

ドローンを題材とする人工知能体験

金沢大学 軸屋一郎

2023年8月31日 (木)3・4限(10時20分~12時30分)

滋賀県立膳所高等学校視聴覚室

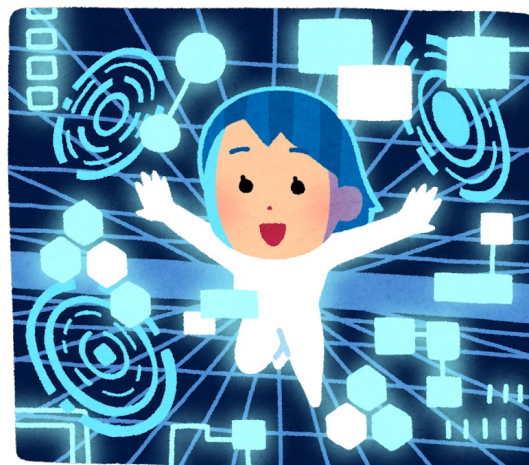
プログラミング指導：久保守

TA：東宗太郎，西藤卓矢，谷口亜門

- プログラミング**言語**は言葉です



喋れれば世界
が広がるよ♪



少しでも工夫しよ
うとするとプログ
ラミングが必須♪

事前学習（オブジェクト指向プログラミング）

- Pythonでは**データ**は**オブジェクト**という抽象的な概念により表現される.
- **オブジェクト**は**値**、**型**、**ID**から構成される.
 - **型**はオブジェクトの設計情報を表す. クラスとかデータ型とも呼ばれる.
 - **ID**はオブジェクトの固有番号を表す.
 - **変数**はオブジェクトの名前を表します.
 - **値**は**メンバ変数**と**メンバ関数**から構成される.
 - **メンバ変数**はオブジェクトに格納されるオブジェクトの名前を表す.
 - **メンバ関数**はオブジェクトに内包された操作を表す.

補足： Pythonではメンバ変数を「属性」、メンバ関数を「メソッド」と呼ぶことが多いのですが、意味が分かりづらくなるために、他のプログラミング言語からメンバ変数とメンバ関数の用語を借用して説明しています

例： [1,2,3] は単なる数字の組み合わせではない
様々なメソッドを内包するリスト型のオブジェクト

```
<untitled> * x
1 x = [1, 2, 3]
2 print(x)
3 x.append(4)
4 print(x)

Shell x
>>> %Run -c $EDITOR_CONTENT
[1, 2, 3]
[1, 2, 3, 4]
```

クラスを意識すると解釈しやすい
<https://docs.python.org/ja/3/library/stdtypes.html>
<https://docs.python.org/ja/3/tutorial/datastructures.html>

リスト x に値 4 を追加

Pythonではデータは
オブジェクト



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

OK!

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
分けられると効率良いかも



講義概要

前半

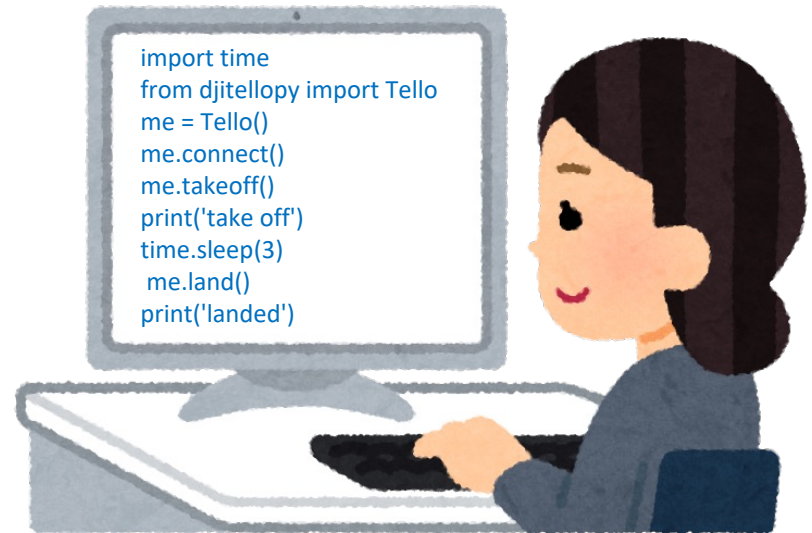
- ドローン
- 学習：飛行制御
- 課題：飛行制御

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**をお願いします

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

バッテリー残量確認

1_1_battery.py

```
from djitellopy import Tello

me = Tello()

me.connect()

print('connected')

print(me.get_battery())

me.end()

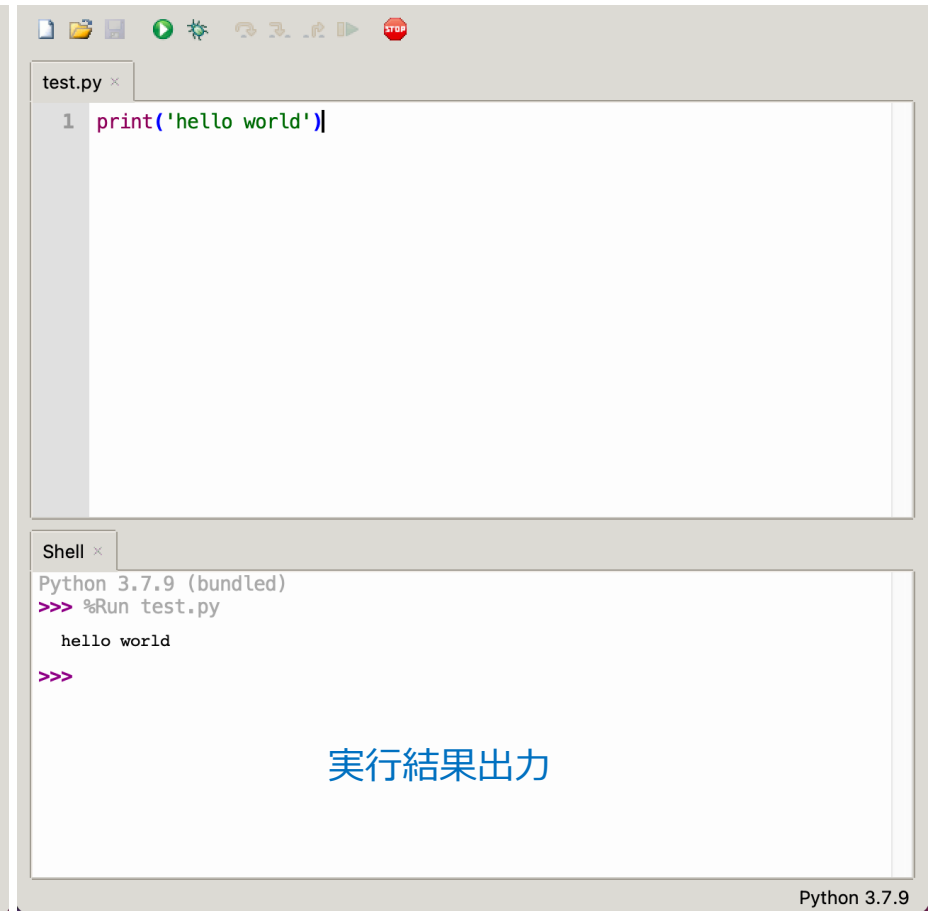
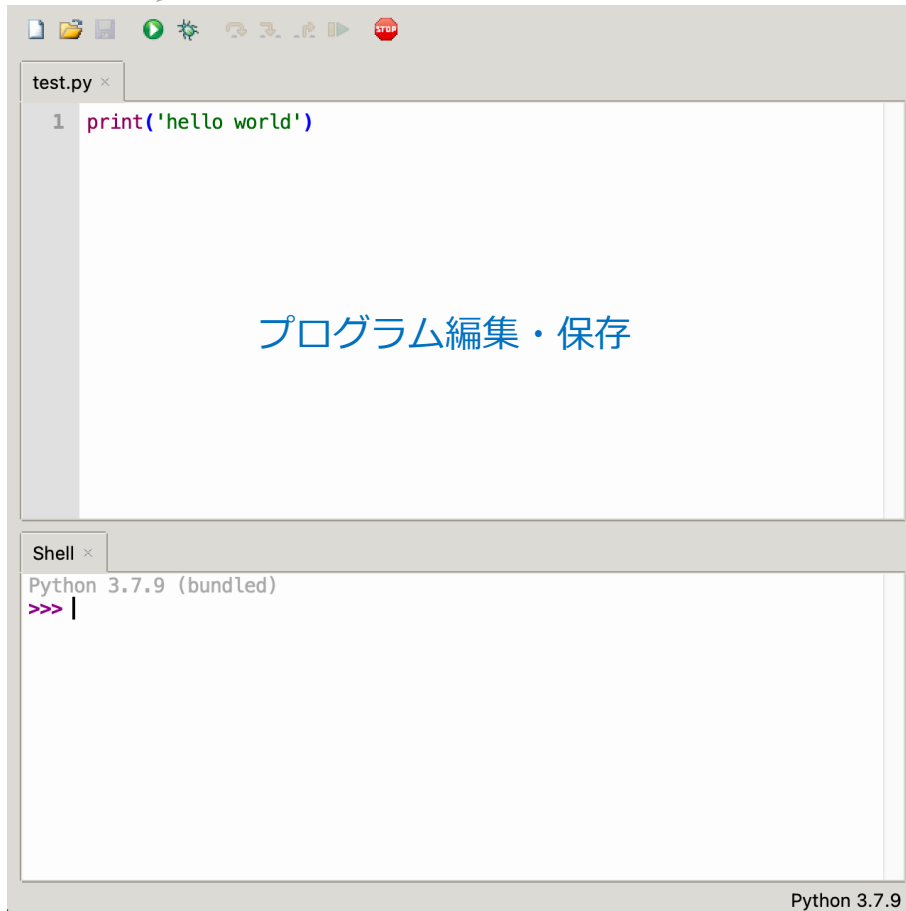
print('terminated')
```

