スーパーサイエンスハイスクール事業

石川県立泉丘高等学校理数科1年

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9月13日 (火)6・7限(14時5分~15時55分)

石川県立泉丘高等学校 iStudio

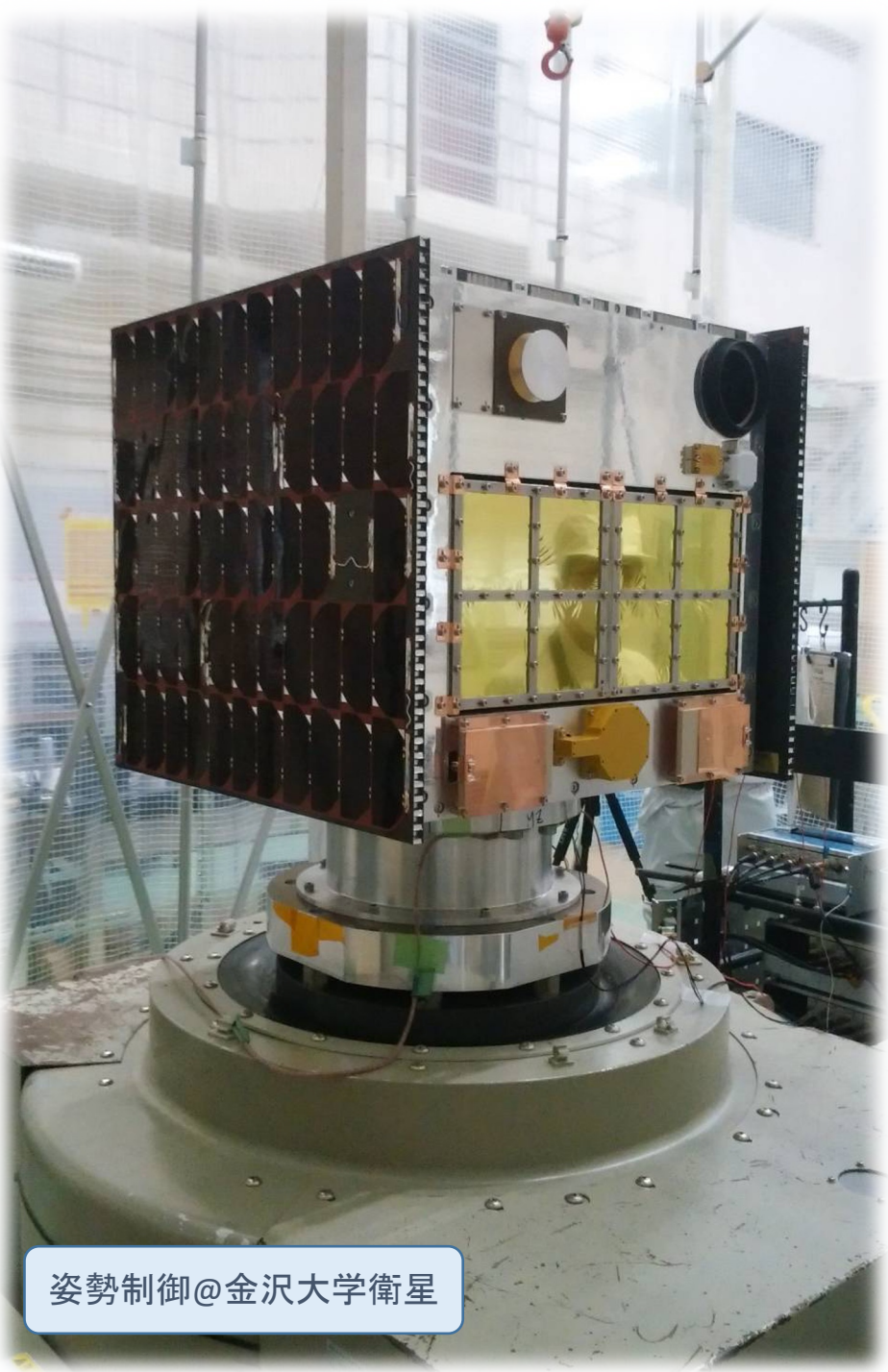
プログラミング指導: 久保守・今井祐希

教材評価: 一方井祐子

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
  - 学習用：　指示に従い実行して下さい
  - 課題用：　課題が難しければ参考にして下さい

