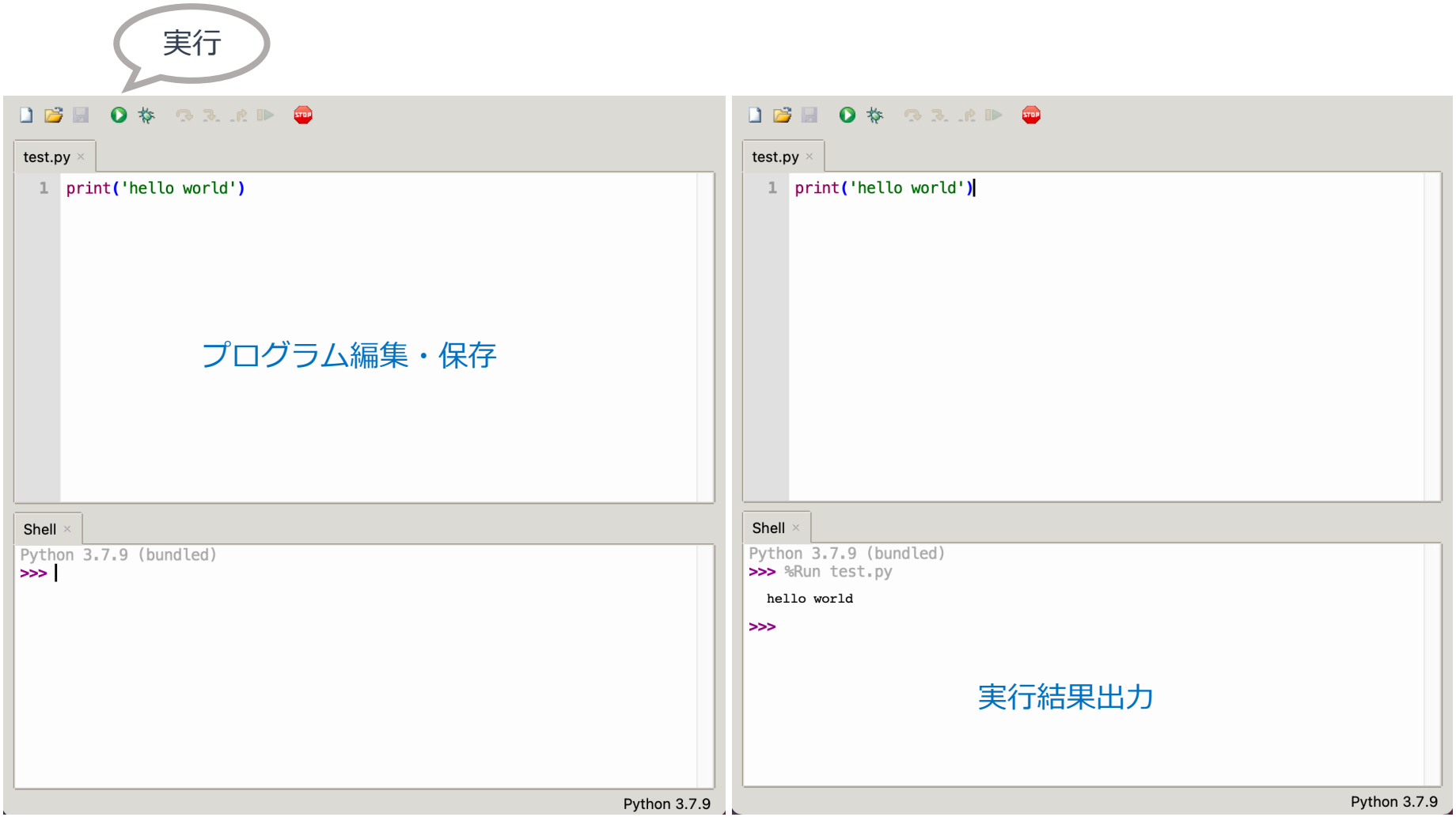
ドローンを使って学びながら遊んでみよう♪

- プログラミング技法の詳細にはとらわれず**可能性**を感じてください
- 基本機能の組み合わせにより**機能拡張**できることを体感しよう



Thonny

- Python統合開発環境：



講義概要

前半

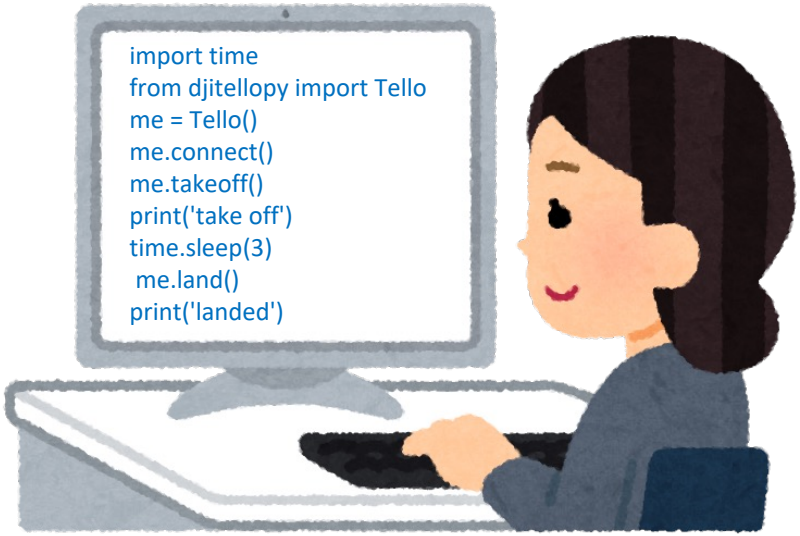
- **ドローン**
- 学習：飛行制御
- 課題：飛行制御

後半

- デジタル画像
- 学習：人工知能（画像処理）
- 人工知能
- 学習：人工知能（顔認識）
- 課題：人工知能
- 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
  - 飛行は 広い場所 & 下が平坦 が基本
  - 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：ドローン使用時は**電灯点灯**でお願いします

