

Tello 飛行実験ソース

美都

2020 年 11 月 8 日

目次

1	Tello 本体 仕様	2
1.1	Tello の制御	2
2	lib.rs	6
3	コントローラー	6
4	エラークラス	6
5	ステータスモジュール	7
5.1	モジュールトップ	7
5.2	データクラス	7
5.3	マネージャクラス	10

1 Tello 本体 仕様

1.1 Tello の制御

制御は、192.168.10.1:8889 に対して、UDP でコントロールコマンドをテキストで送る。
コントロールコマンド列は、次のようになる。

制御コマンド

コントローラーの制御コマンド群。レスポンスは、ok/error。

コマンド	動作
command	SDK 制御 ON
streamon	ビデオストリーム オン
streamoff	ビデオストリーム オフ
emergency	緊急停止
mon	ミッションパッド有効
moff	ミッションパッド無効
mdirection x	ミッションパッドの検知モード設定
	$x=0$ 下方向のみ有効
	$x=1$ 前方のみ有効
	$x=2$ 下・前方の両方が有効
	$x=0,1$ の時、ステータス取得が 20Hz。 $x=2$ の時、ステータス取得が 10Hz。
ap ssid pass	Tello の Wi-Fi を端末モードに切り替える ssid と pass には、AP の ssid とパスワードを指定する。
wifi ssid pass	Tello の ssid と pass を変更する。

離着陸

離着陸を行う。レスポンスは、ok/error。

コマンド	動作
takeoff	離陸する。
land	着陸する。

単純動作コマンド

移動のためのコマンド群。レスポンスは、ok/error。

コマンド	動作
up x	x cm 上昇する。 $20 \leq x \leq 500$ 。
down x	x cm 下降する。 $20 \leq x \leq 500$ 。
forward x	x cm 前進する。 $20 \leq x \leq 500$ 。
back x	x cm 後退する。 $20 \leq x \leq 500$ 。
left x	x cm 左に進む。 $20 \leq x \leq 500$ 。
right x	x cm 右に進む。 $20 \leq x \leq 500$ 。
cw x	x° 時計回りに旋回する。 $1 \leq x \leq 360$
ccw x	x° 半時計回りに旋回する。 $1 \leq x \leq 360$
speed x	移動速度を $x(\text{cm/s})$ に設定する。 $10 \leq x \leq 100$
stop	その場でホバリングする。

複合動作コマンド

移動のためのコマンド群。レスポンスは、ok/error。

全てのコマンドで、 $|x_n|, |y_n|, |z_n|$ は、同時に 20 以下になってはいけない。さらに、各々の値は、

$$0 < x_n, y_n, z_n < 500(\text{cm})$$

$$10 < \text{speed} < 100(\text{cm/s})$$

を満たす。

コマンド	動作
flip x	x で示す方向に宙返りする。 "l" 左 "r" 右 "f" 前方 "b" 後方
go $x\ y\ z\ \text{speed}$	現位置を基準とし、 (x, y, z) へ $\text{speed}(\text{cm/s})$ で移動する。
curve $x_1\ y_1\ z_1\ x_2\ y_2\ z_2\ \text{speed}$	座標 (x_1, y_1, z_1) を経由して、 (x_2, y_2, z_2) へ $\text{speed}(\text{cm/s})$ で移動する。移動経路の半径 r は、 $0.5 < r < 10\text{m}$ とする。条件を満たさない場合、error を返す。

ミッションパッドコマンド

ミッションパッド関係のコマンド群。

コマンド中 mid_n は、ミッションパッド ID を意味する。書式は、"m1-m8" となる。

レスポンスは、ok/error。

全てのコマンドで、 $|x_n|, |y_n|, |z_n|$ は、同時に 20 以下になってはいけない。さらに、各々の値は、

$$0 < x_n, y_n, z_n < 500(cm)$$

$$10 < speed < 100(cm/s)$$

を満たす。

コマンド	動作
$go\ x\ y\ z\ speed\ mid$	mid のパッドを基点として、 $speed(cm/s)$ で、 (x, y, z) の位置に移動する。
$curve\ x_1\ y_1\ z_1\ x_2\ y_2\ z_2\ speed\ mid$	ミッションパッド mid を基点として、座標 (x_1, y_1, z_1) を経由して、 (x_2, y_2, z_2) へ $speed(cm/s)$ で移動する。移動経路の半径 r は、 $0.5 < r < 10m$ とする。条件を満たさない場合、error を返す。
$jump\ x\ y\ z\ speed\ yaw\ mid_1\ mid_2$	ミッションパッド mid_1 より、 mid_2 へ、 (x, y, z) を経由して移動し、 yaw° 旋回する。