- 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

Thonny

- Python統合開発環境：

実行



緑色実行ボタンを押す

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. 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

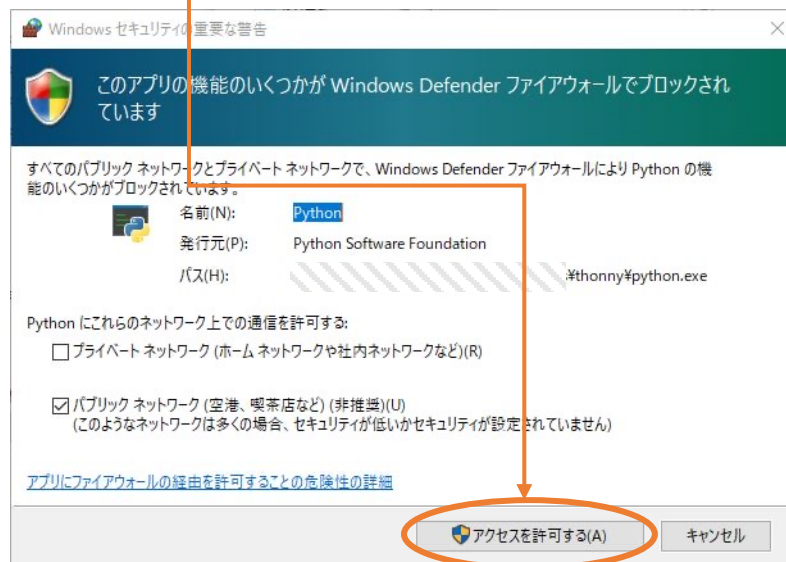


4. PCのwifi接続先から**配布機体のSSID**を選択

- SSIDはバッテリー下に記載 (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

5. 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



6. Telloの電源オフ

- 飛行可能時間：約10分（適宜バッテリー交換）
- **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）



バッテリー残量確認（解説 1）

```
from djitellopy import Tello
```

```
me = Tello()
```

```
me.connect()
```

```
print('connected')
```

```
print(me.get_battery())
```

```
me.end()
```

```
print('terminated')
```

- パッケージdjitellopyからクラスTelloを読み込み

雛形の束

雛形

聞いたことない…
意味不明…

詳細に捉われず
雰囲気だけでOK！

OK!

パッケージを追加することによりPythonの機能を拡張できます

バッテリー残量確認（解説 1）

```
from djitellopy import Tello
```

```
me = Tello()
```

```
me.connect()
```

```
print('connected')
```

```
print(me.get_battery())
```

```
me.end()
```

```
print('terminated')
```

- パッケージdjitellopyからクラスTelloを読み込み

雛形の束

雛形

事前学習との
関係は？



GitHub を
確認しましょう



パッケージを追加することによりPythonの機能を拡張できます

DJITellopy@Github

カメラ画像取り込みの技術的理由により
DJITellopyのバージョンを 2.4.0 にダウングレード

クラスの読み込み
from djitellopy import Tello
に対応。詳しくは __init__.py を参照

The screenshot shows the DJITelloPy repository on GitHub. The left sidebar displays the file structure, with the `djitellopy` folder selected. Inside this folder, the `tello.py` file is highlighted. A green box highlights the version `2.4.0` in the file selector. A pink box highlights the `tello.py` file in the file list, with a callout labeled 'モジュール' (Module). Another pink box highlights the `__init__.py` file in the file list, with a callout labeled 'パッケージ' (Package). The main area shows the code for `tello.py`, starting with `@enforce_types` and `class Tello:`. A pink box highlights the `class Tello:` line, with a callout explaining that it corresponds to the `__init__.py` file. Another pink box highlights the class variables (e.g., `RESPONSE_TIMEOUT`, `TAKEOFF_TIMEOUT`, etc.) within the `Tello` class, with a callout labeled 'クラス変数を規定' (Specify class variables). The code continues with various constants and settings for the Tello drone.

1019行までクラス Tello が延々と定義されます

<https://github.com/damiafuentes/DJITelloPy/blob/1c709cc0937e0e5ccacc73315f8487e3393a31cc/djitellopy/tello.py#L21>

バッテリー残量確認（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

雛形

実体化



クラスが分かれば
何でもわかる！

設計書から車を作ったら走れる



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

DJITelloPy@Github

The screenshot shows the DJITelloPy repository on GitHub. The left sidebar displays the file structure, with `tello.py` selected. The main area shows the code for the `Tello` class. Annotations in pink boxes highlight key parts of the constructor:

- インスタンスの生成**
`me = Tello()`
に対応。コンストラクタを規定
- インスタンス変数を規定**

The code in `tello.py` is as follows:

```
21 class Tello:
93     def __init__(self,
94                 host=TELLO_IP,
95                 retry_count=RETRY_COUNT):
96
97         global threads_initialized, client_socket, drones
98
99         self.address = (host, Tello.CONTROL_UDP_PORT)
100        self.stream_on = False
101        self.retry_count = retry_count
102        self.last_received_command_timestamp = time.time()
103        self.last_rc_control_timestamp = time.time()
104
105        if not threads_initialized:
106            # Run Tello command responses UDP receiver on background
107            client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
108            client_socket.bind(('', Tello.CONTROL_UDP_PORT))
109            response_receiver_thread = Thread(target=Tello.udp_response_receiver)
110            response_receiver_thread.daemon = True
111            response_receiver_thread.start()
112
113            # Run state UDP receiver on background
114            state_receiver_thread = Thread(target=Tello.udp_state_receiver)
115            state_receiver_thread.daemon = True
116            state_receiver_thread.start()
117
118            threads_initialized = True
119
120        drones[host] = {'responses': [], 'state': {}}
121
122        self.LOGGER.info("Tello instance was initialized. Host: '{}'. Port: '{}'.format(host, Tello.CONTROL_UDP_PORT)
123
```

A large orange arrow points from the `__init__` method down to line 118, indicating that the constructor is defined up to this point.