講義概要

前半

- ドローン
- **学習：飛行制御**
- 課題：飛行制御

後半

- デジタル画像
- 学習：人工知能（画像処理）
- 人工知能
- 学習：人工知能（顔認識）
- 課題：人工知能
- まとめ

バッテリ残量確認

1_1_battery.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！
  - 「実習用サンプルプログラム」の下の「1_学習用_飛行制御」からサンプルプログラム 1_1_battery.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

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の緑色三角実行ボタンを押してプログラムを**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~~
   - **1_1_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

プログラム実行中に変化

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

1_1_battery.py

```
from djitellopy import Tello

me = Tello()

me.connect()

print('connected')

print(me.get_battery())

me.end()

print('terminated')
```

書き換えて好きな文言を出力してみよう
例えば，print(‘The End’)

- 再度実行してみましょう！

バッテリ残量確認（解説１）

```
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

雛形の束

雛形

モジュールを追加することによりPythonの機能を拡張できます

バッテリ残量確認（解説２）

```
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

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

バッテリ残量確認（解説３）

```
from djitellopy import Tello

me = Tello()

me.connect()

print('connected')

print(me.get_battery())

me.end()

print('terminated')
```

- 関数connect()を実行すると様々な**コマンドをtelloに送信可能**となります（紫色点灯）
- 関数**print**により動作確認用の画面出力.

バッテリ残量確認（解説４）

```
from djitellopy import Tello

me = Tello()

me.connect()

print('connected')

print(me.get_battery())

me.end()

print('terminated')
```

- 関数get_battery()を呼び出し、関数printと組み合わせて充電率を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）

```
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を終了します

離陸・着陸

1_2_takeoff.py

```
import time

from djitellopy import Tello

me = Tello()

me.connect()

me.takeoff()

print('take off')

time.sleep(3)

me.land()

me.end()

print('terminated')
```

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

離陸・着陸（解説）

```
import time
from djitellopy import Tello
me = Tello()
me.connect()
me.takeoff()
print('take off')
time.sleep(3)
me.land()
me.end()
print('terminated')
```

- takeff() は離陸
- sleep() は待機
  - 引数は待機時間[秒]
- land() は着陸

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

ただし，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が基本 → 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

1_3_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()

print('terminated')
```

- 実行してみましょう！

離陸・着陸のプログラムとの違いがどこにあるか確認してみましょう

離陸・前進・後退・着陸（解説）

```
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()

print('terminated')
```

- move_forward() は前進
  - 引数は20から500の間の整数
  - 移動距離[cm]を表します
- move_back() は後退
  - 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

1_4_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()

print('terminated')
```

- 実行してみましょう！

離陸・前進・交代・着陸のプログラムとの違いがどこにあるか確認してみましょう

離陸・回転・着陸（解説）

```
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()

print('terminated')
```

- rotate_clockwise() は時計回り回転
  - 引数は1から360の間の整数
  - 回転角度[deg]を表します
- rotate_counter_clockwise() は反時計回り回転
  - 引数1から360の間の整数
  - 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
- **選択**
  - If文により処理を分岐できる

プログラム

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

条件部

実行部

必要な機能に差替

実行結果

xは正

3>0なのでx>0の実行部を選択

複数の条件から選択

- **繰返し**
  - For文により処理を繰り返せる

条件部

プログラム