分割主鏡制御@せいめい望遠鏡

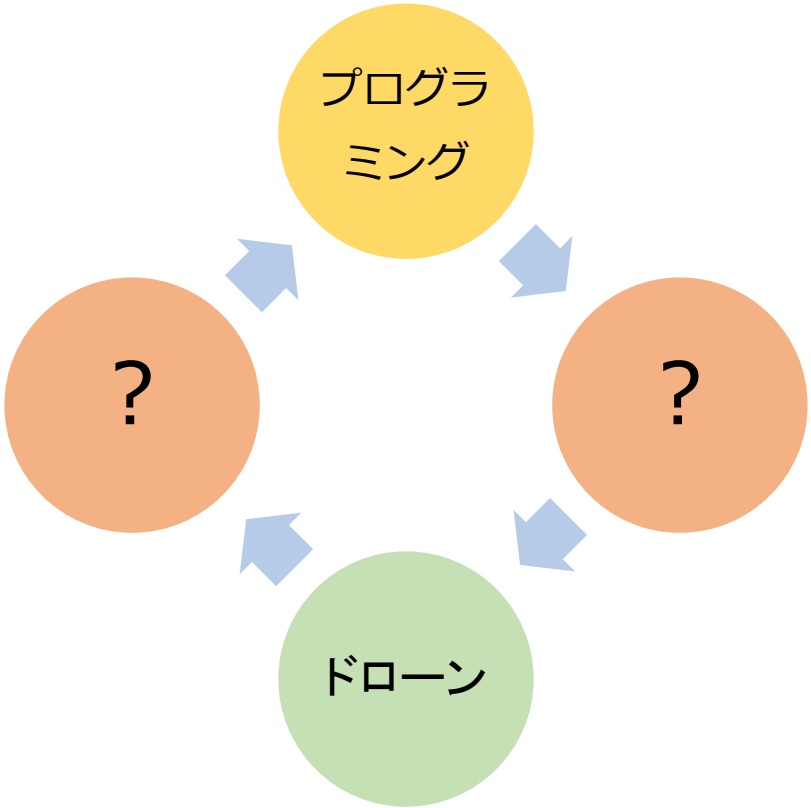


姿勢制御@金沢大学衛星

講義の目的

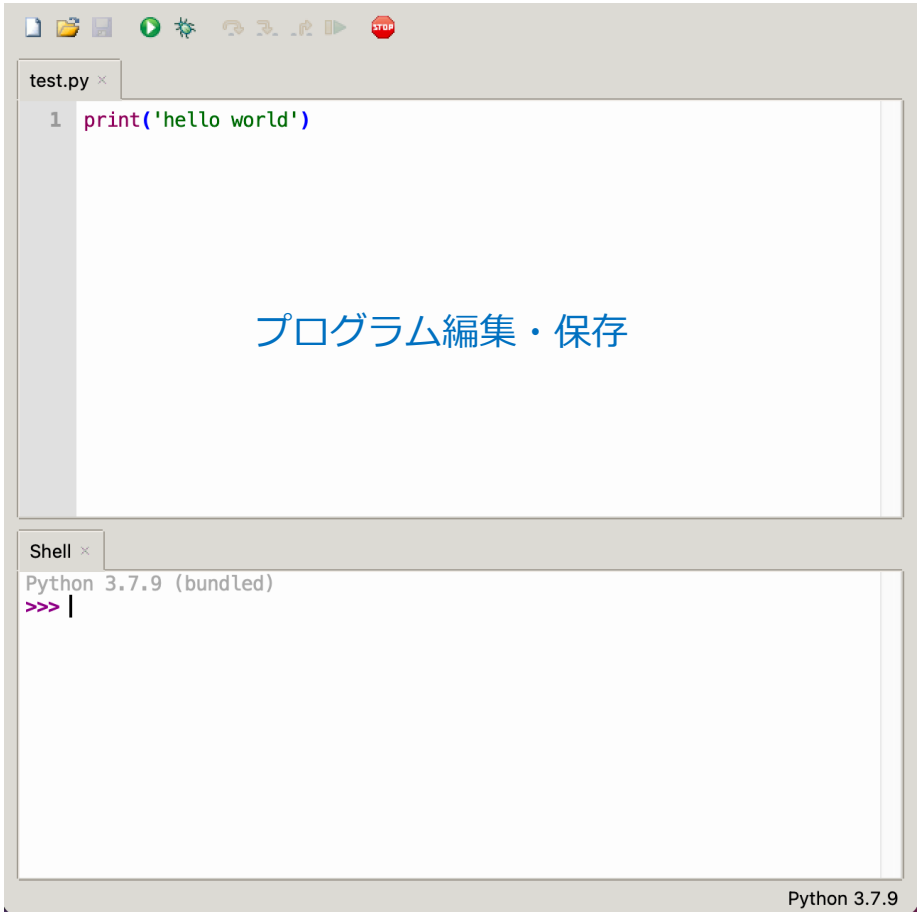
ドローンを使って学びながら遊んでみよう♪

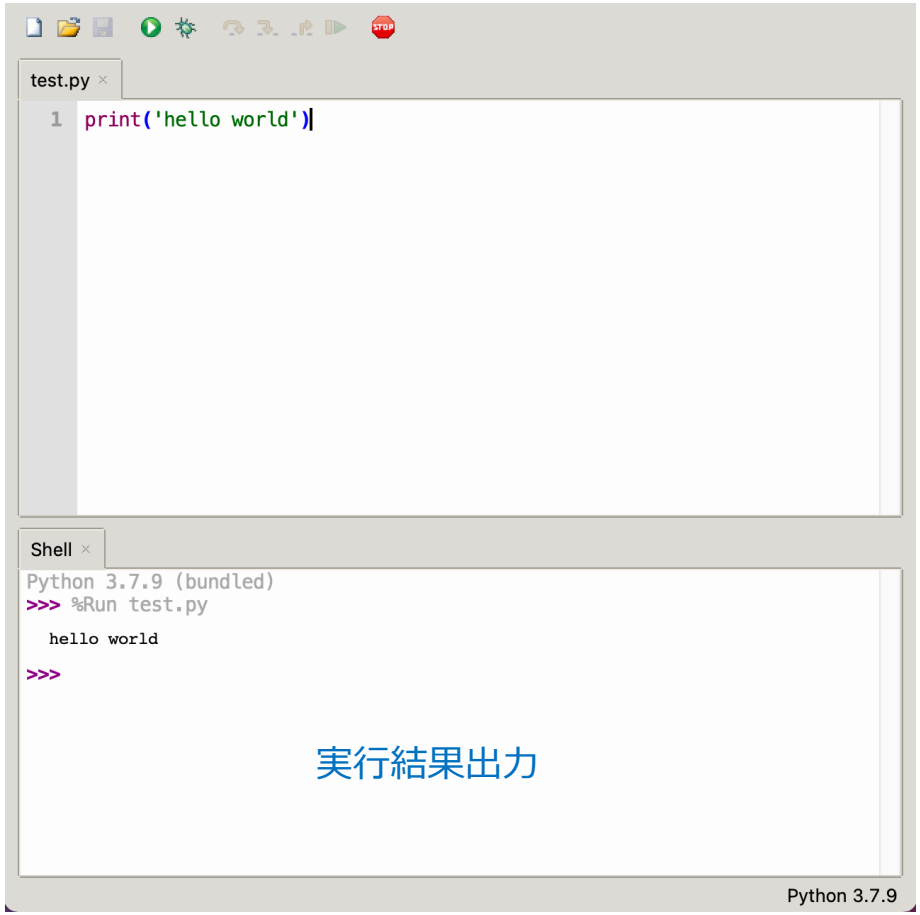
- プログラミング技法の詳細にはとらわれず可能性を感じてください
- 基本機能の組み合わせにより機能拡張できることを体感しよう



Thonny

- 統合開発環境：





緑色実行ボタンを押す

講義概要

3限

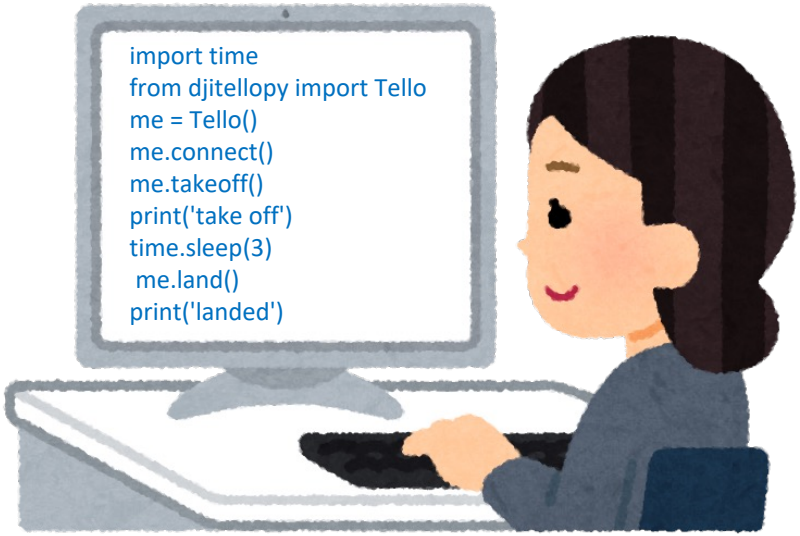
- **ドローン**
- 実習：飛行制御

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

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
  - 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！
  - 「実習用サンプルプログラム」の下の「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

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

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

代表的なモジュールは numpy, sympy, matplotlib. 機械学習なら tensorflow, pytorch など.

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

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

本講義では深入りしませんが，興味をもった方は是非勉強してみてください

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()
      ↑---------- ドット演算子
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）
- **Print文**は動作確認用.

