

科学の甲子園大阪府大会 標準開発環境の準備と使い方

科学の甲子園大阪府大会 実行委員会

令和元年九月

1 標準開発環境とは

本ドキュメントでは科学の甲子園大阪府大会でドローン制御を行うための標準的なソフトウェア開発環境の構築方法、およびソフトウェアの使い方について説明する。

ドローンの制御にはモータやセンサなどのハードウェアを直接制御するソフトウェアから、タスク実行のため大局観から戦術を制御するものまで抽象度に応じてさまざまなレベルのソフトウェアが必要になる。

本大会では高レベル API である Tello SDK (<https://www.ryzerobotics.com/jp/tello/downloads> → “Tello SDK”) 以上のレイヤを使うことを条件とする。選手諸君はもちろん Tello SDK を直接用いても構わない。しかしながら、Tello SDK を用いるだけでも UDP ネットワークプログラミング等の技術が必要になり、初心者が容易に開発できるとは言えない。

そこで、誰もが比較的簡単に本大会に臨むソフトウェアを記述できるように実行委員会が用意したのが本ドキュメントで解説する「標準開発環境」である。これを用いれば選手諸君はビジュアルプログラミング言語である Scratch を用いて簡単にドローン制御が可能になる。

本ドキュメントでは標準開発環境の準備および使い方について Windows 10 環境を前提とした説明を行う。なお、macOS および Linux についても大筋では同じように行うことができる。

2 環境構築

ユーザプログラムを開発するための Scratch、実行委員会が用意するサーバプログラムを実行するための Python 実行環境が必要である。本節ではこれらの環境構築方法について説明する。

2.1 Scratch 2.0 のインストール

Scratch 2.0 オフラインエディタをインストールする。最新版のバージョン 3.x では動作しないので注意すること。

1. <https://scratch.mit.edu/download/scratch2> を開く。

2. このページに書いてある通り、はじめに Adobe AIR をインストールする。
3. つづいて、Windows 版インストーラをダウンロードし、実行する。

2.2 Python + OpenCV 環境

Python 公式サイトから最新版をダウンロード・インストールし、その後で必要なモジュール群 (numpy と OpenCV) を追加インストールする。まずは Python である。基本的には OS に合わせた最新版をインストールすれば良い。

一番簡単なのは Microsoft Store からのインストールである (図 1)。



図 1 Microsoft Store から Python をインストール

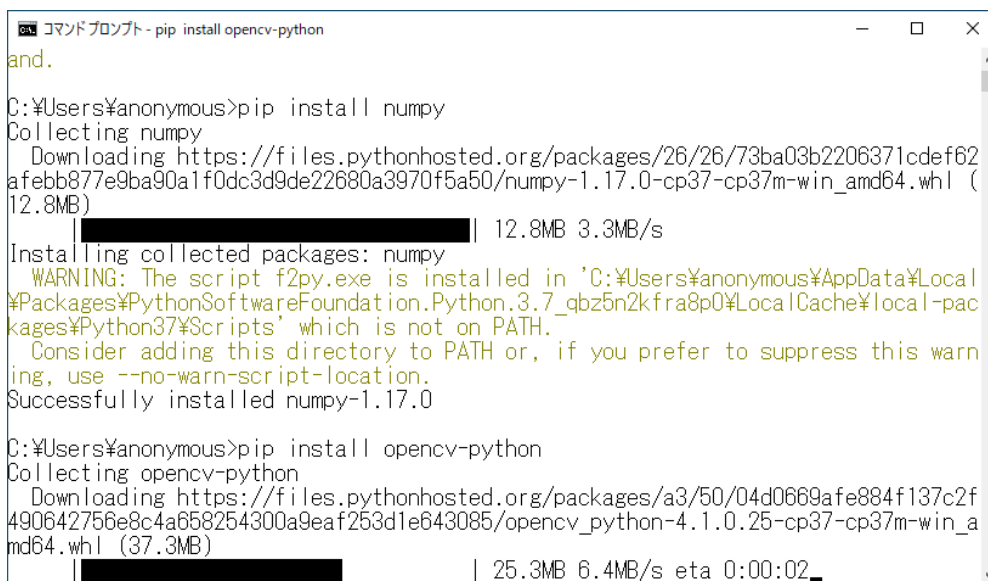
アカウントの都合など、Store からのインストールが難しい場合は以下の手順で Python 公式サイトからダウンロードしてインストールしても良い。

