E2実験４日目
ドローンプロジェクトスライド

永井先生・藤田先生・M2白戸・M1横田・B4吉清

目次

1-1.必要なツールボックス

MATLABにインストールしておくべきもの

- Aerospace Blockset
- Optimization Toolbox
- Simulink Control Design
- Signal Processing Toolbox
- Computer Vision Toolbox
- Simulink 3D Animation

MATLABのバージョン

- R2020a
  インストール時に固まることがあるが，何回か試せば入る

MATLABコマンドで asbQuadcopterStartと打ち，なにか出てくるかテストしてください

1-2.課題の締め切り・評価方法

基礎課題に取り組んだ人
- レポートを作成(HPの評価方法参照)

発展課題に取り組んだ人
- 発表かレポートか選択(HPの評価方法参照)
  （発表の人も保険としてレポートを出してもいい）

  発展課題発表
  * 発表日
    - 7月9日
  * 出席者
    - 藤本先生・（古関先生）・大西先生・他の先生
    - B3
    - 暇なTA
  * 発表内容
    - 友達と取り組んだ場合自分がやったところ

1-3.参考資料

1. ドキュメンテーション “Quadcopter Project” [1]：
   Mathworksによるwebページ

2. ビデオ “Drone Simulation and Control” [2]

   part 1 クアドコプターの紹介（センサ，飛ぶ仕組みなど）
   part 2 制御器設計
   part 3 実験機への実装法（各ブロックの役割がわかる）
   part 4 モデルの作り方
   part 5 ＰＩＤ制御器のチューニング

[1] https://jp.mathworks.com/help/aeroblks/quadcopter-project.html

[2] https://jp.mathworks.com/videos/
drone-simulation-and-control-part-1-setting-up-the-control-problem-1539323440930.html

基礎課題

以下のすべての課題に対し解答してください。
基礎課題１　ドローンの回転の向きを説明してください
基礎課題２　ドローンの制御器ブロックを説明してください
基礎課題３　重力補償の有無による比較を行い，
　　z軸のシミュレーションを回し，
　　ステップ応答結果をプロットしてください
基礎課題４　極配置法によりPID制御器を設計し，
　　z軸のシミュレーションを回し，
　　シミュレーション結果をプロットしてください

基礎課題のヒント

* 基礎課題 1

Mathworks のビデオ Drone Simulation and Control の Part 1を参照

* 基礎課題 2

それぞれの制御器の役割を説明してください。特に，ピッチとロールに関係する制御器が 2 つあることがポイントです。
Mathworks のビデオ Drone Simulation and Control の Part2 を参照

[2] https://jp.mathworks.com/videos/drone-simulation-and-control-part-1-setting-up-the-control-problem-1539323440930.html