```
for k in range(3):
    print('Morning mum')
    print('I'm hungry')
```

実行部

必要な機能に差替

実行結果

Morning mum
I'm hungury
k=0の実行結果

Morning mum
I'm hungury
k=1の実行結果

Morning mum
I'm hungury
k=2の実行結果

3回繰り返し

講義概要

前半

- ドローン
- 学習：飛行制御
- **課題：飛行制御**

- 出題順に関わらず自由に課題に取り組んで下さい
- **「2_課題_飛行制御」に解答記入用の白紙ファイル**を用意しましたので、解答プログラムを記入して下さい
- 必要に応じて「1_学習_飛行制御」からプログラムを転記して下さい
- ヒントが欲しい場合には「模範回答」を参照していただいても構いません

後半

- デジタル画像
- 学習：人工知能（画像処理）
- 人工知能
- 学習：人工知能（顔認識）
- 課題：人工知能
- まとめ

Table of contents

djitellopy.tello.Tello

connect()

connect_to_wifi()

curve_xyz_speed()

curve_xyz_speed_mid()

disable_mission_pads()

emergency()

enable_mission_pads()

end()

flip()

宙返り
- flip_back()
- flip_forward()
- flip_left()
- flip_right()

get_acceleration_x()

get_acceleration_y()

get_acceleration_z()

get_barometer()

get_battery()

get_current_state()

get_distance_tof()

get_flight_time()

get_frame_read()

get_height()

get_highest_temperature()

get_lowest_temperature()

get_mission_pad_distance_x()

get_mission_pad_distance_y()

get_mission_pad_distance_z()

get_mission_pad_id()

get_own_udp_object()

get_pitch()

get_roll()

get_speed_x()

get_speed_y()

get_speed_z()

get_state_field()

get_temperature()

get_udp_video_address()

get_video_capture()

get_yaw()

go_xyz_speed()

go_xyz_speed_mid()

go_xyz_speed_yaw_mid()

land()

move()

平行移動
- move_back()
- move_down()
- move_forward()
- move_left()
- move_right()
- move_up()

parse_state()

query_attitude()

query_barometer()

query_battery()

query_distance_tof()

query_flight_time()

query_height()

query_sdk_version()

query_serial_number()

query_speed()

query_temperature()

query_wifi_signal_noise_ratio()

raise_result_error()

回転
- rotate_clockwise()
- rotate_counter_clockwise()

send_command_with_return()

send_command_without_return()

send_control_command()

send_rc_control()

send_read_command()

send_read_command_float()

send_read_command_int()

set_speed()

set_wifi_credentials()

streamoff()

streamon()

takeoff()

udp_response_receiver()

udp_state_receiver()

自由課題も可！

@djitellopy API reference https://djitellopy.readthedocs.io/en/latest/

往復運動

2_1_roundtrip.py (新規作成)

下記の機能を実現するプログラムを作成して下さい.

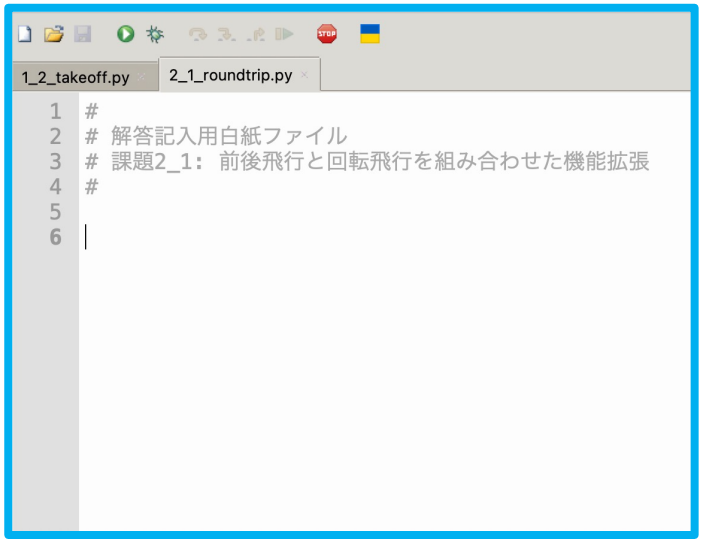
1. 離陸 } 1_2_takeoff.pyと共通
2. 30cm前進
3. 2秒待機
4. 90deg時計回り回転
5. 3秒待機
6. 90deg半時計回り回転
7. 2秒待機
8. 30cm後退
9. 2秒待機

2〜9 } 1_3_move.pyと1_4_rotate.pyを参考に新規作成

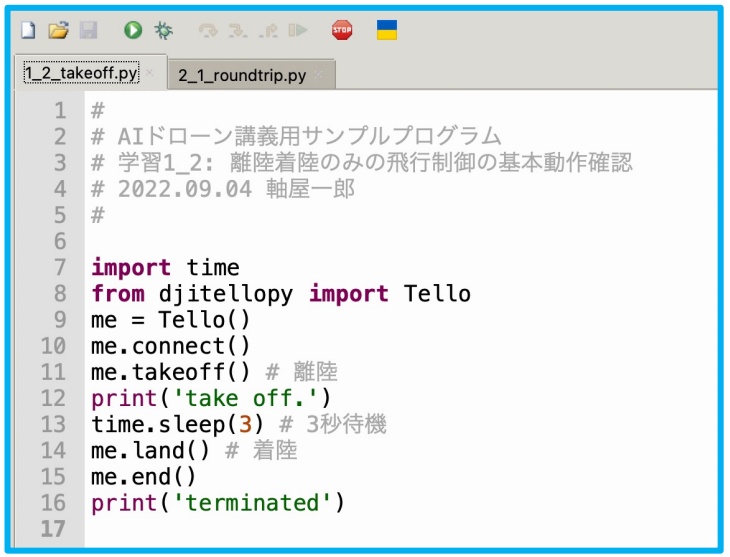
10. 着陸 } 1_2_takeoff.pyと共通

新規プログラム作成の一例

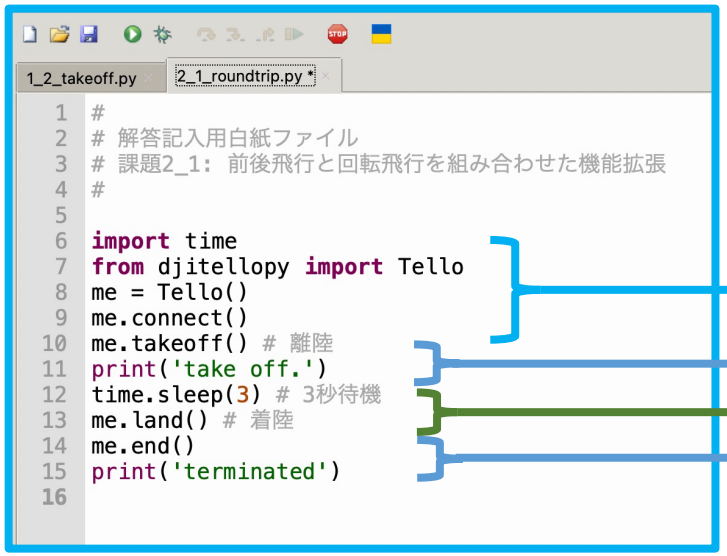
1) 記入用ファイルを開く
フォルダのファイルをダブルクリック



2) 参照元ファイルを開く
転記箇所をマウスでハイライトさせてから**コピー(Ctrl+C)**



3) 記入用ファイルに転記する
転記箇所にて**ペースト(Ctrl+V)**



4) 新規作成箇所を編集

Pythonプログラム編集時の留意事項：
- 文の冒頭を揃える必要があります．文頭に余計な半角スペースなどが入るとエラーが発生します
- If文・For文などの条件部はコロン：で締めます．実行部はインデントによりコードブロックを規定します．
- プログラムは半角文字で記載します．
- シングルクオーテーションで区切られた文字列は全角文字の使用を許容します
- #はコメントアウトを意味し，補足説明などを記載できます．

モジュール読込みなどの準備が必要

1_2_takeoff.pyと共通

1_3_move.pyと1_4_rotate.pyを参考に書換え

1_2_takeoff.pyと共通

反復横跳び

2_2_jump.py (新規作成)

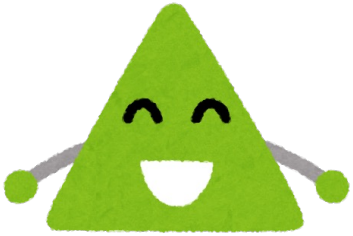
telloに各幅30cmで5回反復横跳びをさせてみよう.

右飛行: move_right(), 左飛行: move_left()

正三角形飛行

2_3_triangle.py (新規作成)

telloに一辺50cmの正三角形を描かせてみよう.



バッテリ確認後飛行

2_4_batcheck.py (新規作成)

1_1_battery.pyと1_2_takeoff.pyにIf-Else文を組み合わせて，充電率30%未満なら「NG !!!」と表示，それ以外なら離着陸を行うプログラムを作成してみよう.

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

講義概要

前半

- ドローン
- 学習：飛行制御
- 課題：飛行制御

後半

- **デジタル画像**
- **学習：人工知能（画像処理）**
- 人工知能
- 学習：人工知能（顔認識）
- 課題：人工知能
- まとめ

カメラ画像撮影

3_1_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！

モデルとPC操作担当を分担しましょう．複数人モデルも可

逆光だと顔が真っ暗になります．顔全体が明るく映るよう工夫しましょう

画像データをバックアップする場合はファイル名 camera_photo.png を書き換えて下さい

カメラ画像撮影（解説）