1. <https://www.python.org> を開く。
2. Downloads → Windows をクリックする。
3. 使用中の OS に合わせた最新版 (本ドキュメント執筆時では 3.7.4) のインストーラをダウンロードする。(例: Windows 10 64 bit 版を使っているのであれば Windows x86-64 executable installer)
4. インストーラを起動し、最初の画面で “Add Python 3.7 to PATH” にチェックを入れ、先に進み、インストールを完了する。

ここまでで Python がインストールされる。続いてモジュールを以下の手順でインストールする (図 2)。

1. コマンドプロンプトを開く (スタートメニュー → Windows システムツール → コマンドプロンプト)

2. `pip install --upgrade pip` と入力する。(ここでコマンドが見つからない等のエラーが出る場合は一度コマンドプロンプトを閉じて、再度試してみよ。それでも同じエラーが出る場合は Python を再インストールする。)
3. `pip install numpy` と入力する。
4. `pip install opencv-python` と入力する。



```
コマンドプロンプト - pip install opencv-python
and.
C:\Users\anonymous>pip install numpy
Collecting numpy
  Downloading https://files.pythonhosted.org/packages/26/26/73ba03b2206371cdef62
afebb877e9ba90a1f0dc3d9de22680a3970f5a50/numpy-1.17.0-cp37-cp37m-win_amd64.whl (
12.8MB)
    | ████████████████████████████████████████████████████████████████████████████ | 12.8MB 3.3MB/s
Installing collected packages: numpy
  WARNING: The script f2py.exe is installed in 'C:\Users\anonymous\AppData\Local
\ Packages\PythonSoftwareFoundation.Python.3.7_qbz5n2kfra8p0\LocalCache\local-pac
kages\Python37\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warn
ing, use --no-warn-script-location.
Successfully installed numpy-1.17.0

C:\Users\anonymous>pip install opencv-python
Collecting opencv-python
  Downloading https://files.pythonhosted.org/packages/a3/50/04d0669afe884f137c2f
490642756e8c4a658254300a9eaf253d1e643085/opencv_python-4.1.0.25-cp37-cp37m-win_a
md64.whl (37.3MB)
    | ████████████████████████████████████████████████████████████████████████████ | 25.3MB 6.4MB/s eta 0:00:02
```

図2 pip コマンドで必要なモジュールをインストール

2.3 甲子園ドローン制御ソフトウェアのダウンロードと保存

https://github.com/yagshi/koshien_osaka から [oitdroned.py](#) (Python プログラム) および [oitdrone.s2e](#) (Scratch 2.0 機能拡張ファイル) をダウンロードする。保存場所はどこでも構わないが、以下ではユーザを `anonymous` と仮定して、そのユーザフォルダ (`C:\Users\anonymous`) として説明する*1。

*1 ここで `anonymous` は Windows のユーザ名。

アカウントとユーザーフォルダ

Windows の場合、アカウントが **anonymous** だとすると、**ユーザーフォルダ**は

- エクスプローラでみると **PC → ローカルディスク (C:) → ユーザー → anonymous**
- コマンドプロンプトでは **C:\Users\anonymous**

となります (図 3)。これらはどちらも同じものです。標準開発環境ではコマンドプロンプトを多用します。コマンドプロンプトを開くと最初にユーザーフォルダが開きます。そのため、プログラムやデータは (ダウンロードフォルダ等ではなく) ユーザーフォルダに保存しておくのが便利です。