118行目までコンストラクタを記載

<https://github.com/damiafuentes/DJITelloPy/blob/1c709cc0937e0e5ccacc73315f8487e3393a31cc/djitellopy/tello.py#L93-L95>

バッテリー残量確認（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.



DJITelloPy@Github

Files

2.4.0

Go to file

- ▼ djitellopy
 - __init__.py
 - enforce_types.py
 - swarm.py
 - tello.py
- ▼ docs
- ▼ examples
- .gitignore
- LICENSE.txt
- README.md
- doc-requirements.txt
- mkdocs.yml
- requirements.txt
- setup.cfg
- setup.py

Documentation · Share feedback

DJITelloPy / djitellopy / tello.py

Code Blame 1074 lines (902 loc) · 35.2 KB

me 55% faster with GitHub Copilot

Raw Copy Download Edit View

```
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560

def connect(self, wait_for_state=True):
    """Enter SDK mode. Call this before any of the control functions.
    """
    self.send_control_command("command")

    if wait_for_state:
        REPS = 20
        for i in range(REPS):
            if self.get_current_state():
                t = i / REPS # in seconds
                Tello.LOGGER.debug('".connect()" received first state packet after {} seconds".format(t))
                break
            time.sleep(1 / REPS)

        if not self.get_current_state():
            raise Exception('Did not receive a state packet from the Tello')

def send_keepalive(self):
    """Send a keepalive packet to prevent the drone from landing after 15s
    """
    self.send_control_command("keepalive")

def turn_motor_on(self):
    """Turn on motors without flying (mainly for cooling)
    """
    self.send_control_command("motoron")

def turn_motor_off(self):
    """Turns off the motor cooling mode
    """
```

↑ Top

me.connect()
に対応。インスタンスメソッドを呼び出す

} command というコマンドをtelloに送信

<https://github.com/damiafuentes/DJITelloPy/blob/1c709cc0937e0e5ccacc73315f8487e3393a31cc/djitellopy/tello.py#L531-L546>

バッテリー残量確認（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ー残量を確認できます

DJITelloPy@Github

AdGuard for Safari

2.4.0

Go to file

- ▼ djitellopy
 - __init__.py
 - enforce_types.py
 - swarm.py
 - tello.py
- > docs
- > examples
- .gitignore
- LICENSE.txt
- README.md
- doc-requirements.txt
- mkdocs.yml
- requirements.txt
- setup.cfg
- setup.py

Documentation • Share feedback

DJITelloPy / djitellopy / tello.py

Code Blame 1074 lines (902 loc) · 35.2 KB Master with GitHub Copilot

Raw Copy Download Edit View

```
381         return self.get_state_field('bat')
382
383     def get_battery(self) -> int:
384         """Get current battery percentage
385         Returns:
386             int: 0-100
387         """
388         return self.get_state_field('bat')
389
390     def get_udp_video_address(self) -> str:
391         """Internal method, you normally wouldn't call this yourself.
392         """
393         address_schema = 'udp://{ip}:{port}' # + '?overrun_nonfatal=1&fifo_size=5000'
394         address = address_schema.format(ip=self.VS_UDP_IP, port=self.VS_UDP_PORT)
395         return address
396
397     def get_video_capture(self):
398         """Get the VideoCapture object from the camera drone.
399         Users usually want to use get_frame_read instead.
400         Returns:
401             VideoCapture
402         """
403
404         if self.cap is None:
405             self.cap = cv2.VideoCapture(self.get_udp_video_address())
406
407         if not self.cap.isOpened():
408             self.cap.open(self.get_udp_video_address())
409
410         return self.cap
411
412     def get_frame_read(self) -> 'BackgroundFrameRead':
```

me.get_battery()
に対応。インスタンスメソッドを呼び出す

戻り値を規定

<https://github.com/damiafuentes/DJITelloPy/blob/1c709cc0937e0e5ccacc73315f8487e3393a31cc/djitellopy/tello.py#L390-L395>

バッテリー残量確認（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



DJITelloPy@Github

The screenshot shows the DJITelloPy repository on GitHub. The left sidebar displays the file structure, with the `tello.py` file selected. The main area shows the code for `tello.py`, specifically the `end` and `__del__` methods. Two pink callout boxes highlight specific code: one for `me.end()` and its corresponding `end` method, and another for the `__del__` method, which is annotated as a destructor.

Files

- 2.4.0
- Go to file
- djitellopy
 - __init__.py
 - enforce_types.py
 - swarm.py
 - tello.py
- docs
- examples
- .gitignore
- LICENSE.txt
- README.md
- doc-requirements.txt
- mkdocs.yml
- requirements.txt
- setup.cfg
- setup.py

Documentation · Share feedback

DJITelloPy / djitellopy / tello.py

Code Blame 1074 lines (902 loc) · 35.2 KB Code 55% faster with GitHub Copilot

Raw Copy Download Edit View

```
1001
1002 def end(self):
1003     """Call this method when you want to end the tello object
1004     """
1005     if self.is_flying:
1006         self.land()
1007     if self.stream_on:
1008         self.streamoff()
1009     if self.background_frame_read is not None:
1010         self.background_frame_read.stop()
1011     if self.cap is not None:
1012         self.cap.release()
1013
1014     host = self.address[0]
1015     if host in drones:
1016         del drones[host]
1017
1018 def __del__(self):
1019     self.end()
1020
1021
1022 class BackgroundFrameRead:
1023     """
1024     This class read frames from a VideoCapture in background. Use
1025     backgroundFrameRead.frame to get the current frame.
1026     """
1027
1028 def __init__(self, tello, address):
1029     tello.cap = cv2.VideoCapture(address)
1030
1031     self.cap = tello.cap
```

me.end()
に対応。終了処理を規定

デストラクタ

<https://github.com/damiafuentes/DJITelloPy/blob/1c709cc0937e0e5ccacc73315f8487e3393a31cc/djitellopy/tello.py#L1002-L1019>

```
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')
```

- takeoff() 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
```

- 実行してみましょう！

