Tello 飛行実験ソース

美都

2020 年 11 月 18 日

目次

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 <= x <= 500。
down x	$x$cm 下降する。20 <= x <= 500。
forward x	$x$cm 前進する。20 <= x <= 500。
back x	$x$cm 後退する。20 <= x <= 500。
left x	$x$cm 左に進む。20 <= x <= 500。
right x	$x$cm 右に進む。20 <= x <= 500。
cw x	x° 時計回りに旋回する。1 <= x <= 360
ccw x	x° 半時計回りに旋回する。1 <= x <= 360
speed x	移動速度を x(cm/s) に設定する。10 <= x <= 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、

$$0 < x_n, y_n, z_n < 500(cm)$$
$$10 < speed < 100(cm/s)$$

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 $speed$	現位置を基準とし、(x, y, z) へ $speed(cm/s)$ で移動する。
courve x_1 y_1 z_1 x_2 y_2 z_2 $speed$	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。

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。
courve 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 $-100 <= a, b, c, d <= 100$

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
sdk?	SDK バージョン番号	バージョン
sn?	TELLO のシリアル番号	シリアル

2 lib.rs

lib.rs

```
//! Telloのコントロールライブラリ

/// Telloの制御
pub mod control;

/// Telloのステータスの取得
pub mod status;

/// Telloライブラリー用エラー
pub mod error;
```