なお、Microsoft アカウントと紐付けられたアカウントの場合、フォルダ名はユーザ名の後ろに 000 のような数字がつく場合もあります。

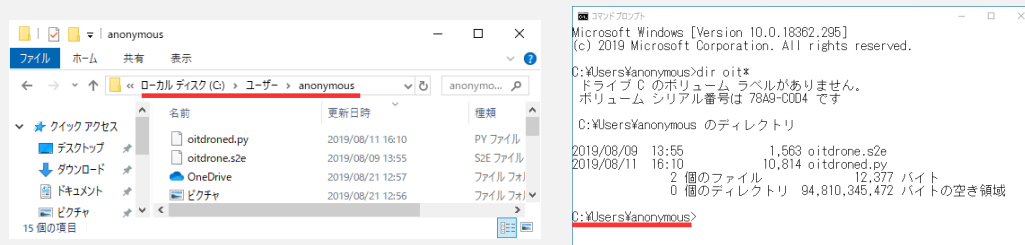


図 3 エクスプローラとコマンドプロンプトで見たユーザーフォルダ

3 ドローン制御プログラムの作成

3.1 システムの全体構成

前節までは各自の PC で一度だけ行えばいい環境構築である。本節ではドローン制御プログラム開発および実行時に必要に応じてその都度行う手順を説明する。

はじめに本システムの全体構成を図 4 に示す。図に示す通り、ユーザの書いた Scratch プログラムは一度 PC のバックエンドで動作する**ドローンサーバ** (oitdroned.py) に制御の指令を送る。ドローンサーバはユーザプログラムの指示を Tello 公式 API に変換してドローンに送信する。また、ドローンサーバは PC に接続された web カメラを使って画像処理によりドローンの位置を求める。この位置情報は Scratch 側で利用可能である。

このようなシステムを動作させるためには、以下の準備・操作が必要である。

1. カメラの接続
2. Scratch によるドローンサーバとの連携の準備
3. ドローンの起動と PC との無線接続
4. ドローンサーバの実行

5. Scratch 上のユーザプログラムの実行

順序はこのとおりでなくとも構わないが、開発・実行のプロセスを考えるとこの通りにやるのが一番便利である。また、上記手順の 1, 2 については一度行えばその日の作業が終わるまでやり直す必要はほぼない。3. はドローンの電源を切る (あるいはバッテリーが切れる) 度にやり直す必要がある。4. はドローンを PC に接続する度に、さらに何らかの事情でドローンサーバを停止する度に行う必要がある。

以下に上記手順の 1~5 を順番に説明する。

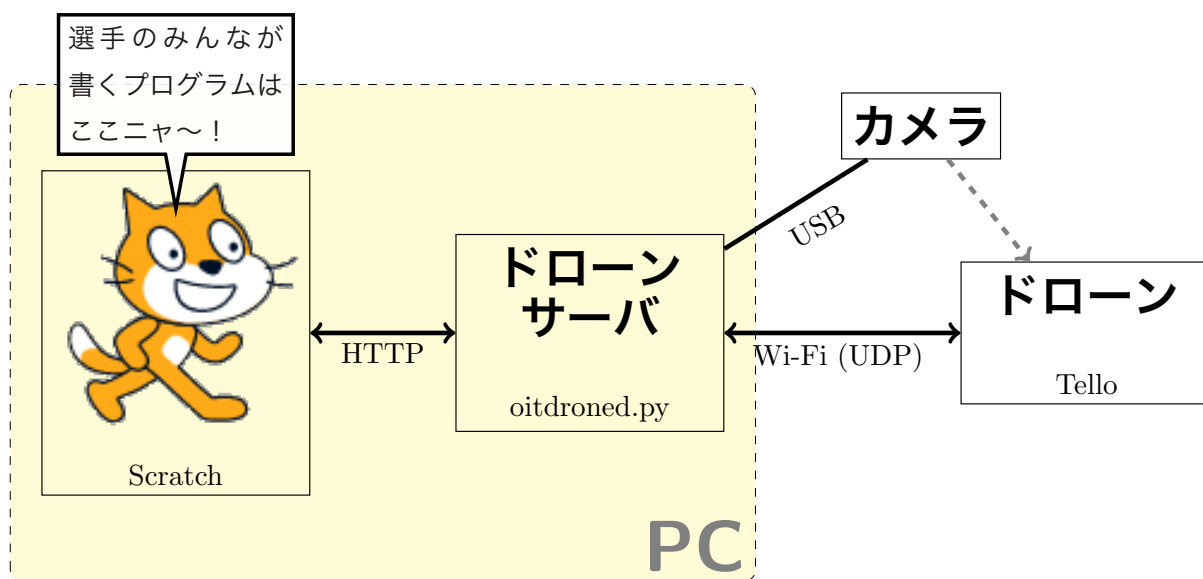


図4 システムの全体構成

3.2 カメラの接続

実行委員会指定の web カメラ (Logicool C270n) を接続する。

3.3 Scratch によるドローンサーバとの連携の準備

スタートメニュー → **S** → **Scratch 2** として Scratch 2.0 オフラインエディタを起動する。
[SHIFT] を押しながら [ファイル] メニューをクリックすると【実験的な HTTP 機能を読み込み】という隠しメニューが現れるのでこれを選択する (図 5)。続いて現れるファイル選択ダイアログから前節でダウンロードした **oitdrone.s2e** ファイルを選択する。