```
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')
```

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



反時計回り:
`rotate_counter_clockwise()`

時計回り:
`rotate_clockwise()`

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

わからなくて当然
気軽に質問してみよう



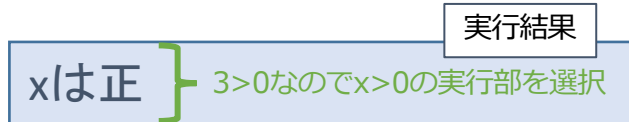
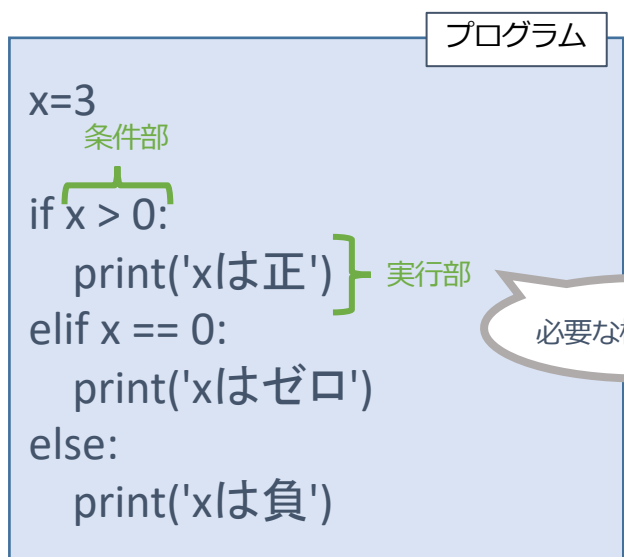
構造化定理

- 順次

- プログラムは上から下に順番に実行

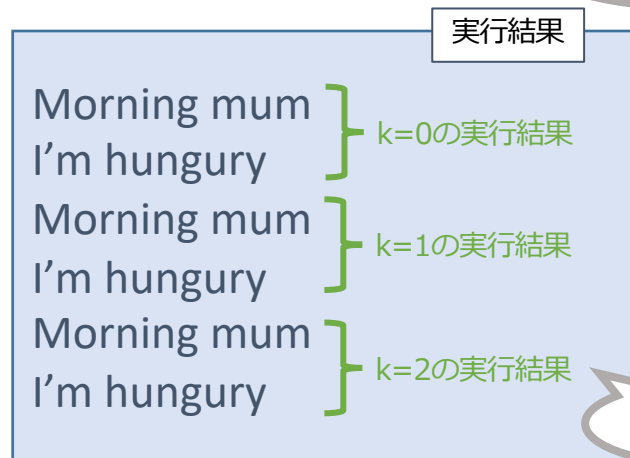
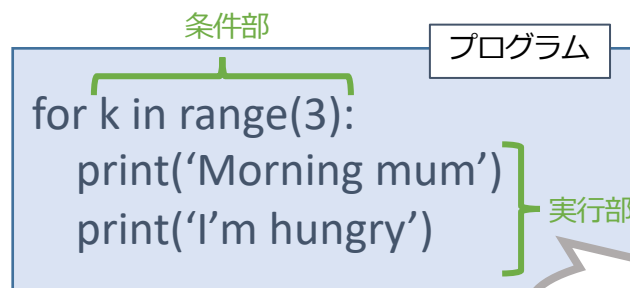
- 分岐

- If文により処理を分岐できる



- 反復

- For文により処理を繰り返せる



関数一覧

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()
~~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()

宙
返
り

機体へダメージ

平
行
移
動

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()
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()
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()

回
転

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

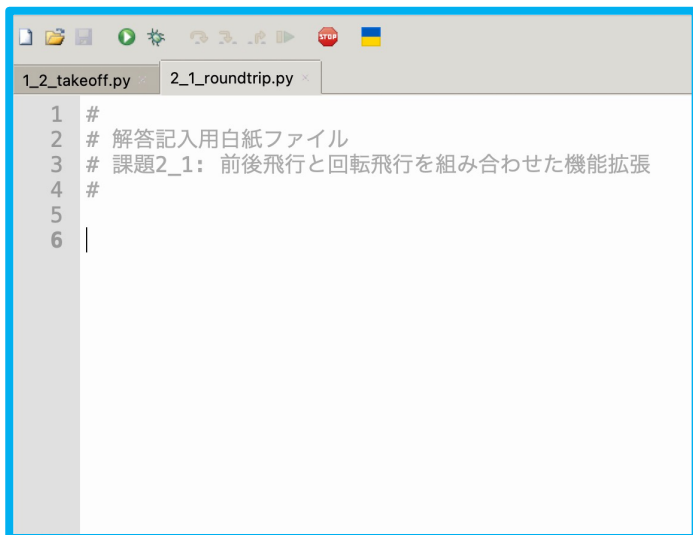
} 1_3_move.pyと1_4_rotate.pyを参考に新規作成

10. 着陸

} 1_2_takeoff.pyと共通

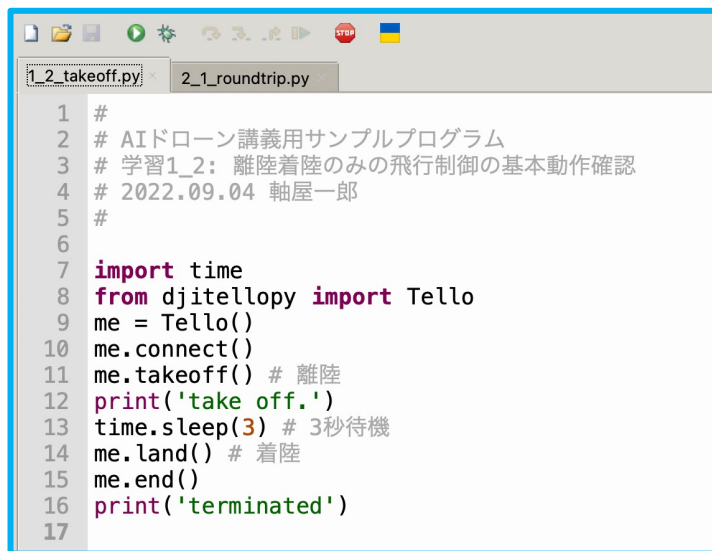
新規プログラム作成の一例

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



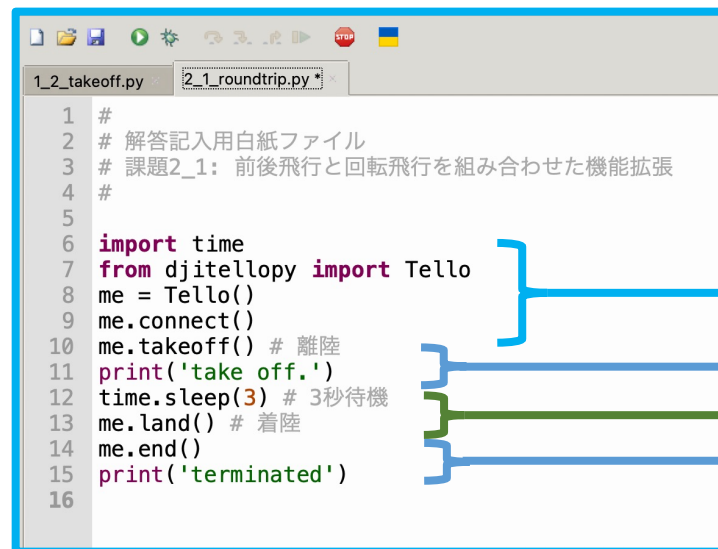
```
1 #
2 # 解答記入用白紙ファイル
3 # 課題2_1: 前後飛行と回転飛行を組み合わせた機能拡張
4 #
5
6 |
```

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してからコピー(Ctrl+C)



```
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
```

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)



```
1 #
2 # 解答記入用白紙ファイル
3 # 課題2_1: 前後飛行と回転飛行を組み合わせた機能拡張
4 #
5
6 import time
7 from djitellopy import Tello
8 me = Tello()
9 me.connect()
10 me.takeoff() # 離陸
11 print('take off.')
12 time.sleep(3) # 3秒待機
13 me.land() # 着陸
14 me.end()
15 print('terminated')
16
```

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。
- シングルクォ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。

モジュール読み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.

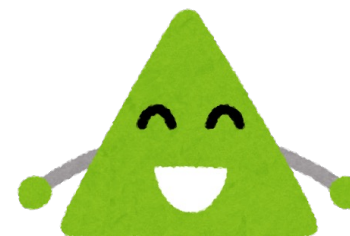
右飛行: move_right(), 左飛行: move_left()



正三角形飛行

2_3_triangle.py (新規作成)

telloに一边50cmの正三角形を描かせてみよう.



バッテリー確認後飛行

2_4_batchcheck.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

一休み



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