3 コントローラー

Tello のコントローロールを司る。

control.rs

```
// Telloの制御
use crate::error::TelloError;
use array_macro::*;
use std::net::{ToSocketAddrs, UdpSocket};
use std::num::Wrapping;
use std::sync::mpsc;
use std::thread;
use std::time::Duration;

const TELLO_CMD_IP: &str = "192.168.10.1:8889";
const TELLO_CMD_BIND: &str = "0.0.0.0:0";
const JOB_RETS_SIZE: usize = 16;

/// Telloのコントローラー
#[derive(Debug)]
pub struct Controller {
    cmd_sender: mpsc::Sender<Job>,
    ret_receiver: mpsc::Receiver<JobRet>,
    next_job_no: Wrapping<u16>,
    job_rets: [JobRet; JOB_RETS_SIZE],
    job_rets_cur_idx: usize,
}

impl Controller {
    /// コントローラーを立ち上げる。
    /// # 引数
```

```
27      ///  - tello_ip : Telloのipアドレス、及び、ポート番号。
28      pub fn new_with_ip(tello_ip: impl ToSocketAddrs) -> Result<Self, TelloError
         > {
29          let socket = UdpSocket::bind(TELLO_CMD_BIND)?;
30          socket.connect(tello_ip)?;
31          let (cmd_tx, cmd_rx) = mpsc::channel();
32          let (ret_tx, ret_rx) = mpsc::channel();
33          thread::spawn(move || {
34              Self::send_proc(socket, cmd_rx, ret_tx);
35          });
36
37          Ok(Self {
38              cmd_sender: cmd_tx,
39              ret_receiver: ret_rx,
40              next_job_no: Wrapping(1u16),
41              job_rets: array![JobRet { id: 0, ret: Ok(0) }; JOB_RETS_SIZE],
42              job_rets_cur_idx: 0,
43          })
44      }
45
46      /// コントローラーを立ち上げる
47      ///
48      /// Telloのipは、"192.168.10.1:8889"とする。
49      pub fn new() -> Result<Self, TelloError> {
50          Controller::new_with_ip(TELLO_CMD_IP)
51      }
52
53      pub fn exec_cmd(&mut self, cmd: TelloCommand) -> Result<u32, TelloError> {
54          let job = Job {
55              id: self.next_job_no.0,
56              cmd,
57          };
58          self.cmd_sender.send(job).expect("コマンド送信パイプエラー");
59          let mut ret_val = Err(TelloError::TelloResponsIllegal("Time out.".
             to_string()));
60          loop {
61              self.recv_job_ret();
62              if let Some(ret) = self.find_job_rets(self.next_job_no.0) {
63                  ret_val = ret.ret.clone();
64                  break;
65              }
66              thread::sleep(Duration::from_millis(100));
67          }
68          self.next_job_no += Wrapping(1);
69          ret_val
70      }
71
72      fn recv_job_ret(&mut self) {
```

```
73          while let Ok(ret) = self.ret_receiver.try_recv() {
74              self.add_job_rets(ret);
75          }
76      }
77
78      /// 戻り値バッファにリターン値を追加する。
79      fn add_job_rets(&mut self, ret: JobRet) {
80          let idx = (self.job_rets_cur_idx + 1) % JOB_RETS_SIZE;
81          self.job_rets[idx] = ret;
82          self.job_rets_cur_idx = idx;
83      }
84
85      /// jobのidよりリターン値を検索する。
86      fn find_job_rets(&self, id: u16) -> Option<&JobRet> {
87          let mut idx = self.job_rets_cur_idx;
88          while idx != (self.job_rets_cur_idx + 1) % JOB_RETS_SIZE {
89              if self.job_rets[idx].id == id {
90                  return Some(&self.job_rets[idx]);
91              }
92              idx = match idx {
93                  0 => JOB_RETS_SIZE - 1,
94                  n => n - 1,
95              }
96          }
97          None
98      }
99
100     /// 内部関数: 指定されたコマンドをTelloに非同期で送信するスレッド本体
101     ///
102     /// # Panics
103     ///   送受信に使用するパイプにエラーが出るとパニックする。
104     fn send_proc(
105         tello_socket: UdpSocket,
106         cmd_recv: mpsc::Receiver<Job>,
107         ret_send: mpsc::Sender<JobRet>,
108     ) -> ! {
109         let mut buff = [0; 10];
110         let err_recv = "コマンド送信部エラー。コマンド送信用パイプ不良";
111         let err_send = "コマンド送信部エラー。コマンド結果返信用パイプ不良";
112         loop {
113             let Job { id, cmd } = cmd_recv.recv().expect(err_recv);
114             if let Err(e) = tello_socket.send(cmd.to_string().as_bytes()) {
115                 let ret = JobRet {
116                     id,
117                     ret: Err(e.into()),
118                 };
119                 ret_send.send(ret).expect(err_send);
120                 continue;
```

```
            }
            let ret = match tello_socket.recv(&mut buff) {
                Ok(i) => match &buff[0..i] {
                    b"ok" => Ok(0),
                    b"error" => Err(TelloError::TelloCmdFail(cmd.to_string())),
                    _ => std::str::from_utf8(&buff[0..i])
                        .unwrap()
                        .parse::<u32>()
                        .or(Err(TelloError::TelloResponsIllegal(
                            String::from_utf8(buff[0..i].to_vec()).unwrap(),
                        ))),
                },
                Err(e) => Err(e.into()),
            };
            ret_send.send(JobRet { id, ret }).expect(err_send);
        }
    }
}

#[derive(Debug)]
struct Job {
    id: u16,
    cmd: TelloCommand,
}

#[derive(Debug)]
struct JobRet {
    id: u16,
    ret: Result<u32, TelloError>,
}

#[derive(Debug)]
pub enum TelloCommand {
    Command,
    Takeoff,
    Land,
}

impl ToString for TelloCommand {
    fn to_string(&self) -> String {
        use TelloCommand::*;
        match self {
            Command => "command".to_string(),
            Takeoff => "takeoff".to_string(),
            Land => "land".to_string(),
        }
    }
}
```

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

```rust
#[derive(Clone)]
pub enum TelloError {
    SocketError(String),
    TelloCmdFail(String),
    TelloResponsIllegal(String),
}

impl From<std::io::Error> for TelloError {
    fn from(e: std::io::Error) -> Self {
        Self::SocketError(e.to_string())
    }
}

impl std::fmt::Display for TelloError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        use TelloError::*;
        match self {
            SocketError(e) => write!(f, "SocketError: {}", e),
            TelloCmdFail(s) => write!(f, "Tello Error: {}", s),
            TelloResponsIllegal(s) => write!(f, "Illegal Respons from tello
              .[{}]", s),
        }
    }
}

impl std::fmt::Debug for TelloError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "{}", self)
    }
}

impl std::error::Error for TelloError {}
```