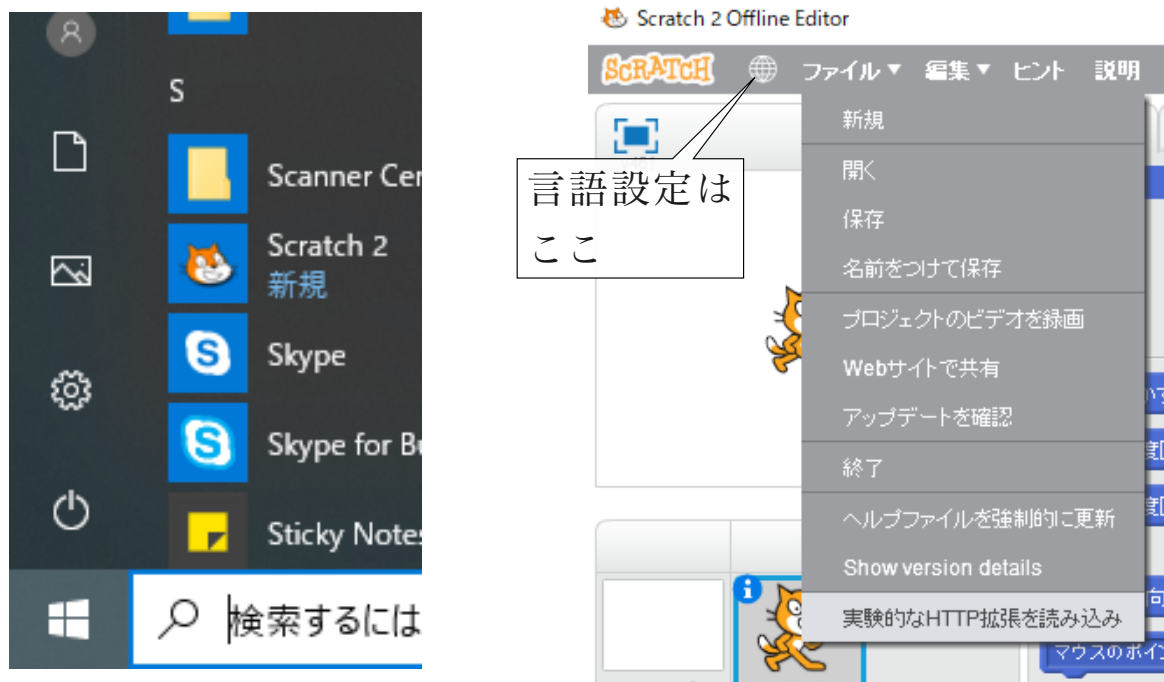


図5 Scratch 2 を起動して【実験的な HTTP 拡張を読み込み】を選択

すると、【その他】のカテゴリの中に【ドローン】の項目が表示され、図6のようにドローン関係のさまざまな機能ブロックが現れる。また、この部分には図6に示すように赤（あるいは緑）の丸表示があるが、これは後述するドローンサーバプログラムが動作して、Scratch 側との連携が正常に行われているかどうかを示すインジケータである。赤のときはサーバが起動していないもしくは連携に失敗している状態なので次項で説明するやり方でドローンサーバ (oitdroned.py) を起動する。緑のときは連携動作中である。