```
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()
```

CV とは Computer Vision の略称
コンピュータによる視覚 を意味します

OpenCV

TelloからPCにストリーミング中継が行われます
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）

```
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()
```

- Djitellopy を用いて瞬時的カメラ画像を切り出す
- OpenCV を用いて画像ファイルを保存

関数を挿入することにより機能が追加されています

画像処理

3_2_RGB0.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_OB = np.concatenate((img,img_B), axis=1)
    img_GR = np.concatenate((img_G, img_R), axis=1)
    img_OBGR = np.concatenate((img_OB, img_GR), axis=0)
    return img_OBGR


face_src = cv2.imread('camera_photo.png')
face_src_OBGR = colorChange0(face_src)
cv2.imwrite('camera_photo_0_OBGR.png', face_src_OBGR )
```

- 実行してみましょう！

3_1_photo.pyで撮影した写真を使って実験をしよう！

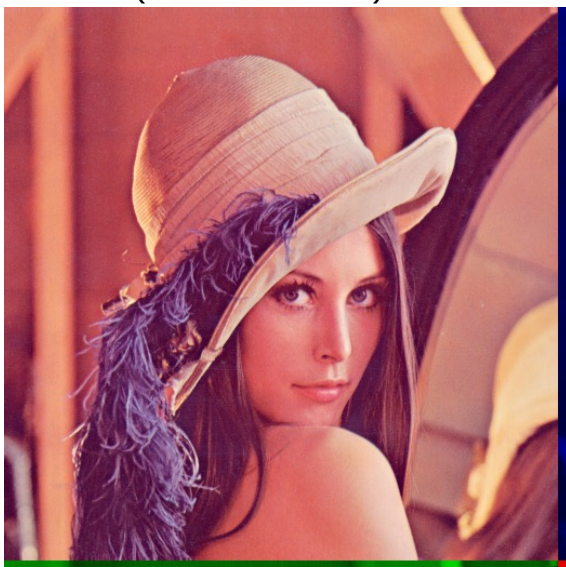
camera_photo_0_BGR.pngを開いて画像処理の効果を確認しよう

オプション: 3_2_RGB_255.py も動かしてみよう

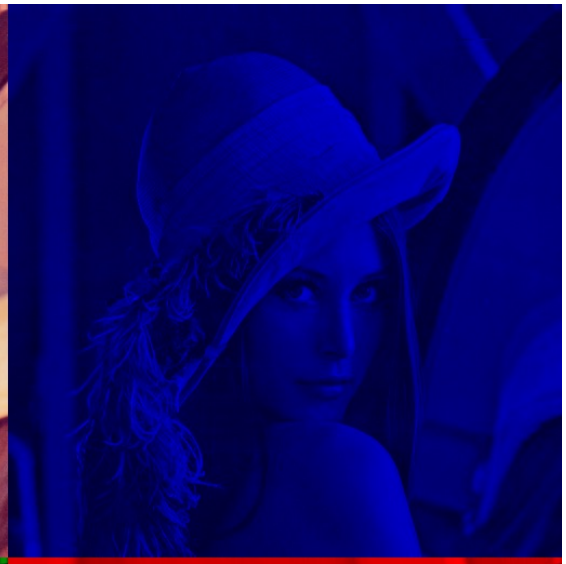
画像処理

※ Lena (標準テスト画像)