```
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！

プログラムと同じディレクトリ
に画像ファイルが保存されます

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

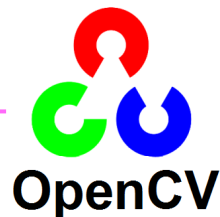
カメラ画像撮影（解説）

import cv2

from djitellopy import Tello

me = Tello()

me.connect()



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

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()

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()
```

```
def get_frame_read(self) -> 'BackgroundFrameRead':  
    """Get the BackgroundFrameRead object from the camera drone. Then, you just need to call  
    backgroundFrameRead.frame to get the actual frame received by the drone.  
    Returns:  
        BackgroundFrameRead  
    """
```

- 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

```
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

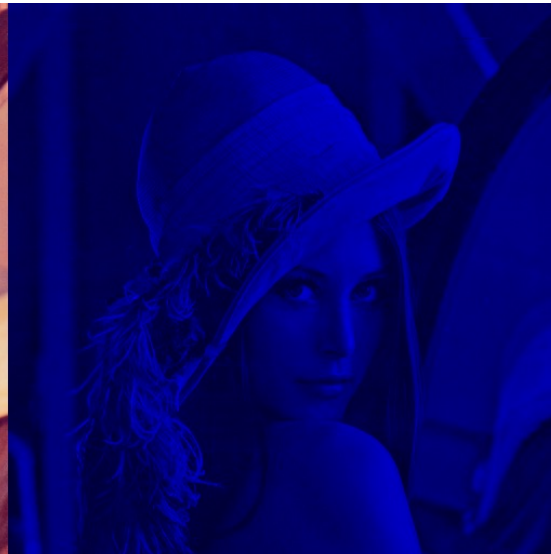
画像処理

※ Lena (標準テスト画像)