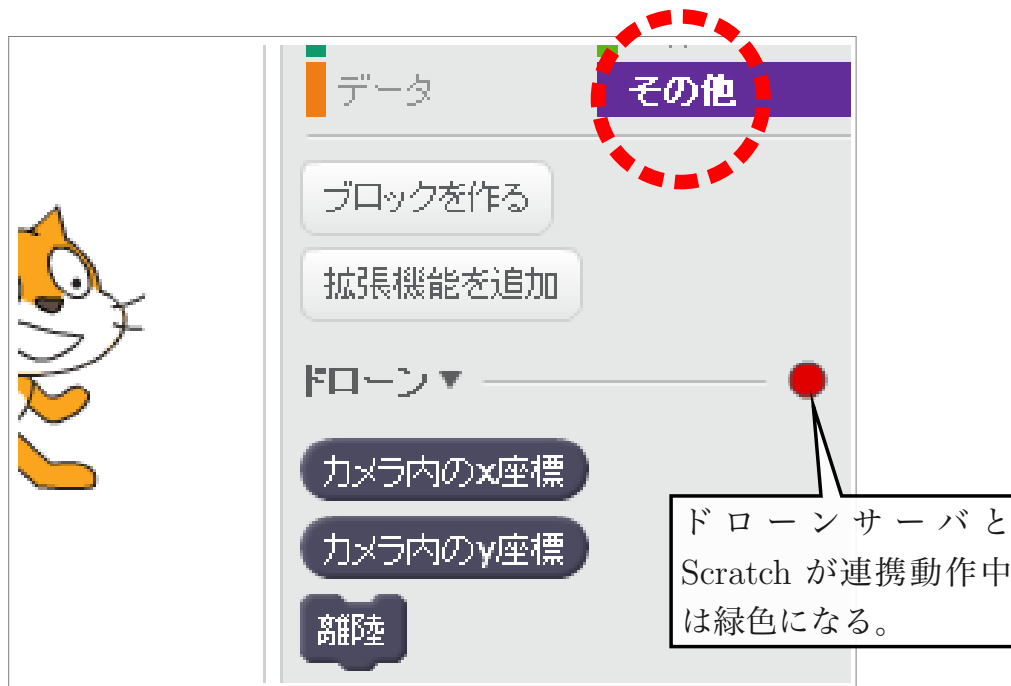


図6 ドローン拡張ブロック

3.4 ドローンの起動と PC との無線接続

これは Tello のマニュアル通りに行えばいいのでここでは概要だけ説明する。

はじめに Tello の電源を入れる。しばらくすると、**TELLO-??????** という SSID の Wi-Fi アクセスポイントが現れるのでそこに接続する。

なお、Tello との接続中、PC は無線によるインターネット接続が切れるので注意されたい。

3.5 ドローンサーバ (oitdroned.py) の実行

(ダウンロードした **oitdroned.py** がユーザーフォルダに保存されているという前提で説明するのでそうでない場合はまず **oitdroned.py** をユーザーフォルダに保存すること。)

まずコマンドプロンプトを起動する。PC にカメラが内蔵されていない場合は **python oitdroned.py** と入力する。内蔵カメラがある場合は **python oitdroned.py --camera 1** と入力する。

すべてが正常に動作すれば図 7 に示すように Python コマンドを入力したコマンドプロンプトに加えて web カメラが捉えた画像とドローンのフロントカメラが捉えた画像が画面に表示されるはずである。(もし、web カメラ画像が正しく表示されない場合は **python oitdroned.py --camera 1** の 1 を 0 や 2 に変えて試してみよ。この数字は PC が認識したカメラ番号を表す。)