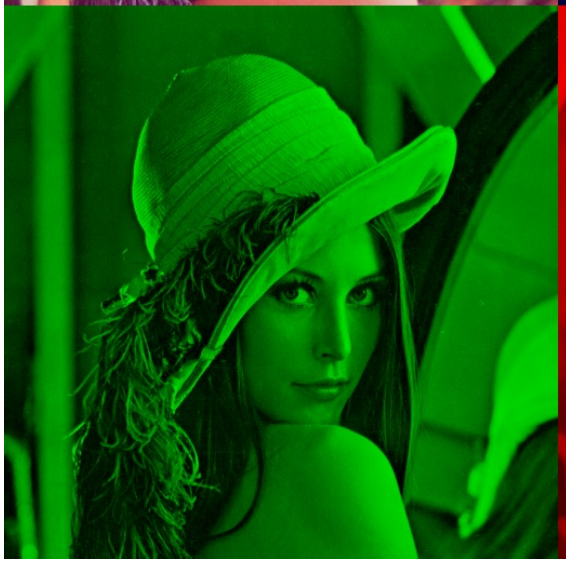
本当の私♪



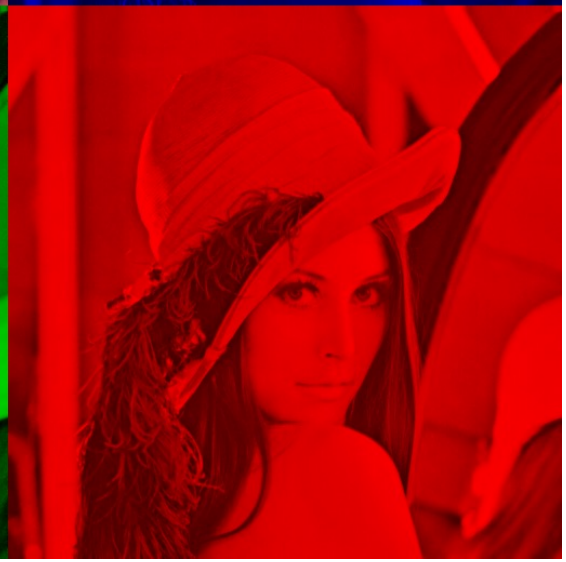
青♪



緑♪

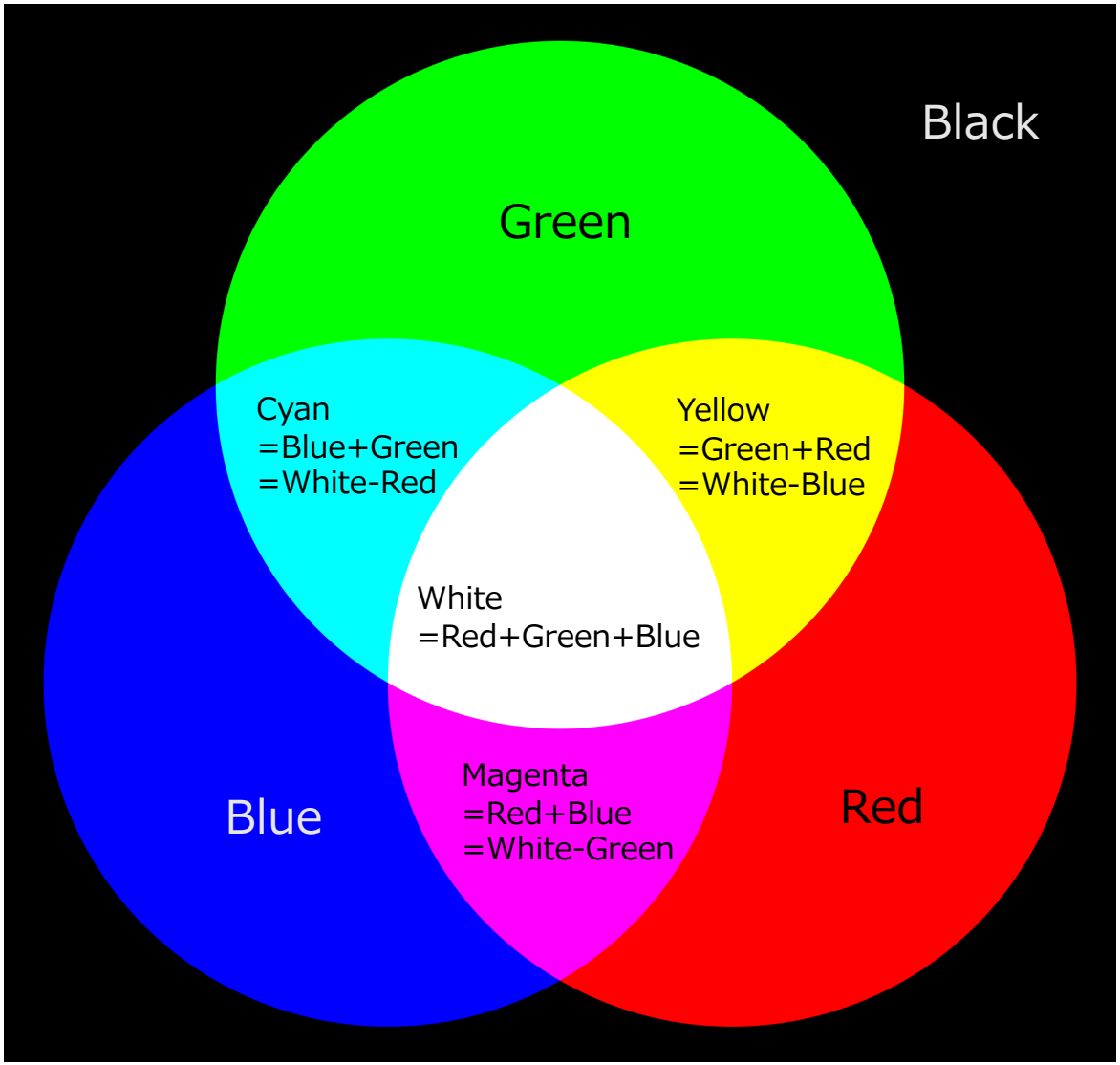


赤♪



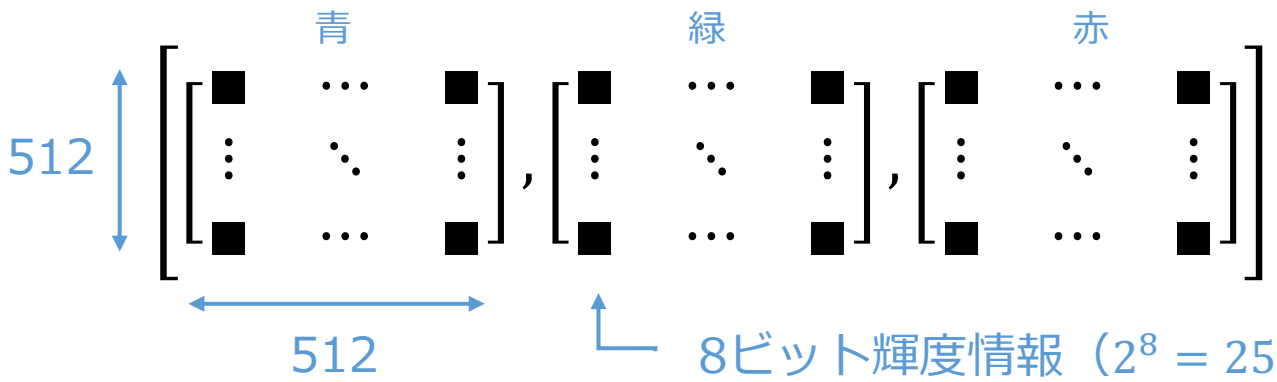
デジタル画像データを直接操作

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

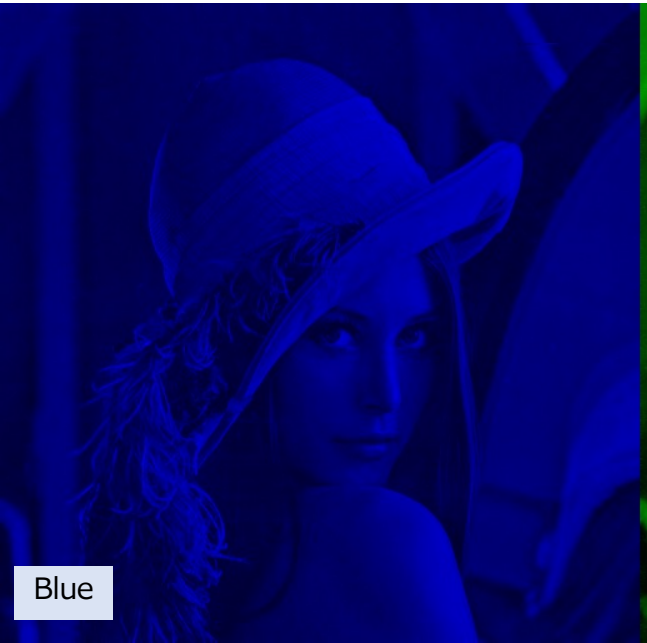
デジタル画像

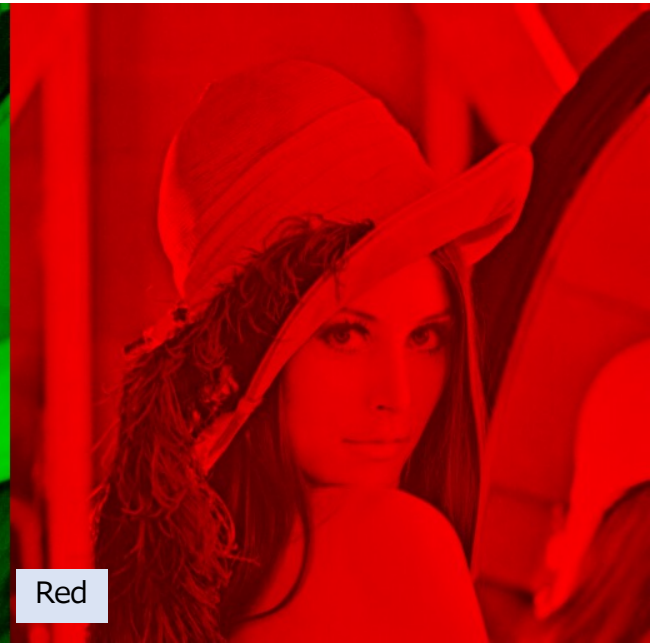
配列：数字を格納する箱



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



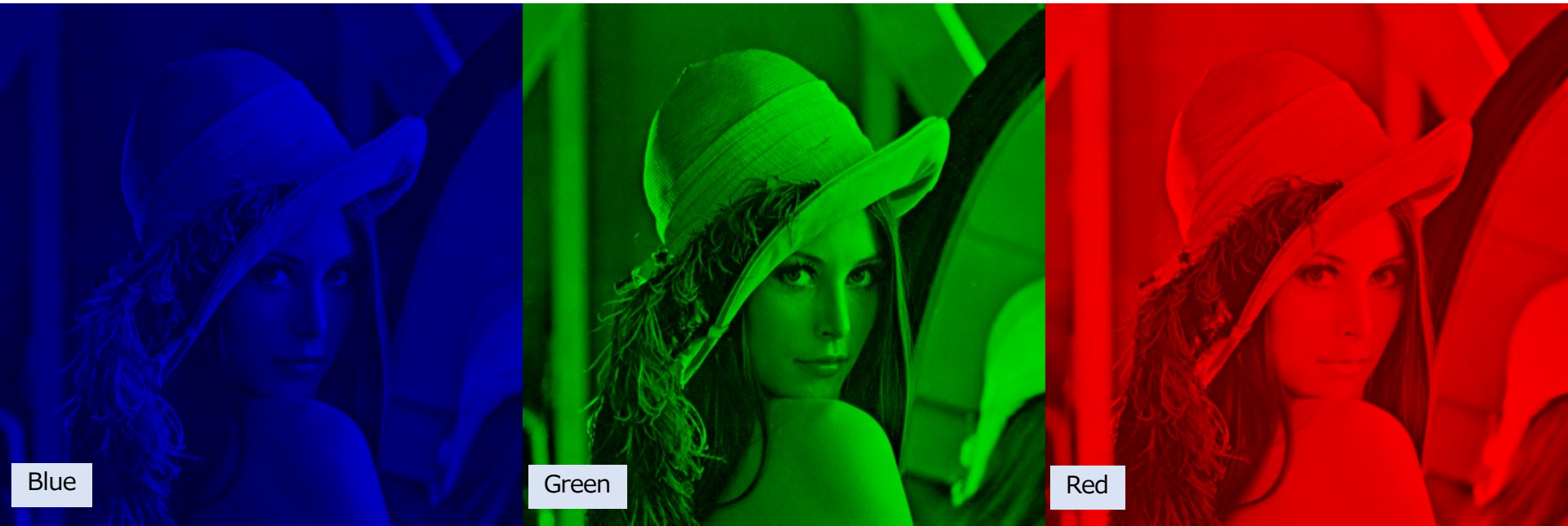


デジタル画像

配列：数字を格納する箱

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

青画像＝緑・赤の輝度情報をゼロに書換え



色抽出（解説）

```
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_OB = np.concatenate((img,img_B), axis=1)

    img_GR = np.concatenate((img_G, img_R), axis=1)

    img_OBGR = np.concatenate((img_OB, img_GR), axis=0)

    return img_OBGR


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

face_src_OBGR = colorChange0(face_src)

cv2.imwrite('camera_photo_0_OBGR.png', face_src_OBGR )
```

- 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]が青画像成分
  - img_B[:, :, 1]が緑画像成分
  - img_B[:, :, 2]が赤画像成分