* 基礎課題 3

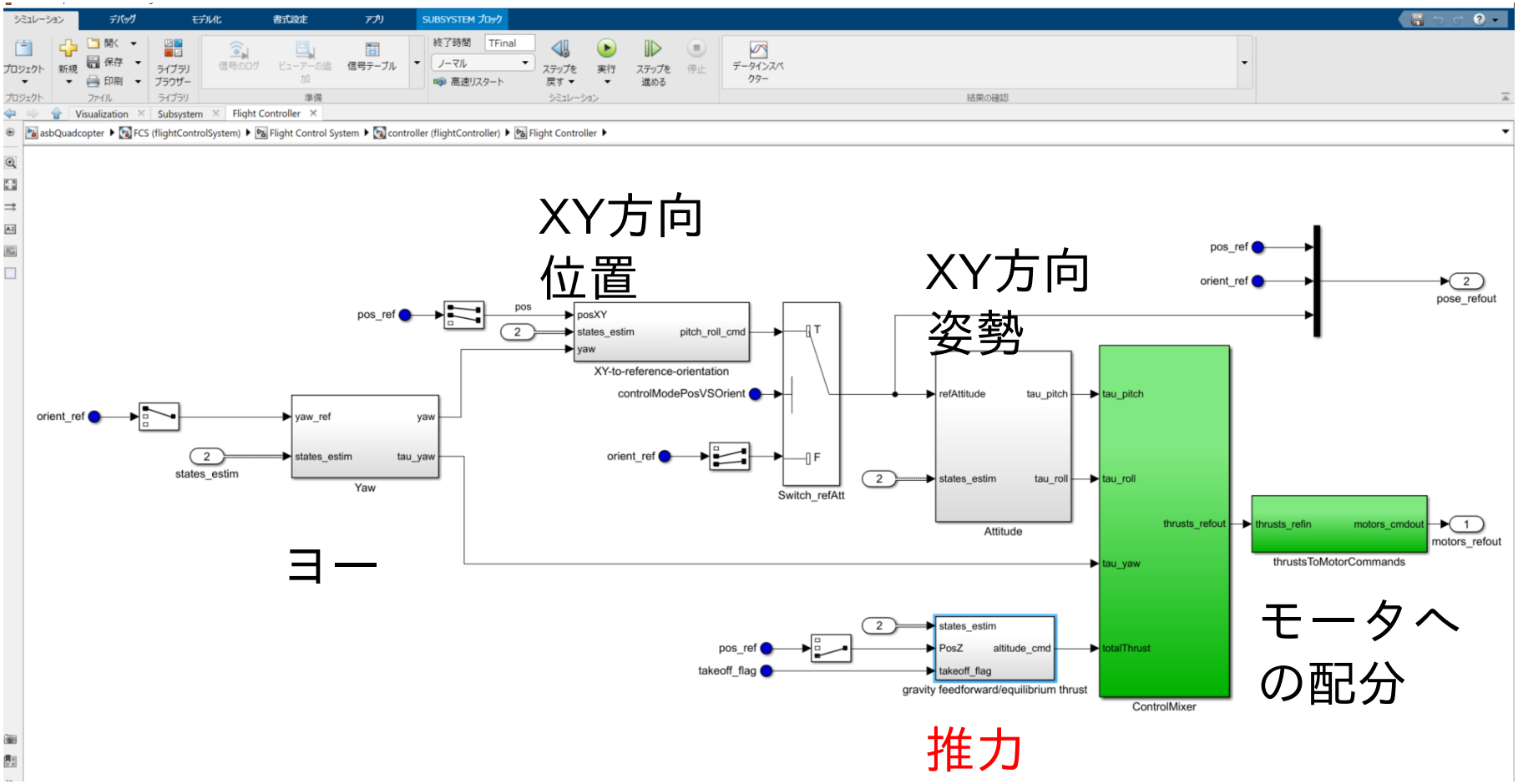
説明スライドを参照すること

* 基礎課題 4

説明スライドを参照すること

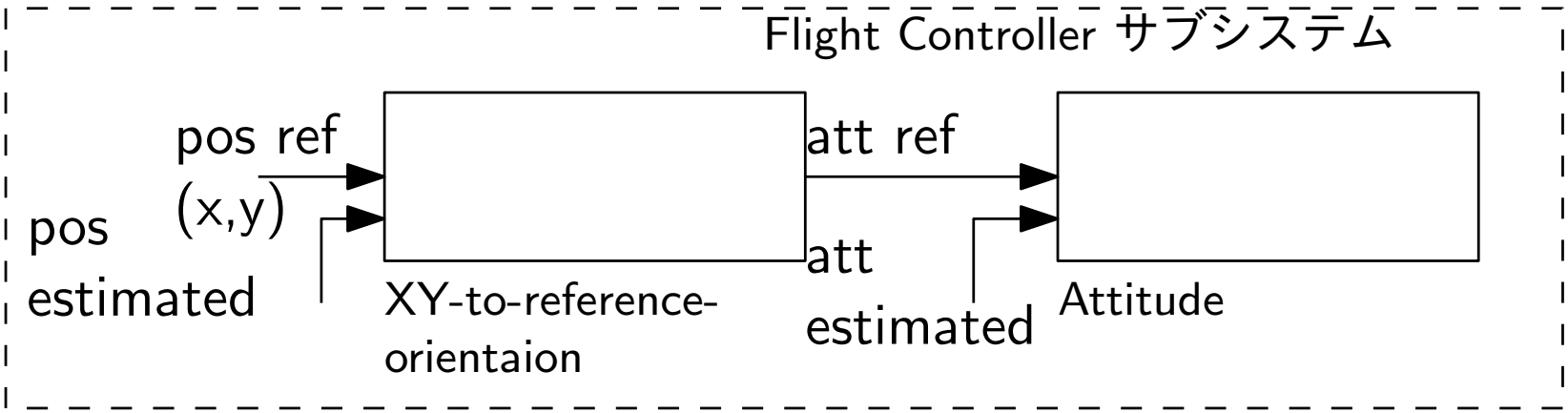
基礎課題２のヒント

* Flight Controller のブロック線図



補足　ロールとピッチの制御

* ロール・ピッチの制御
  カスケード構造（外側と内側に制御器を置いてループを組む）



- XY-to-reference-orientation ブロック
  外側のフィードバックループ
  位置のダイナミクスを制御
- Attitude ブロック