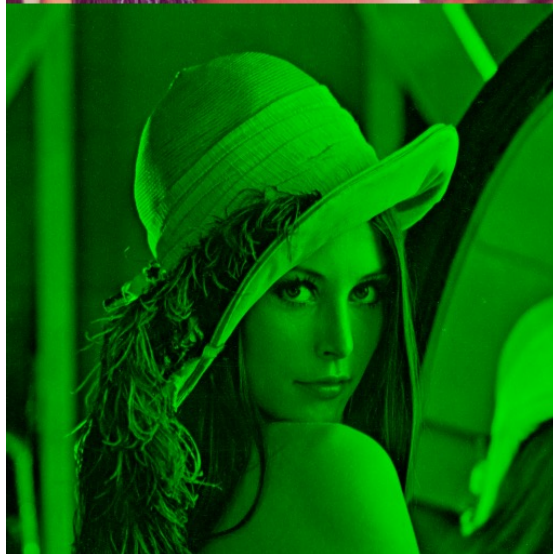
本当の私 ♪



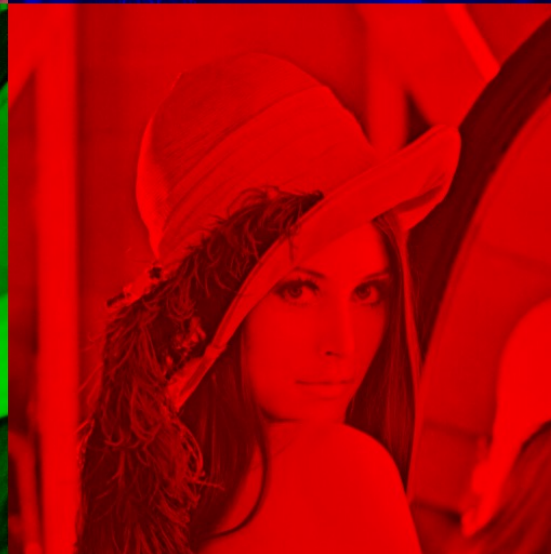
青 ♪



緑 ♪

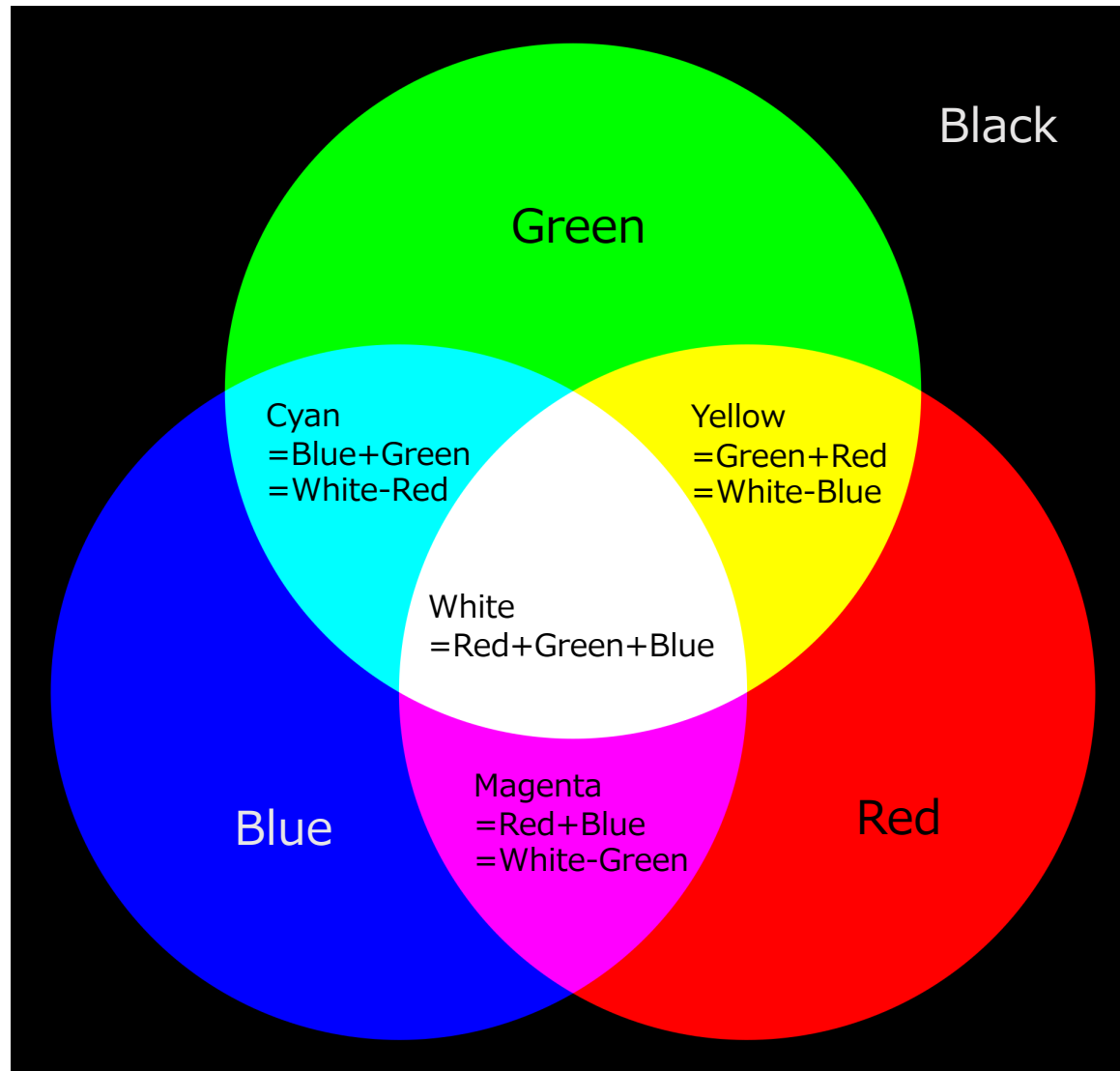


赤 ♪



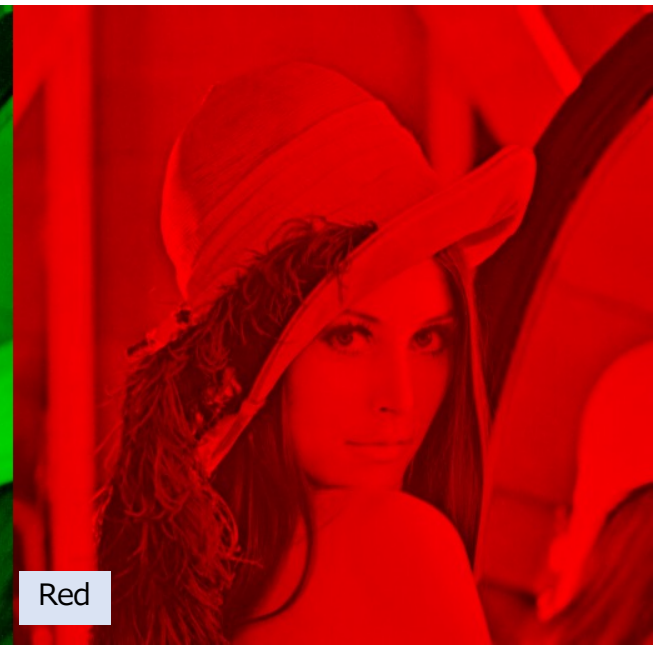
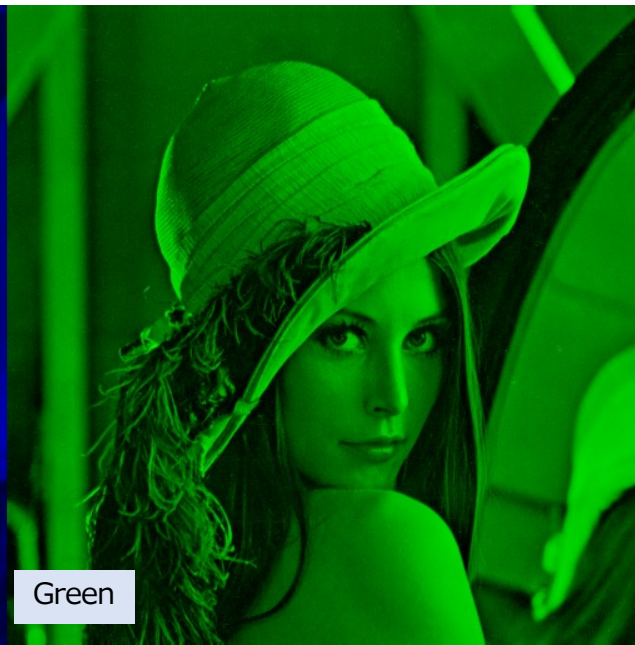
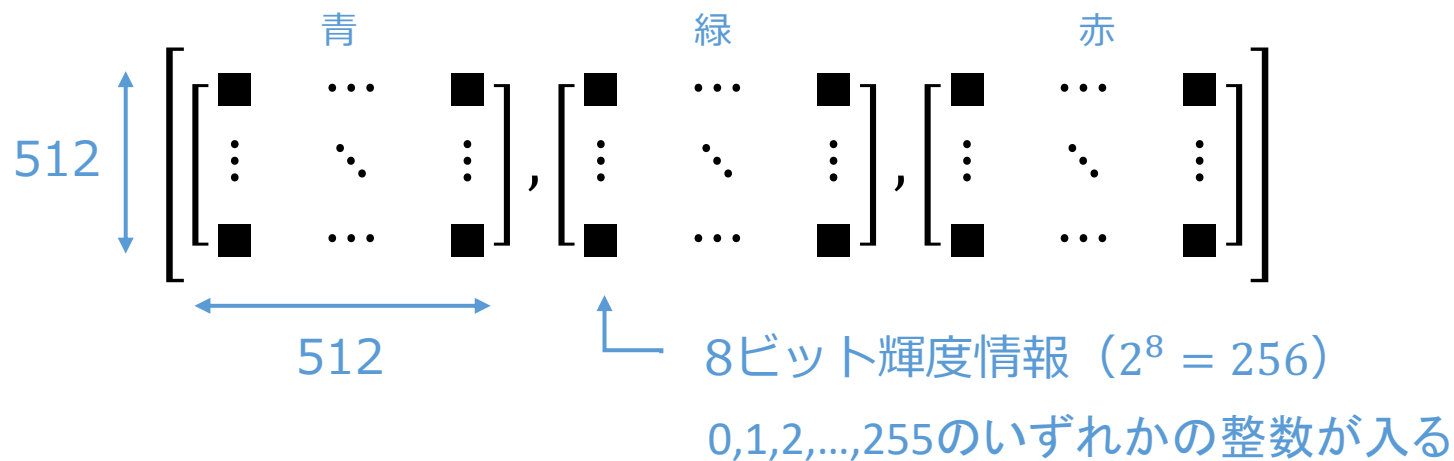
デジタル画像データを直接操作

光の三原色



デジタル画像

配列: 数字を格納する箱



デジタル画像

配列: 数字を格納する箱

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



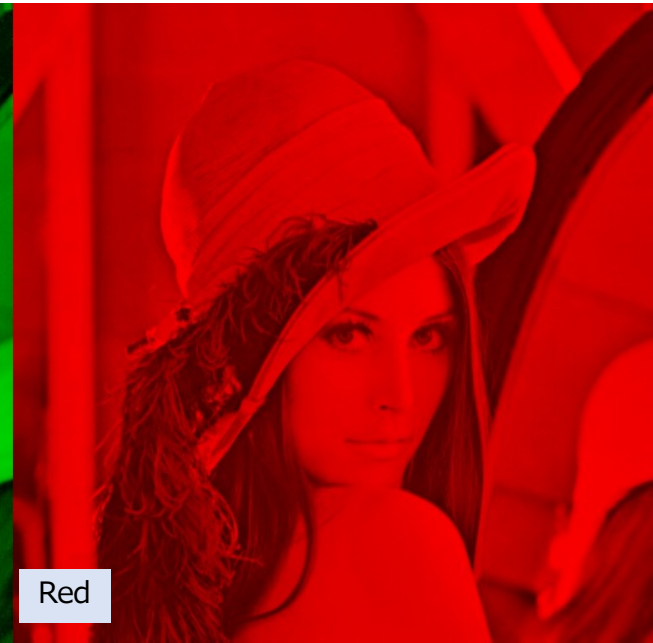
青画像 = 緑・赤の輝度情報をゼロに書換え



Blue



Green



Red

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
```

- `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

```
face_src = cv2.imread('camera_photo.png') ← カメラ撮影画像を読み込む
```

```
face_src_OBGR = colorChange0(face_src) ← 関数呼び出し：  
画像を関数に渡し加工画像を受け取る
```

```
cv2.imwrite('camera_photo_0_OBGR.png', face_src_OBGR) ← 加工画像を保存
```



全部理解は大変…
ブロック単位で
軽く眺めよう

講義概要

前半

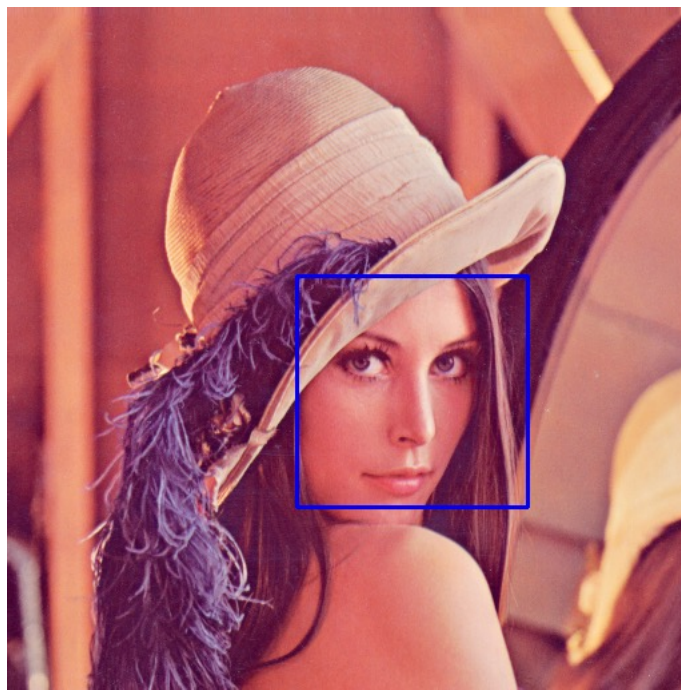
- ドローン
- 学習：飛行制御
- 課題：飛行制御

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