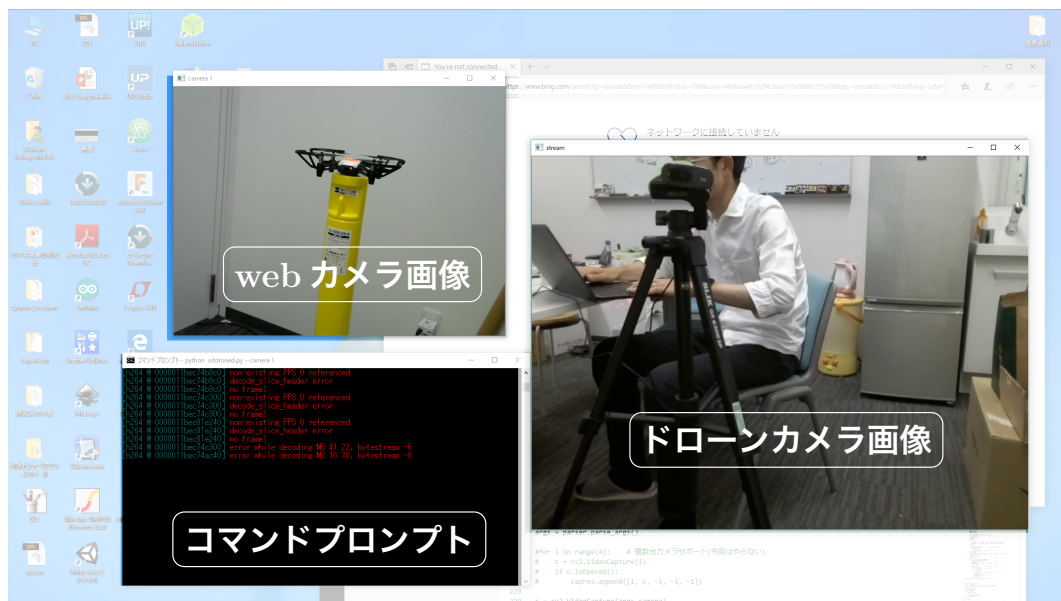


図7 ドローンサーバ (oitdroned.py) が正常に起動した際に表示される画面の例

3.6 Scratch 上のユーザプログラムの実行

ここまでできていれば図6で示した赤丸が緑色になっているはずである。これで Scratch からドローンを制御する準備が完全に整った。

プログラムを作成して試してみよう。図8のようにプログラムを作成し、緑の旗をクリックすればドローンが離陸し、1 m ほど向こう側に行って、こちらを振り返って写真を撮影し、元に位置に戻って着陸するはずである。ドローンが撮影した写真は画面に表示されるとともに `capture.jpg` というファイルで保存される。



図8 Scratch による「ドローン自撮り」プログラムの例

4 “設計” 演習

4.1 「自撮り」プログラムの問題点

前節で作った「自撮り」プログラム (図 8) は直観的に (いい加減に、適当に) 作ったものであり、これはエンジニアリング (工学) とは言わない。具体的には**往路の前進距離 100 cm に根拠がない**ことが問題である。

なお、180 度回頭するのは「**真後ろ (180 度方向) にいる被写体の方を向くため**」、復路の 100 cm 前進は「**往路の値と等しくして元の位置に戻るため**」という根拠がある*2ので工学的な“設計”と言っても差し支えなからう。

ものづくりにおいては、**根拠のないパラメタは極力排する**、極論するとあってはならないのがエンジニアリングである。

4.2 要求仕様に基づいた設計

では具体的に“設計”をしてみよう。はじめに自撮りの要求仕様を決める、つまり問題設定を行う。

問題設定

まあまあ仲の良い 3 人のチームメンバが並んで立っている姿をいい感じに自撮りする。

「まあまあ仲の良い 3 人」が並んで立つということは、肩を組んで密着するほどではないにせよ、比較的狭い間隔で立つと考えられる。そこで一人あたりの横幅は 50 cm としよう*3。3 人で 1.5 m である。「いい感じに自撮りする」ために、幅 1.5 m は完全にフレームに収めよう。

幅 1.5 m のものを撮影するには、ドローンはどの程度離れば良いであろう？ この問題を解くには、ドローンのカメラの水平画角を知る必要がある。調べて得られればそれを使えば良いが、筆者は検索があまり得意でないので実験的に確かめることにした。

図 9 がその実験である。分度器的な図をプリンタで出力し、その上でドローンを起動し、主観映像を撮影した。主観映像の左端すれすれのところにプラスドライバーが写っているのがわかる。図 9 左の「上から見た写真」によると、このドライバーは約 $+25^\circ$ の角度にある。したがって、**このドローンカメラの水平画角は約 50°** ということがわかる。

ここまでくればあとは計算するのみである。水平画角 50° のカメラで幅 1.5 m の被写体を撮影するのに必要な距離 d は、

$$\tan \frac{50^\circ}{2} = \frac{1.5/2}{d}$$

*2 両方とも当たり前のことだが、きちんと理由を説明できる点が工学設計において重要である。

*3 本来はこの 50 cm に根拠があるのか、という話になるが、ここではそこまで突っ込まない。(きちんと“設計”したい人は実際に仲良し三人組で並んで測ってみてください。)