ドット演算子「.」の解読がPythonプログラム理解の鍵

本講義では深入りしませんが，興味をもった方は是非勉強してみてください

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')
```

- 関数get_battery()を呼び出し、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を終了します

シングルクオーテーション ‘ で囲まれた箇所を書き換えて好きな文言を出力してみよう

例えば，print(‘The End’)

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！
  - 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

プログラムを解読してしてみましょう

英単語の意味から類推できるように関数名が選択されています

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()
```

- 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

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

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）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()
```

- 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

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）

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()
```

- 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

```
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
```

順次の考え方に基づき課題に取り組んで下さい

繰り返しを知っている人は試してみて下さい

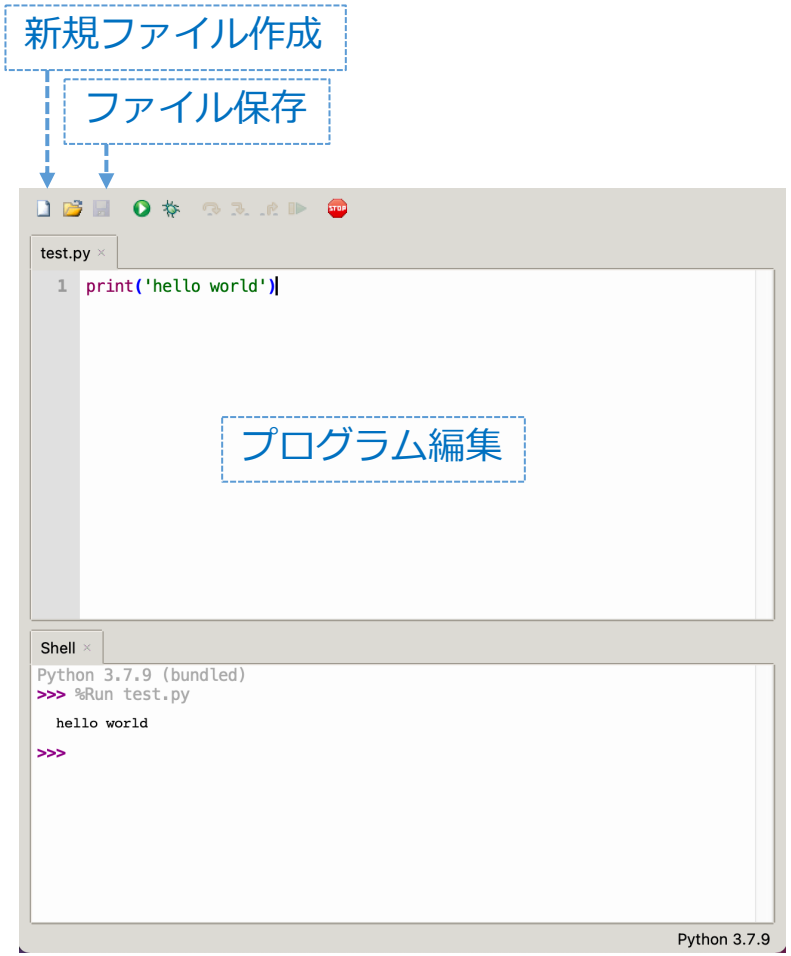
プログラムを簡潔に記載できます

往復運動

課題3-5: tello_motion.py

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
2. 30cm前身
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
10. 着陸



プログラムを一から書いても良いし，サンプルプログラムをコピーしてから追記でもOK

反復横跳び

課題3-6: tello_jump.py

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

課題3-7: tello_triangle.py

telloに一辺50cmの正三角形を描かせてみよう.

自由課題

関数move_*()と関数rotate_*()を組み合わせるとさまざまな飛行を実現できます．flip_*()を用いると宙返りも実現できます．ここで*はワイルドカードを意味します.具体的な関数名は次ページを参照して下さい．自由な発想のもとに飛行プログラムを作成して下さい．既存のプログラムに機能追加するのでもOK．（オススメ：flip_back()を挿入）

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

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

一休み

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

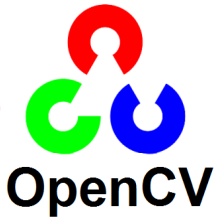
なるだけ顔の全体が明るく映るように光の加減に工夫して撮影しましょう

画像データをバックアップする場合はファイル名 camera_photo.png 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()
```



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

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）

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()
```

- 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

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_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！