```
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)
```

```
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)
```

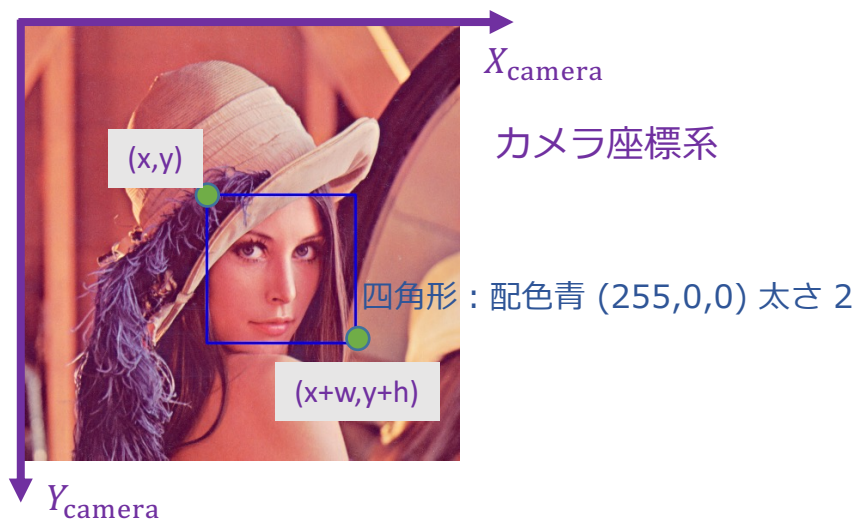
```
cv2.imwrite('face_detect.png', face_src_rev )
```

- 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

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



```
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')

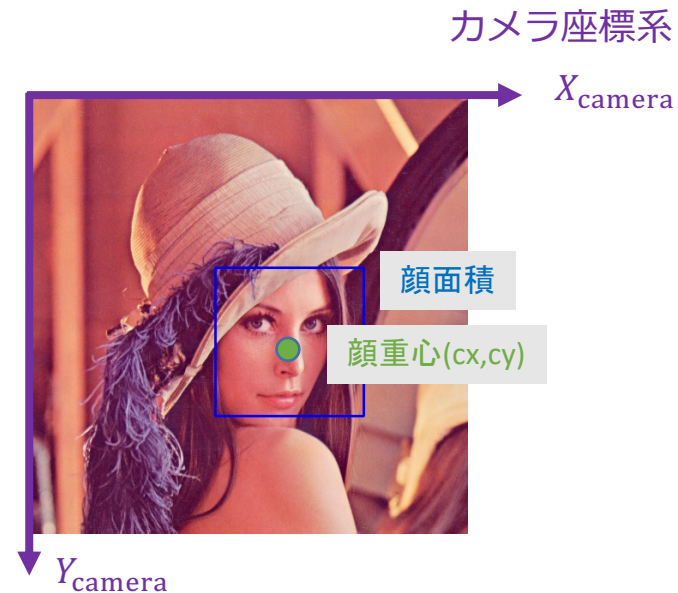
face_src_rev, face_info = findBiggestFace(face_src)

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]`

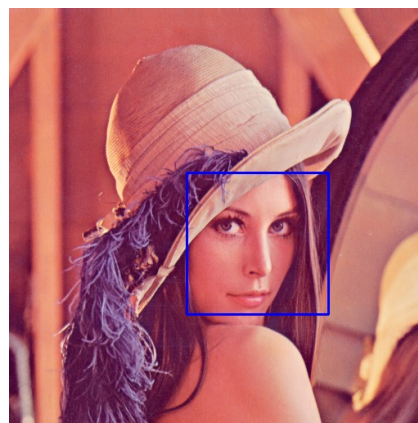


ソフトウェアセンサ： 計測データを追加取得

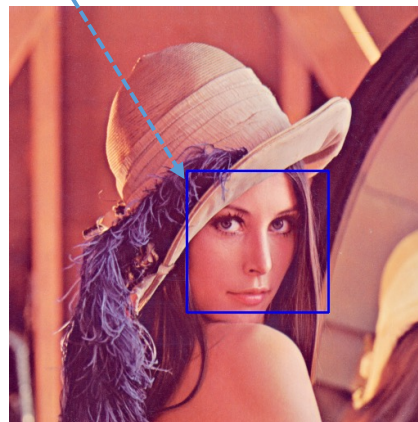
飛行制御 + 顔認識



ちょっ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



何か近づいてきた？

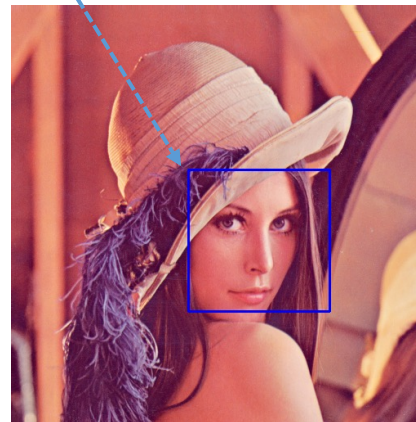
飛行制御 + 顔認識



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と4_4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 を組み合わせる.
- 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 を組み合わせる.
- 取得画像を `findFace()` に渡す



ドローンで鬼ごっこ♪

4_3_tracking_opencv.py (動作検証)

4_4_tracking_mediapipe.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

4_1_fly_photo.py (新規作成)

考え方:

4_1_fly_photo.pyに

- 1) モジュール読み込み
- 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。

```
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を読み込む
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

機能を組み合わせる（飛ばして顔認識）