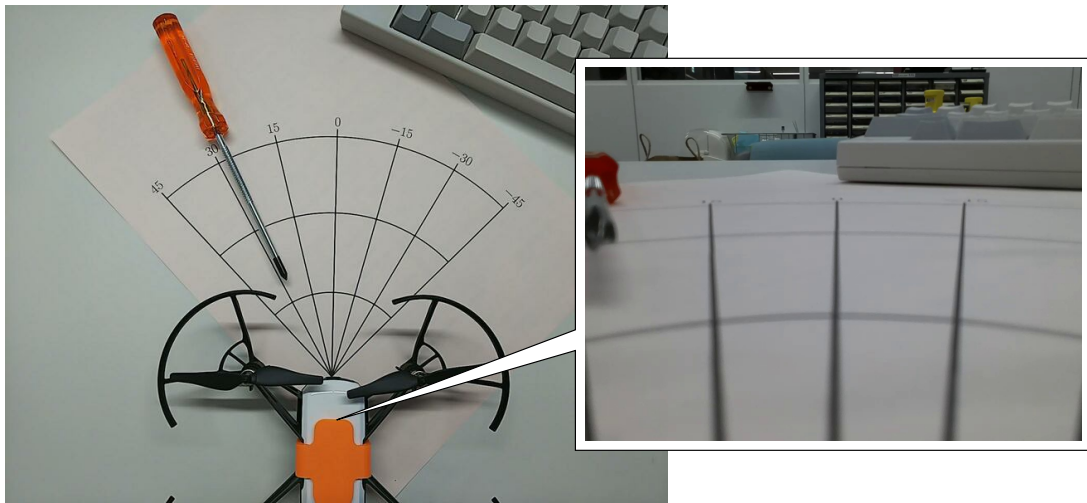


図9 ドローンの水平画角を調べる実験 (左: 実験の様子, 右: ドローン主観カメラの映像)

より

$$d \simeq 1.6$$

となる。なんと**適当に決めた 100 cm ではダメで、160 cm 飛ぶ必要があった。**

ただしドローンの飛行には誤差もあるし、多少風景が写り込んでいた方が自撮りには良いであろう。そこで安全係数として 1.2 を乗ずることにすると $1.6 \times 1.2 = 1.92$ となる。

さらにプログラムコードのメンテナンス性を考えると 192 よりも 200 の方が良いという考え方もあろう。そこで今回は最終的に**前進距離を 200 cm とする。**

前節 (図 8) の 100 cm は**何の根拠もない適当な値**で、エンジニアリングではない。一方、この 200 cm は仕様を決めて**数学的に導出した根拠のある値**であり、これこそが工学的な“設計”である。

大会当日はレポート課題が課せられるが、そこではプログラムソースコードのパラメタの根拠について**工学的な説明**がされることを期待する。

5 標準開発環境 API (Scratch ブロック) リファレンス

本節では標準開発環境の Scratch ブロックについて説明する。ただし、これら Scratch ブロックのうちドローンの飛行制御に関するものはすべて Tello SDK (<https://www.ryzerobotics.com/jp/tello/downloads> → “Tello SDK”) の対応機能の Scratch 版である。そのため Tello SDK に対応機能があるものについては詳細を Tello SDK の公式ドキュメントに譲る。

- **カメラ内の x 座標**

web カメラが捉えたドローンの x 座標を表す。ここで座標とはカメラのフレーム座標で図 10 のとおりである。つまり、web カメラで撮影したドローンがカメラのフレーム内の中央

にあれば x 座標は 360 程度になる。なお、ドローンを見失った場合、この値は -1 になる。
また、誤認識などにより複数点を認識してしまった場合は、**最大面積で認識したものの座標のみが返される**。

- **カメラ内の y 座標**

y である以外は (見失った場合に -1 になる点も含めて) 《カメラ内の x 座標》と同じ。

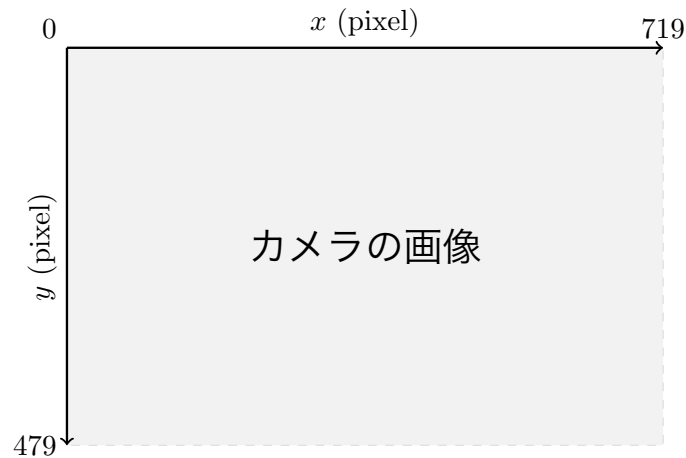


図 10 カメラ座標系

- **認識したマーカ数**

web カメラで認識したマーカの数。0 の場合はマーカを一つも見つけていない状態で、2 以上の場合は何かを誤認識して複数見えている場合が考えられる。1 の場合であっても、ドローンを正しく認識せず別のものを誤認識している可能性もあるので注意。