関数は必要な時に呼び出されます

関数呼び出し：
画像を関数に渡し加工画像を受け取る

講義概要

前半

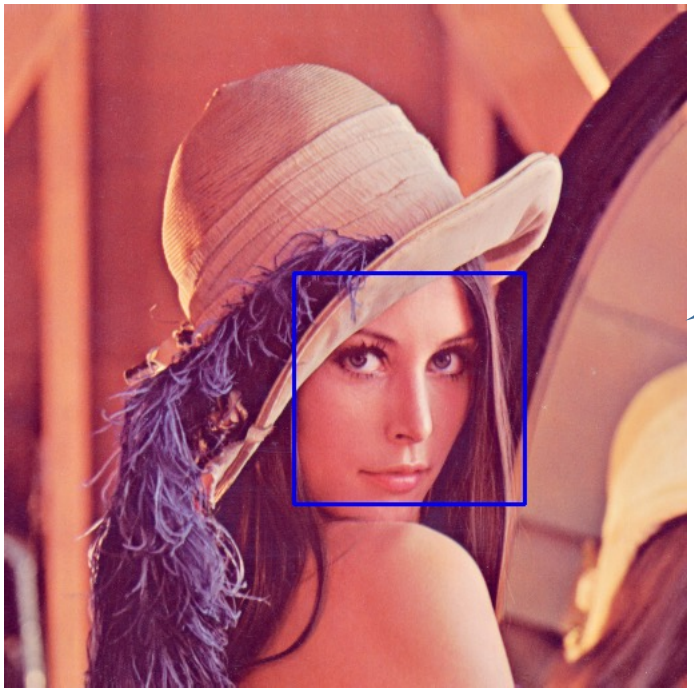
- ドローン
- 学習：飛行制御
- 課題：飛行制御

後半

- デジタル画像
- 学習：人工知能（画像処理）
- **人工知能**
- **学習：人工知能（顔認識）**
- 課題：人工知能
- 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()



顔の識別は人間には簡単だけど…
画像を数字の組み合わせで
表す時には難しい

顔認識

3_3_detect.py

```
import cv2

face_detect_classifier = 'haarcascade_frontalface_default.xml'

face_cascade = cv2.CascadeClassifier(face_detect_classifier)


def findFace(img):

    faces = face_cascade.detectMultiScale(img)

    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 )
```

- 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）

```
import cv2

face_detect_classifier = 'haarcascade_frontalface_default.xml'

face_cascade = cv2.CascadeClassifier(face_detect_classifier)
```

- 顔認識用分類機を構成
- 分類機を適用
- 画像に検出顔位置を書き込む

```
def findFace(img):
    faces = face_cascade.detectMultiScale(img)
    for x, y, w, h in faces:
        cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
    return img
```

オプションを追加可能

face_cascade.detectMultiScale(img, 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

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

face_src_rev = findFace(face_src)

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

カメラ座標系

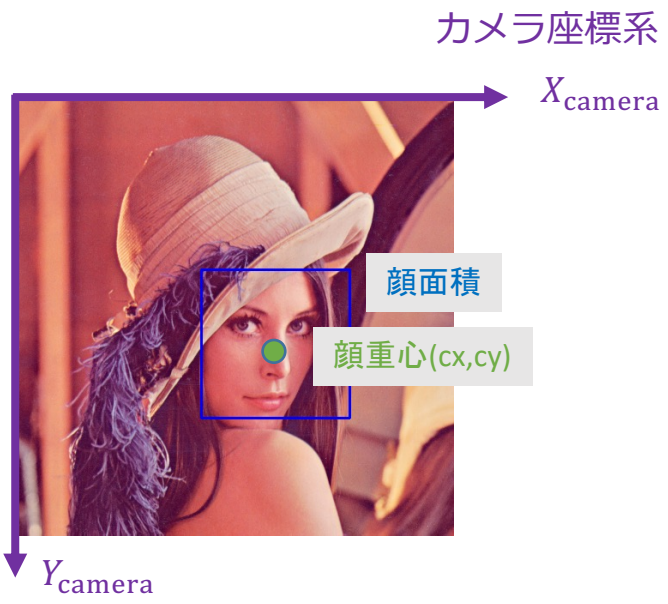
四角形：配色青 (255,0,0) 太さ 2

3_4_detect_max.py

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


def findBiggestFace(img):
    faces = face_cascade.detectMultiScale(img)
    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 )
```

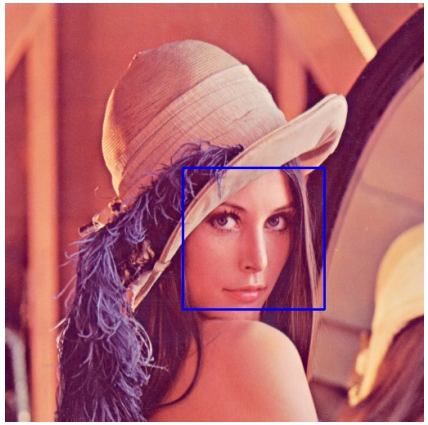
- 顔の中心位置 (cx,cy) を計算
- 顔の面積 area を計算
- 複数の顔が検出された場合に最大の顔を検出
- 顔の中心位置 (cx,cy) と面積 area をまとめて出力
  face_info = [[cx,cy],area]



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

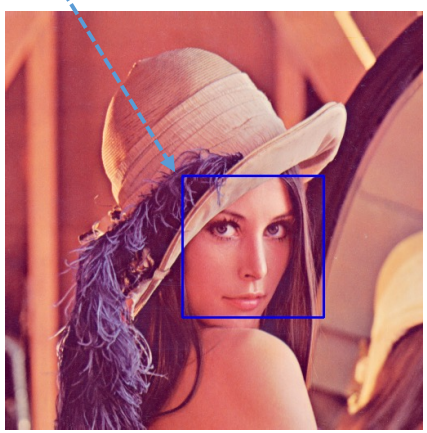
良い感じ♪

何か近づいてきた？

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♪

講義概要

前半

- ドローン
- 学習：飛行制御
- 課題：飛行制御

後半

- デジタル画像
- 学習：人工知能（画像処理）
- 人工知能
- 学習：人工知能（顔認識）
- **課題：人工知能**
- まとめ

- 4_1と4_2は**「4_課題_人工知能」に解答記入用の白紙ファイル**を用意しましたので、解答プログラムを記入して下さい
- ヒントが欲しい場合には「模範回答」を参照していただいても構いません
- 4_3はサンプルプログラムを動作させてからプログラムの解読を試みて下さい

わからなくて当然
気軽に質問してみよう

ドローンで自撮り飛行♪

4_1_fly_photo.py (新規作成)

ドローンを飛ばして写真撮影を行うプログラムを作成.

- 1_2_takeoff.py と 3_1_photo.py を組合わせる.
- 50cm高度追加には me.move_up(50) を挿入

飛ばして顔認識♪

4_2_fly_detect.py (新規作成)

ドローンを飛ばして撮影してから，顔認識を行なった写真を保存.

- 3_3_detect.py と 4_1_fly_photo.py を組合わせる.
- 取得画像を findFace() に渡す

ドローンで鬼ごっこ♪

4_3_visual_tracking.py (動作検証)

サンプルプログラムの動作検証.

- me.move_up(80) は身長160cm相当.
- **ビデオウィンドウ上で「q」を押したら自動着陸**

機能を組み合わせる（自撮り飛行）

1_2_takeoff.py | 3_1_photo.py