```
3_3_detect.py x 4_1_fly_photo.py x
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 } モジュールをプログラム冒頭に記載
8 face_detect_classifier = 'haarcascade_frontalface_default.xml' # Haar
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保存
21
```

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。

解説：鬼ごっこ飛行

4_3_tracking_opencv.py (動作検証)

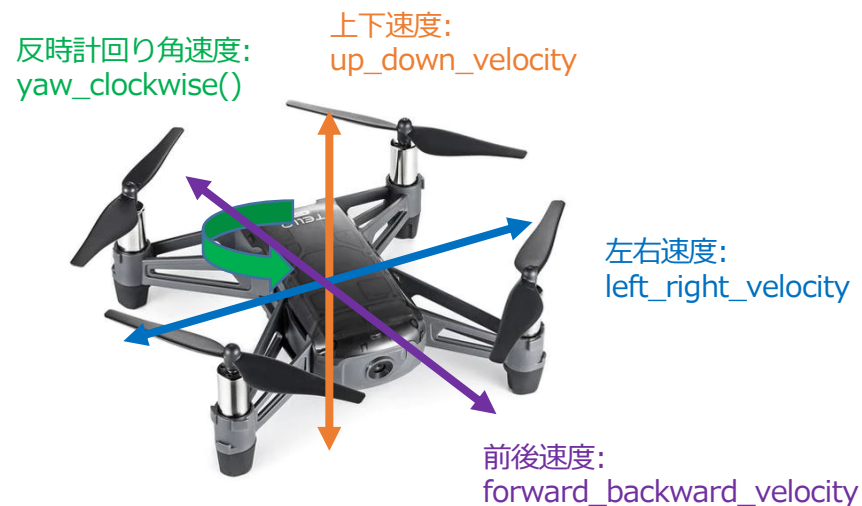
@djitellopy API reference

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

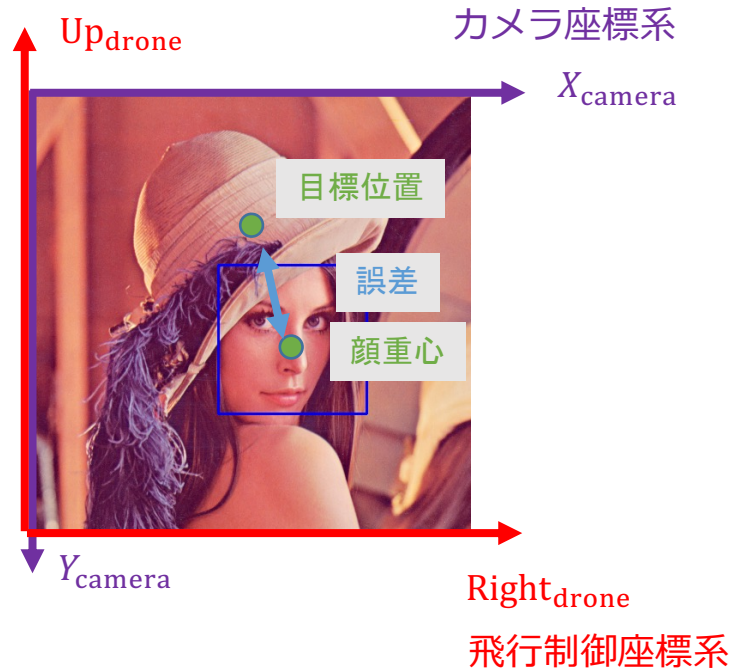
Send RC control via four channels. Command is sent every
self.TIME_BTW_RC_CONTROL_COMMANDS seconds.

Parameters:

Name	Type	Description	Default
<code>left_right_velocity</code>	<code>int</code>	-100~100 (left/right)	<i>required</i>
<code>forward_backward_velocity</code>	<code>int</code>	-100~100 (forward/backward)	<i>required</i>
<code>up_down_velocity</code>	<code>int</code>	-100~100 (up/down)	<i>required</i>
<code>yaw_velocity</code>	<code>int</code>	-100~100 (yaw)	<i>required</i>



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

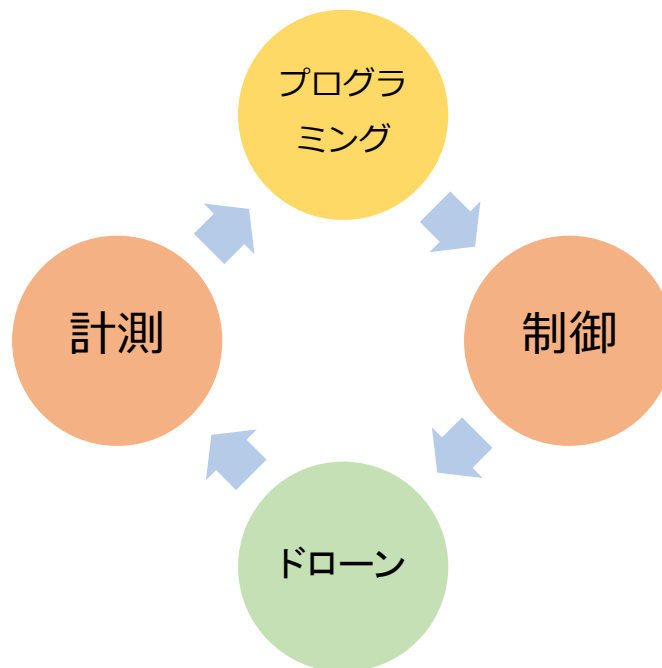
- ドローン
- 学習：飛行制御
- 課題：飛行制御

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**

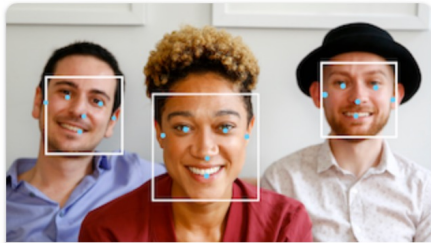
まとめ

- **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
 - **クラスとオブジェクト**を意識することにより理解の助けとなる
 - **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



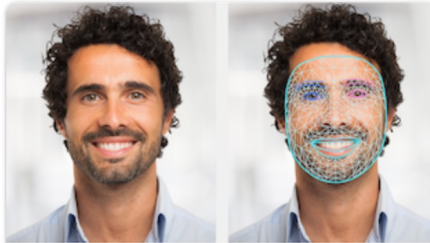
Mediapipe でできること

- Face Detection
- Face Landmark Detection
- Gesture Recognition
- Pose Landmark Detection
etc.



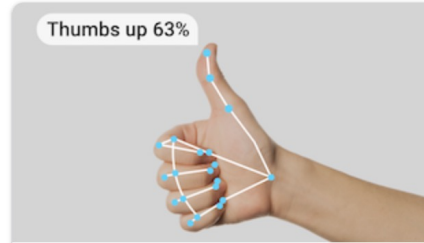
Face Detection
Detect faces in real time.

[See demo](#)



Face Landmark Detection
Detect face landmarks and
blendshape scores in real time.

[See demo](#)



Gesture Recognition
Identify and recognize hand gestures.

[See demo](#)

[Customize](#)



Pose Landmark Detection
Identify key points on the body in real time.

[See demo](#)

<https://mediapipe-studio.webapps.google.com/home>

課題研究へのお誘い

4_4_tracking_mediapipe.py (動作検証)

```
4_4_visual_tracking_mediapipe_utf8.py
61 while True:
62     frame = me.get_frame_read().frame 画像取得: opencv使用
63     if startflag:
64         feature_face = detect_max_face(frame)
65         pre_error = tracking_face(feature_face, frame.shape, pre_error)
66     else:
67         me.send_rc_control(0, 0, 0, 0) 顔検出結果 制御誤差
68
69     cv2.imshow('Face Tracking', cv2.resize(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB),
70     fps.count()
71     if cv2.waitKey(1) & 0xFF == ord('s'):
72         print("tracking start")
73         startflag = True
74     if cv2.waitKey(1) & 0xFF == ord('q'):
75         break
76
77 me.land()
78 me.streamoff()
79 me.end()
80 cv2.destroyAllWindows()
81
82 def detect_faces(img, offset={"x": 0, "y": 0}):
83     feature_faces = []
84     results = face_detection.process(img) 顔検出: mediapipe使用
85     if results.detections:
86         for detection in results.detections:
87             if detection.score[0] < DETECTION_CONFIDENCE: continue
88             bbox = detection.location_data.relative_bounding_box
89             ih, iw, _ = img.shape
90             x, y, w, h = int(bbox.xmin * iw), int(bbox.ymin * ih), int(bbox.width * iw)
91             cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)
```

• 面白いことを試してみませんか？

- 表情読み取り飛行制御
- 指差し飛行制御
- ジェスチャー飛行制御

力の余っている人
いませんか？



学術的興味点: オブジェクトの学びが「情報の科学的理解」にどのように役立つか？

→ ドローンに限らず皆さんの課題研究に役立ったなら事例報告をフィードバックして下さい