プロポコマンド

プロポ操作のコマンド。

各動作方向のチャンネルの操作量を指定する。

コマンド	動作
$rc\ a\ b\ c\ d$	"a" 左右
	"b" 前後
	"c" 上下
	"d" 旋回
$-100 \leq a, b, c, d \leq 100$	

問い合わせコマンド

各種問い合わせコマンド。

レスポンスは、問い合わせの結果。

コマンド	動作	レスポンス
speed?	現在の速度 (cm/s)	10-100
battery?	バッテリーの残量	0-100
time?	今回のフライト時間	秒数
wifi?	Wi-Fi 電波の SNR 比	SNR 値

2 lib.rs

lib.rs

```
1  /// Telloのコントロールライブラリ
2
3  /// Telloの制御
4  mod control {}
5
6  /// Telloのステータスの取得
7  pub mod status;
8
9  /// Telloライブラリー用エラー
10 pub mod error;
```

3 コントローラー

Tello のコントロールを司る。

4 エラークラス

エラー処理クラス。ライブラリーの全てのエラーを包含する。

error.rs

```
1  pub enum TelloError {
2      SocketError(std::io::Error),
3      TelloCmdFail(String),
4      TelloResponsIllegal(String),
5  }
6
7  impl From<std::io::Error> for TelloError {
8      fn from(e: std::io::Error) -> Self {
9          Self::SocketError(e)
10     }
11 }
12
13 impl std::fmt::Display for TelloError {
14     fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
15         use TelloError::*;
16         match self {
17             SocketError(e) => write!(f, "SocketError: {}", e),
18             TelloCmdFail(s) => write!(f, "Tello Error: {}", s),
19             TelloResponsIllegal(s) => write!(f, "Illegal Respons from tello
20                                     .[{}]", s),
21         }
22     }
23 }
```

```

21     }
22 }
23
24 impl std::fmt::Debug for TelloError {
25     fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
26         write!(f, "{}", self)
27     }
28 }

```

5 ステータスモジュール

Tello のステータスを取得するための処理。

5.1 モジュールトップ

mod.rs

```

1  /// ステータスデータ
2  pub mod data;
3  /// ステータス取得の管理
4  pub mod manager;

```

5.2 データクラス

ステータスデータを表すクラス。FromStr を実装し、UDP からの受信データに対して、parse が可能。

data.rs

```

1  /// Telloのステータスデータ
2  #[derive(Default, Debug, PartialEq, Clone)]
3  pub struct StatusData {
4      pub mid: i32,
5      pub x: i32,
6      pub y: i32,
7      pub z: i32,
8      pub mpry: (i32, i32, i32),
9      pub pitch: i32,
10     pub roll: i32,
11     pub yaw: i32,
12     pub vgx: i32,
13     pub vgy: i32,
14     pub vgz: i32,
15     pub templ: i32,
16     pub temph: i32,
17     pub tof: i32,
18     pub h: i32,
19     pub bat: u32,

```

```

20     pub baro: f64,
21     pub time: i32,
22     pub agx: f64,
23     pub agy: f64,
24     pub agz: f64,
25 }
26
27 impl StatusData {
28     /// 数値文字列を指定の数値型に変換する。
29     /// (ステータス解析のユーティリティー)
30     fn parse<I>(item: &str, src: &str) -> I
31     where
32         I: std::str::FromStr + Default,
33     {
34         item.parse::<I>().unwrap_or_else(|_| {
35             eprintln!("field value error:[{}]", src);
36             I::default()
37         })
38     }
39 }
40
41 impl std::str::FromStr for StatusData {
42     type Err = TelloStatusParseError;
43
44     /// Telloの受信データの文字列を解析する
45     fn from_str(src: &str) -> Result<Self, TelloStatusParseError> {
46         let mut ret = Self::default();
47
48         let end = match src.match_indices("\r\n").next() {
49             Some((cnt, _)) => cnt,
50             None => src.len(),
51         };
52
53         for pair in src[0..end].trim().split(';') {
54             let item: Vec<&str> = pair.split(':').collect();
55             if item.len() == 1 {
56                 continue;
57             } else if item.len() != 2 {
58                 eprintln!("field format error1: [{}]", pair);
59                 continue;
60             }
61
62             match item[0].trim() {
63                 "mid" => ret.mid = Self::parse::<i32>(item[1], pair),
64                 "x" => ret.x = Self::parse::<i32>(item[1], pair),
65                 "y" => ret.y = Self::parse::<i32>(item[1], pair),
66                 "z" => ret.z = Self::parse::<i32>(item[1], pair),
67                 "pitch" => ret.pitch = Self::parse::<i32>(item[1], pair),