```
1  #
2  # AIドローン講義用サンプルプログラム
3  # 学習1_2: 離陸着陸のみの飛行制御の基本動作確認
4  # 2022.09.04 軸屋一郎
5  #
6
7  import time
8  from djitellopy import Tello
9  me = Tello()
10 me.connect()
11 me.takeoff() # 離陸
12 print('take off.')
13 time.sleep(3) # 3秒待機
14 me.land() # 着陸
15 me.end()
16 print('terminated')
17
```

モジュールをプログラム冒頭に記載

初期化

飛行制御

終了（print文は確認のために入れてますが必須ではありません）

高度が不足するなら me.move_up(80) を挿入

1_2_takeoff.py | 3_1_photo.py

```
1  #
2  # AIドローン講義用サンプルプログラム
3  # 学習3_1: tello内蔵カメラでカメラ撮影
4  # 2022.09.04 軸屋一郎
5  #
6
7  import cv2 # opencvを読込む
8  from djitellopy import Tello
9  me = Tello()
10 me.connect()
11 me.streamon() # ビデオストリーミング開始
12 print('camera on')
13 frame_read = me.get_frame_read() # 画像取得
14 cv2.imwrite('camera_photo.png',frame_read.frame) # 取得画像を保存
15
16 print('photo recorded')
17 me.streamoff() # ビデオストリーミング終了
18 print('camera off')
19 me.end()
20
```

モジュールをプログラム冒頭に記載

初期化

カメラ撮影

終了

4_1_fly_photo.py (新規作成)

考え方：

4_1_fly_photo.pyに

1) モジュール読込み

2) 初期化

3) 飛行制御＋カメラ撮影

4) 終了

の順に記載します。

飛行制御＋カメラ撮影は**順次**の考えに従い処理順に関数を並べます。

機能を組み合わせる（飛ばして顔認識）

3_3_detect.py　4_1_fly_photo.py

```
1  #
2  # AIドローン講義用サンプルプログラム
3  # 学習3_3: 学習3_1で撮影したカメラ画像を用いて顔認識を実施（顔全体を明るく撮影）
4  # 2022.09.10 軸屋一郎
5  #
6
7  import cv2
8  face_detect_classifier = 'haarcascade_frontalface_default.xml' # Ha
9  face_cascade = cv2.CascadeClassifier(face_detect_classifier) # 分類器
10
11 def findFace(img): # imgは顔検出を行いたい画像データ
12     faces = face_cascade.detectMultiScale(img) # 分類機を適用
13     for x, y, w, h in faces: # リストfacesを順番に読み込む
14         cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
15         # 第四引数の(255, 0, 0)はカラーBGRを指定，第五引数の2は線の太さを指定
16     return img # 検出顔を四角で書き込んだimgを保存
17
18 face_src = cv2.imread('camera_photo.png') # 画像を読み込み
19 face_src_rev = findFace(face_src) # 顔検出を実施
20 cv2.imwrite('face_detect.png', face_src_rev ) # 顔検出後の画像データを画
21
```

モジュールをプログラム冒頭に記載

分類機と顔認識の関数を**準備**

顔認識を**実行**

4_2_fly_detect.py (新規作成)

考え方：　4_1_fly_photo.pyに顔認識の機能を追加します。

分類機と顔認識の顔認識の関数は使う前に準備します。Tello()の実行前に記載して構いません。

顔認識の実行は**順次**の考えに従い処理順に記載します。具体的にはカメラ画像を保存した後で実行します。

解読：鬼ごっこ飛行

@djitellopy API reference

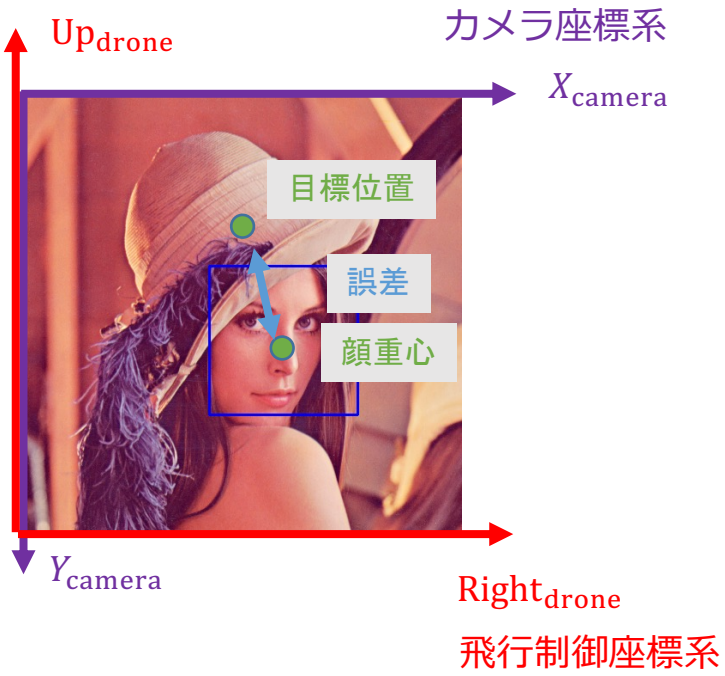
```
send_rc_control(self, left_right_velocity,
forward_backward_velocity, up_down_velocity,
yaw_velocity)
```

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

前半

- ドローン
- 学習：飛行制御
- 課題：飛行制御

後半

- デジタル画像
- 学習：人工知能（画像処理）
- 人工知能
- 学習：人工知能（顔認識）
- 課題：人工知能
- **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♪**
  - **本講義をきっかけ**として**自分でもプログラムを書いて遊んでみたい**と興味を抱く高校生が現れるなら幸いです