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

本当の私♪

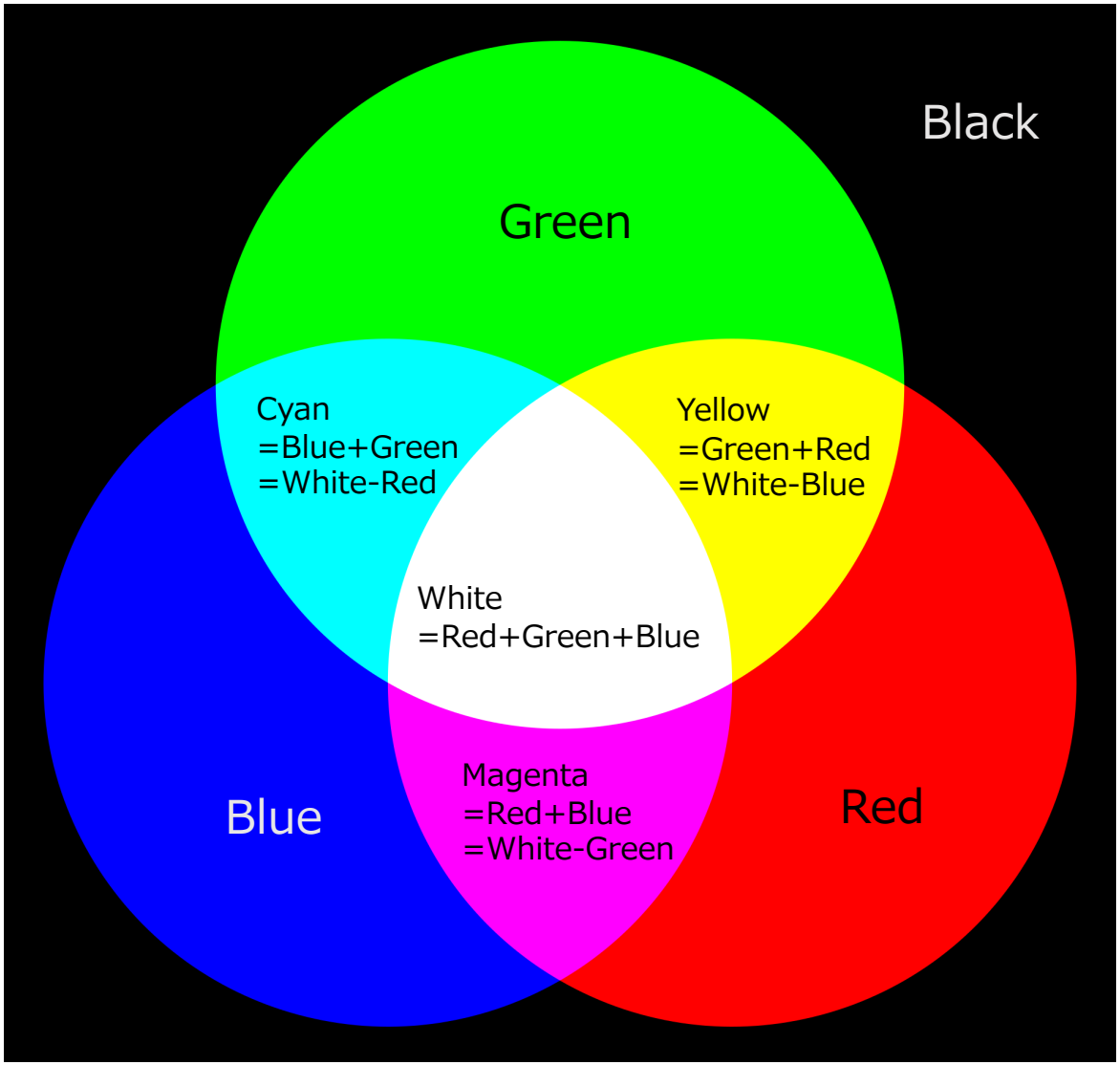
青♪

緑♪

赤♪

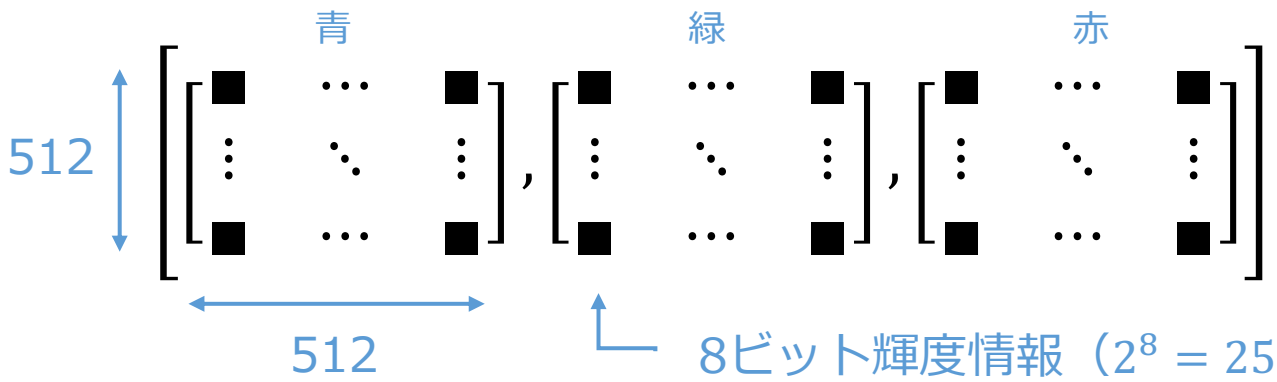
デジタル画像データを直接操作

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

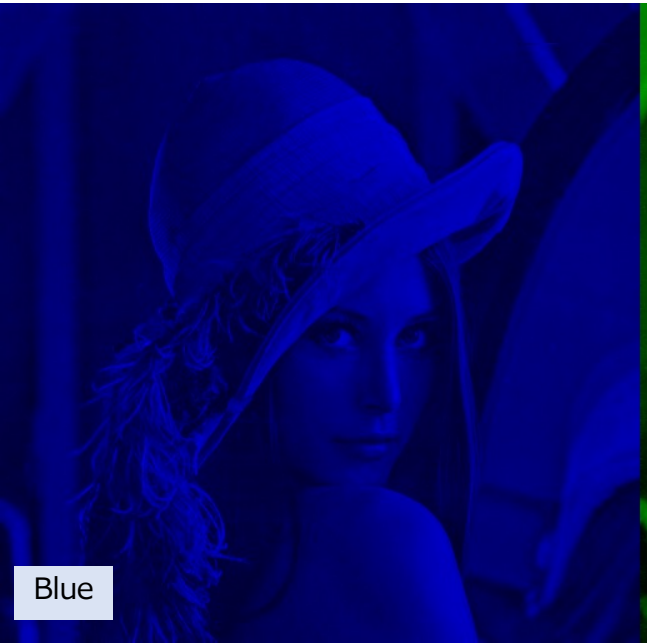
デジタル画像

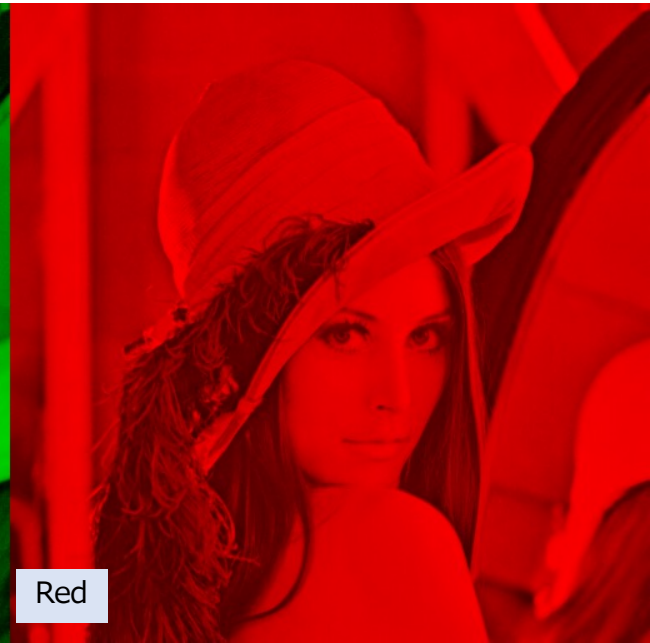
配列：数字を格納する箱



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





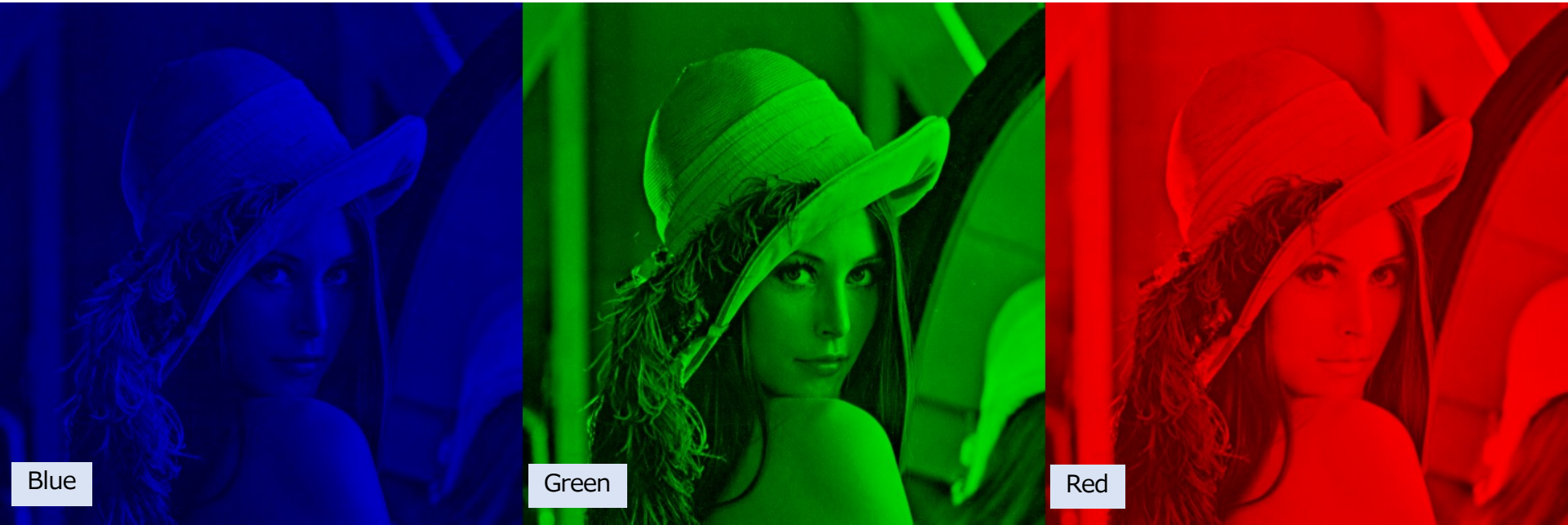
デジタル画像

配列：数字を格納する箱