```



```

68         "roll" => ret.roll = Self::parse::i32(item[1], pair),
69         "yaw" => ret.yaw = Self::parse::i32(item[1], pair),
70         "vgx" => ret.vgx = Self::parse::i32(item[1], pair),
71         "vgy" => ret.vgy = Self::parse::i32(item[1], pair),
72         "vgz" => ret.vgz = Self::parse::i32(item[1], pair),
73         "templ" => ret.templ = Self::parse::i32(item[1], pair),
74         "temph" => ret.temph = Self::parse::i32(item[1], pair),
75         "tof" => ret.tof = Self::parse::i32(item[1], pair),
76         "h" => ret.h = Self::parse::i32(item[1], pair),
77         "bat" => ret.bat = Self::parse::u32(item[1], pair),
78         "baro" => ret.baro = Self::parse::f64(item[1], pair),
79         "time" => ret.time = Self::parse::i32(item[1], pair),
80         "agx" => ret.agx = Self::parse::f64(item[1], pair),
81         "agy" => ret.agy = Self::parse::f64(item[1], pair),
82         "agz" => ret.agz = Self::parse::f64(item[1], pair),
83         "mpry" => {
84             let values: Vec<&str> = item[1].split(',').collect();
85             if values.len() == 3 {
86                 ret.mpry = (
87                     Self::parse::i32(values[0], pair),
88                     Self::parse::i32(values[1], pair),
89                     Self::parse::i32(values[2], pair),
90                 );
91             } else {
92                 eprintln!("field format error2: [{}] ", pair);
93             }
94         }
95         _ => {}
96     }
97 }
98
99     Ok(ret)
100 }
101 }
102
103 ///ステータス変換用のエラー。実質不使用。
104 #[derive(Debug, PartialEq, Clone)]
105 pub struct TelloStatusParseError();
106
107 impl std::fmt::Display for TelloStatusParseError {
108     fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
109         write!(f, "ステータス解析失敗。でも出るはずがない。")
110     }
111 }

```

5.3 マネージャクラス

UDP 通信を管理し、ステータスの取得を可能とする。

manager.rs

```
1  /// ステータス取得の管理
2  use super::data::StatusData;
3  use crate::error::TelloError;
4  use std::net::UdpSocket;
5  use std::str;
6  use std::sync::mpsc;
7  use std::thread;
8
9  /// Telloのステータス受信とデータの管理
10 #[derive(Debug)]
11 pub struct Manager {
12     data: StatusData,
13     rx: mpsc::Receiver<StatusData>,
14 }
15
16 impl Manager {
17     /// Manageの生成。
18     ///
19     /// # 引数
20     /// Telloステータス受信用ソケットか、None。
21     /// Noneの場合、デフォルトとして"0.0.0.0:8890"のポートを使用する。
22     ///
23     pub fn new(socket: impl Into<Option<UdpSocket>>) -> Result<Self, TelloError> {
24         {
25             let socket = socket.into().unwrap_or(UdpSocket::bind("0.0.0.0:8890")?);
26             // データ受信スレッドの生成
27             let (tx, rx) = mpsc::channel();
28             thread::spawn(move || {
29                 Self::recieve_proc(socket, tx);
30             });
31
32             Ok(Self {
33                 data: StatusData::default(),
34                 rx,
35             })
36         }
37
38     /// 【内部関数】ステータス受信スレッドの本体。
39     fn recieve_proc(socket: UdpSocket, tx: mpsc::Sender<StatusData>) -> ! {
40         loop {
41             let mut stat_buf = [0; 1024];
42             let (len, _addr) = socket.recv_from(&mut stat_buf).unwrap_or_else(|
```

```

42         e| {
43             eprintln!("ステータス受信ユニット:ソケット受信エラー->{:?}", e
44                 );
45             std::process::exit(1);
46         });
47     let stat: StatusData = str::from_utf8(&stat_buf[0..len]).unwrap().
48         parse().unwrap();
49     tx.send(stat).unwrap_or_else(|e| {
50         eprintln!("ステータス受信ユニット:プロセス通信エラー{:?}", e);
51         std::process::exit(1);
52     });
53 }
54 }
55
56 /// 受信した最新のステータスデータを返す。
57 /// 内部ステータスデータ構造体を最新データに更新するため、mutが必要。
58 pub fn get_data(&mut self) -> StatusData {
59     // メッセージの受信とデータ更新
60     for rx_data in self.rx.try_iter() {
61         self.data = rx_data;
62     }
63     self.data.clone()
64 }
65 }

```