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

```rust
/// ステータスデータ
pub mod data;
/// ステータス取得の管理
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

```rust
/// Telloのステータスデータ
#[derive(Default, Debug, PartialEq, Clone)]
pub struct StatusData {
    pub mid: i32,
    pub x: i32,
    pub y: i32,
    pub z: i32,
    pub mpry: (i32, i32, i32),
    pub pitch: i32,
    pub roll: i32,
    pub yaw: i32,
    pub vgx: i32,
    pub vgy: i32,
    pub vgz: i32,
    pub templ: i32,
    pub temph: i32,
    pub tof: i32,
    pub h: i32,
    pub bat: u32,
    pub baro: f64,
    pub time: i32,
    pub agx: f64,
    pub agy: f64,
    pub agz: f64,
}

impl StatusData {
    /// 数値文字列を指定の数値型に変換する。
    /// (ステータス解析のユーティリティー)
    fn parse<I>(item: &str, src: &str) -> I
    where
        I: std::str::FromStr + Default,
    {
        item.parse::<I>().unwrap_or_else(|_| {
            eprintln!("field value error:[{}]", src);
            I::default()
        })
    }
```

```
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 = (
```

```
                                Self::parse::<i32>(values[0], pair),
                                Self::parse::<i32>(values[1], pair),
                                Self::parse::<i32>(values[2], pair),
                            );
                        } else {
                            eprintln!("field format error2: [{}]", pair);
                        }
                    }
                    _ => {}
                }
            }

            Ok(ret)
        }
}

///ステータス変換用のエラー。実質不使用。
#[derive(Debug, PartialEq, Clone)]
pub struct TelloStatusParseError();

impl std::fmt::Display for TelloStatusParseError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "ステータス解析失敗。でも出るはずがない。")
    }
}
```

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

```
/// ステータス取得の管理
use super::data::StatusData;
use crate::error::TelloError;
use std::net::UdpSocket;
use std::str;
use std::sync::mpsc;
use std::thread;

/// Telloのステータス受信とデータの管理
#[derive(Debug)]
pub struct Manager {
    data: StatusData,
    rx: mpsc::Receiver<StatusData>,
}

impl Manager {
    /// 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          let socket = socket.into().unwrap_or(UdpSocket::bind("0.0.0.0:8890")?);
25          // データ受信スレッドの生成
26          let (tx, rx) = mpsc::channel();
27          thread::spawn(move || {
28              Self::recieve_proc(socket, tx);
29          });
30
31          Ok(Self {
32              data: StatusData::default(),
33              rx,
34          })
35      }
36
37      /// 【内部関数】ステータス受信スレッドの本体。
38      fn recieve_proc(socket: UdpSocket, tx: mpsc::Sender<StatusData>) -> ! {
39          loop {
40              let mut stat_buf = [0; 1024];
41              let (len, _addr) = socket.recv_from(&mut stat_buf).unwrap_or_else(|
                 e| {
42                  eprintln!("ステータス受信ユニット:ソケット受信エラー->{:?}", e
                     );
43                  std::process::exit(1);
44              });
45              let stat: StatusData = str::from_utf8(&stat_buf[0..len]).unwrap().
                 parse().unwrap();
46              tx.send(stat).unwrap_or_else(|e| {
47                  eprintln!("ステータス受信ユニット:プロセス通信エラー{:?}", e);
48                  std::process::exit(1);
49              });
50          }
51      }
52
53      /// 受信した最新のステータスデータを返す。
54      /// 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。
55      pub fn get_data(&mut self) -> StatusData {
56          // メッセージの受信とデータ更新
57          for rx_data in self.rx.try_iter() {
58              self.data = rx_data;
59          }
60
61          self.data.clone()
```

```
62          }
63  }
```