$$\left[\begin{bmatrix} \blacksquare & \cdots & \blacksquare \\ \vdots & \ddots & \vdots \\ \blacksquare & \cdots & \blacksquare \end{bmatrix}, \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}, \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}\right]$$

青　　緑　　赤

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_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
  - img_B[:, :, 1]が緑画像
  - img_B[:, :, 2]が赤画像

講義概要

3限

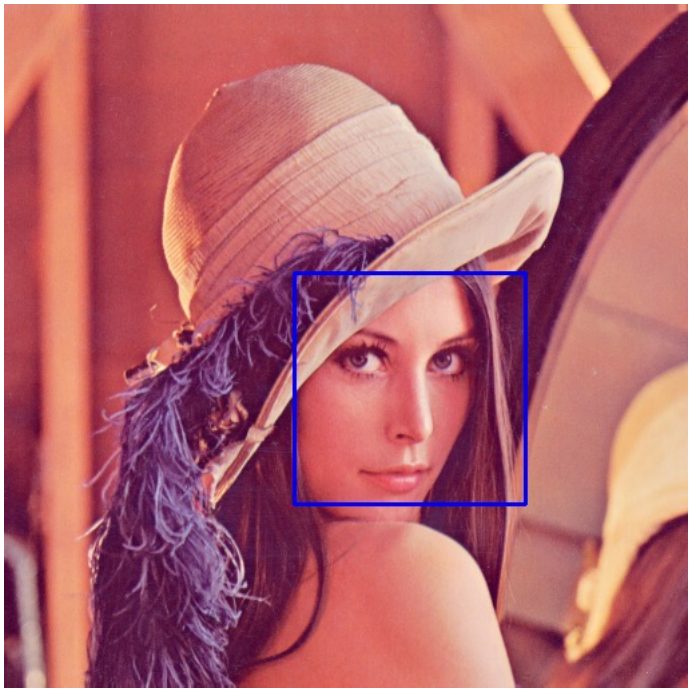
- ドローン
- 実習：飛行制御

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()



人間には簡単だけど…

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 = 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

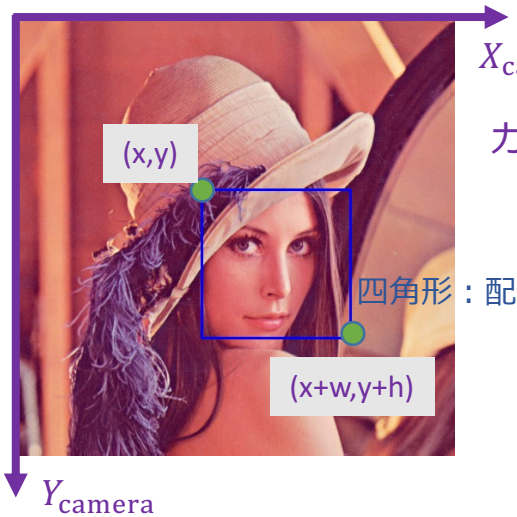
```
import cv2

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

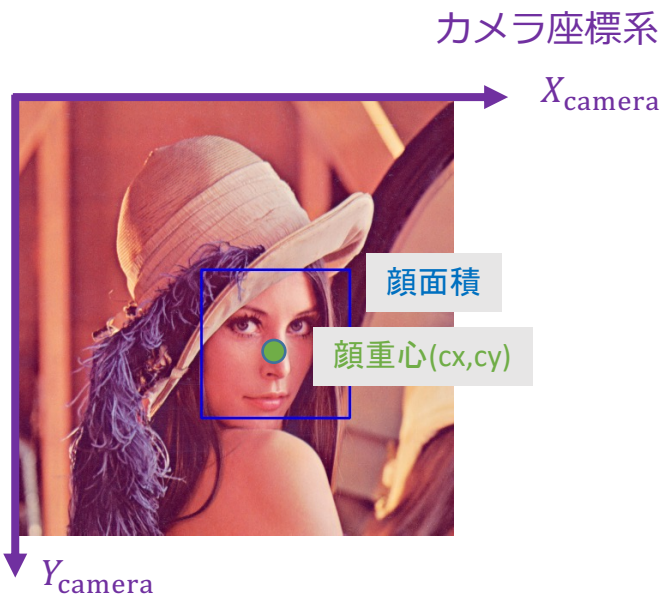


カメラ座標系

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 )
```