- **着陸**

Tello SDK の takeoff コマンド参照。

- **離陸**

Tello SDK の land コマンド参照。

- **??cm 前進**

Tello SDK の forward コマンド参照。

- **??cm 後退**

Tello SDK の back コマンド参照。

- **??cm 左移動**

Tello SDK の left コマンド参照。

- **??cm 右移動**

Tello SDK の right コマンド参照。

- **??cm 上昇**

Tello SDK の up コマンド参照。

- **??cm 下降**

Tello SDK の down コマンド参照。

- **移動速度を??cm/s に**

Tello SDK の speed コマンド参照。

- **??° CCW 回頭**

Tello SDK の ccw コマンド参照。

- **??° CW 回頭**

Tello SDK の cw コマンド参照。

- **?? ?? ?に速さ??で移動**

Tello SDK の go コマンド参照。

- **写真撮影??**

ファイル名を指定してドローンの主観画像を保存する。同一のファイル名が存在する場合は上書きするので注意すること。また写真を撮影すると同時に画面にも一枚ウィンドウを開いて表示する。

- **?? 方向宙返り**

Tello SDK の flip コマンド参照。

- **カーブ?? ?? ?? - ?? ?? ?? 速さ??**

Tello SDK の curve コマンド参照。

- **速度制御 左右=?? 前後=?? 上下=?? 回転=??**

Tello SDK の rc コマンド参照。ただし、このコマンドに限り他の移動系コマンドと異なり**ブロッキングしない**。つまり、プログラムの実行後、即時に次のブロックに進む*4。そのため、このコマンド直後に別の移動コマンド、例えば《前進》などを配置すると、このコマンドは何の働きもしない。一見何のためにあるのか疑問に思うかも知れないが**カメラによるフィードバックを行う際には重宝する**。

- **電池残量 (%)**

Tello SDK の “TELLO STATE” の節にある bat の値。

- **高度 (cm)**

Tello SDK の “TELLO STATE” の節にある h の値。

- **気圧高度 (cm)**

Tello SDK の “TELLO STATE” の節にある baro の値。

- **tof (cm)**

Tello SDK の “TELLO STATE” の節にある tof の値。

- **最低温度**

Tello SDK の “TELLO STATE” の節にある templ の値。

*4 その他の移動コマンドは実行が完了するまで次のブロックには進まない。

- **最高温度**

Tello SDK の “TELLO STATE” の節にある `temph` の値。

- **pitch**

Tello SDK の “TELLO STATE” の節にある `pitch` の値。

- **roll**

Tello SDK の “TELLO STATE” の節にある `roll` の値。

- **yaw**

Tello SDK の “TELLO STATE” の節にある `yaw` の値。

- **x 加速度 (0.001G)**

Tello SDK の “TELLO STATE” の節にある `agx` の値。

- **y 加速度 (0.001G)**

Tello SDK の “TELLO STATE” の節にある `agy` の値。

- **z 加速度 (0.001G)**

Tello SDK の “TELLO STATE” の節にある `agz` の値。

- **x 速度 (cm/s)**

Tello SDK の “TELLO STATE” の節にある `vgx` の値。

- **y 速度 (cm/s)**

Tello SDK の “TELLO STATE” の節にある `vgy` の値。

- **z 速度 (cm/s)**

Tello SDK の “TELLO STATE” の節にある `vgz` の値。

ブロッキングとノンブロッキング

コンピュータのソフトウェアにおいて、何かの処理 (一般に入出力処理) を行う際、その処理が完了するまでプログラムの実行がそこで停止するようなものを**ブロッキング**処理と言います。例えばみなさんがスマートフォンで動画をダウンロードするとき、ダウンロードボタンをクリックしてダウンロード中も次の作業ができますが、これはノンブロッキング処理だからです^a。ノンブロッキングは便利なが多いのですが、ドローン制御の場合に「50 cm 前進」がブロッキングしないと前進が完了する前に次のコマンドを送信してしまうため、カメラ画像でフィードバックしたいときや手動による緊急停止動作を行いたい場合などに都合が悪くなります。そのため `oitdroned.py` では速度制御など一部のコマンド以外ではブロッキングする (ドローンからの動作完了の応答を待ってから Scratch の次のブロックに処理が移る) ようになっています。

^a スマートフォンのダウンロードの場合は、本当はそこまで単純な話ではないのですが、この文脈上のニュアンス的にはそんなものと思ってください。