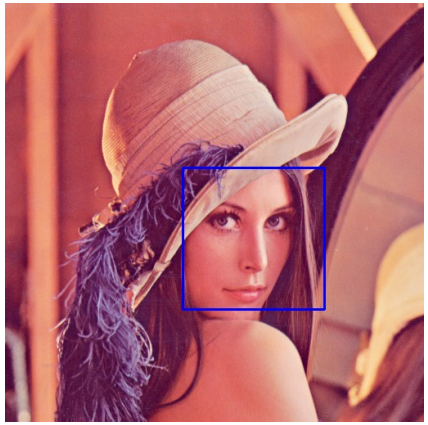
- 顔の中心位置 (cx,cy) を計算
- 顔の面積 area を計算
- 複数の顔が検出された場合に最大の顔を検出
- 顔の中心位置 (cx,cy) と面積 area をまとめて出力
  face_info = [[cx,cy],area]



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

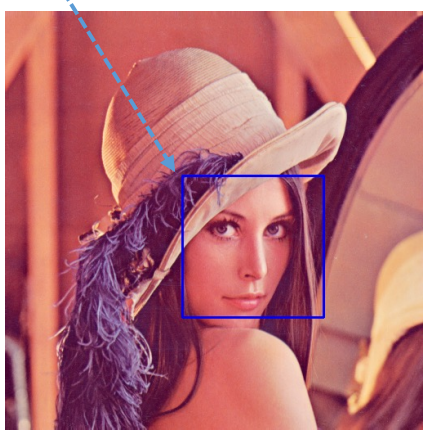
良い感じ♪

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♪

ドローンで自撮り飛行♪

課題4-6: fly_photo.py

ドローンを飛ばして写真撮影を行うプログラムを作成．サンプルプログラムの動作検証でもOK.

ヒント: 学習3-2+学習4-1, 離陸後の高さ調整には move_up(), move_down()

飛ばして顔認識♪

課題4-7: fly_detect.py

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するプログラム作成.

ヒント: 学習4-5+課題4-6, 取得画像を findFace() に渡します

ドローンで鬼ごっこ♪

課題4-9: visual_tracking.py

サンプルプログラムの動作検証.

- me.move_up(90) は身長170cm相当. 身長に合わせて引数を調整して下さい
- ビデオウィンドウ上で「q」を押したら自動着陸

解読：鬼ごっこ飛行

@djitellopy API reference

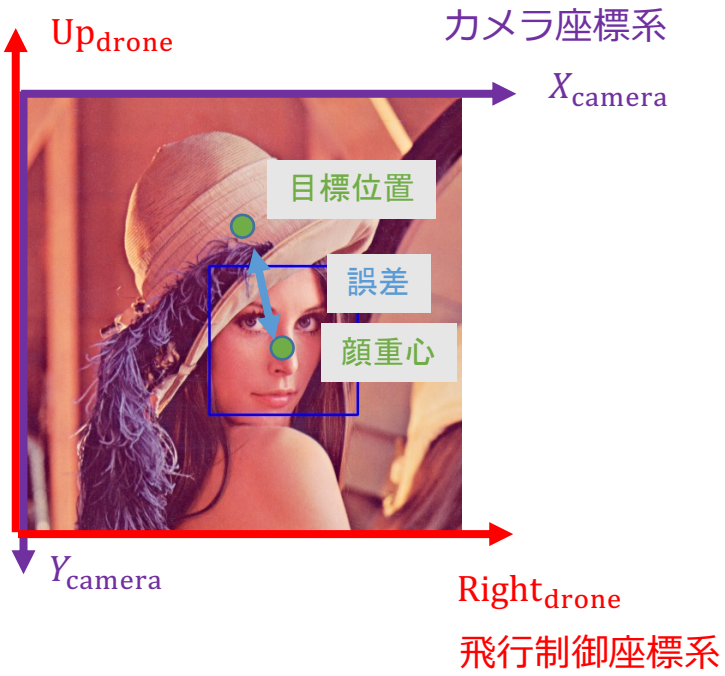
```
send_rc_control(self, left_right_velocity,
forward_backward_velocity, up_down_velocity,
yaw_velocity)
```

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♪**
  - **本講義をきっかけ**として**Pythonの学習 (特にクラスの概念)** に興味をもつ高校生が現れるなら幸いです