  内側のフィードバックループ
  姿勢のダイナミクスを制御

基礎課題の説明

* ドローンのｚ軸に対する制御器設計法

2-1 “制御する”とは？

2-2 モデルに基づく制御の流れ

2-3 ドローンの仕組み

2-4 ドローンz軸のモデリング

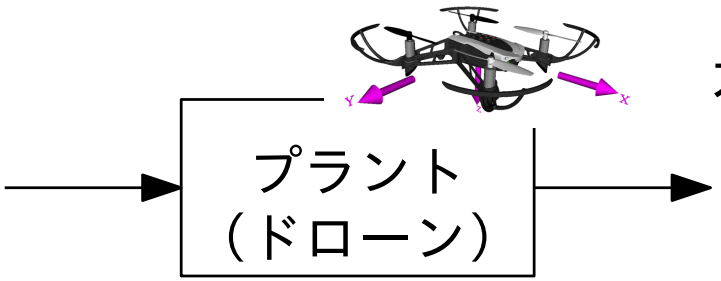
2-5 ドローンz軸の制御器設計

2-6 ドローンのシミュレーション

しくみがわかっている人は発展課題に取り組んでいてかまいません

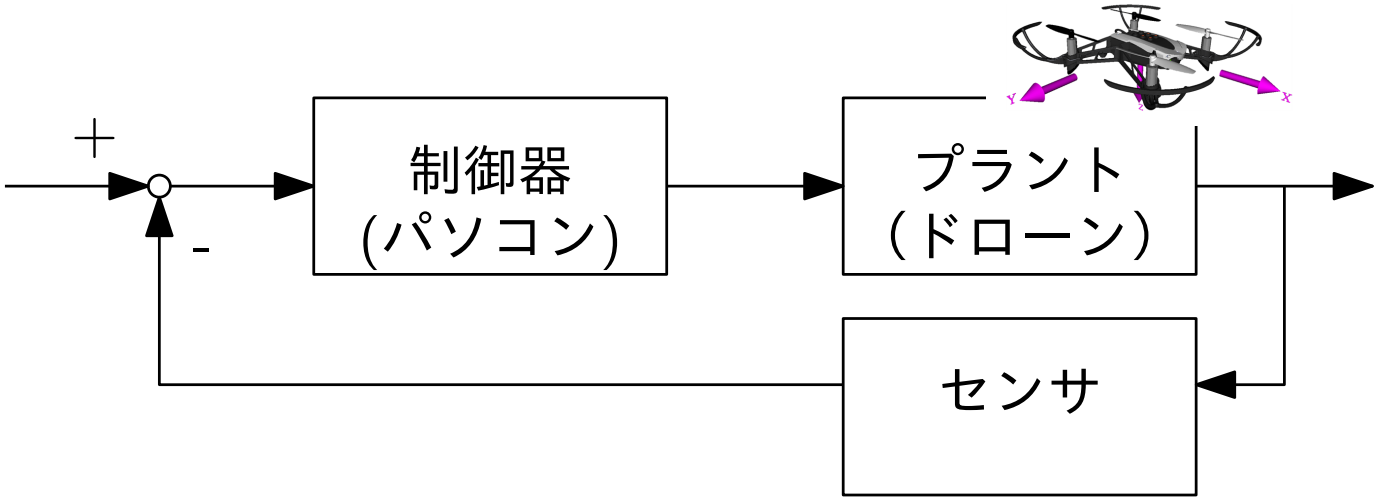
2-1. “制御する”とは？

* もともとのシステム



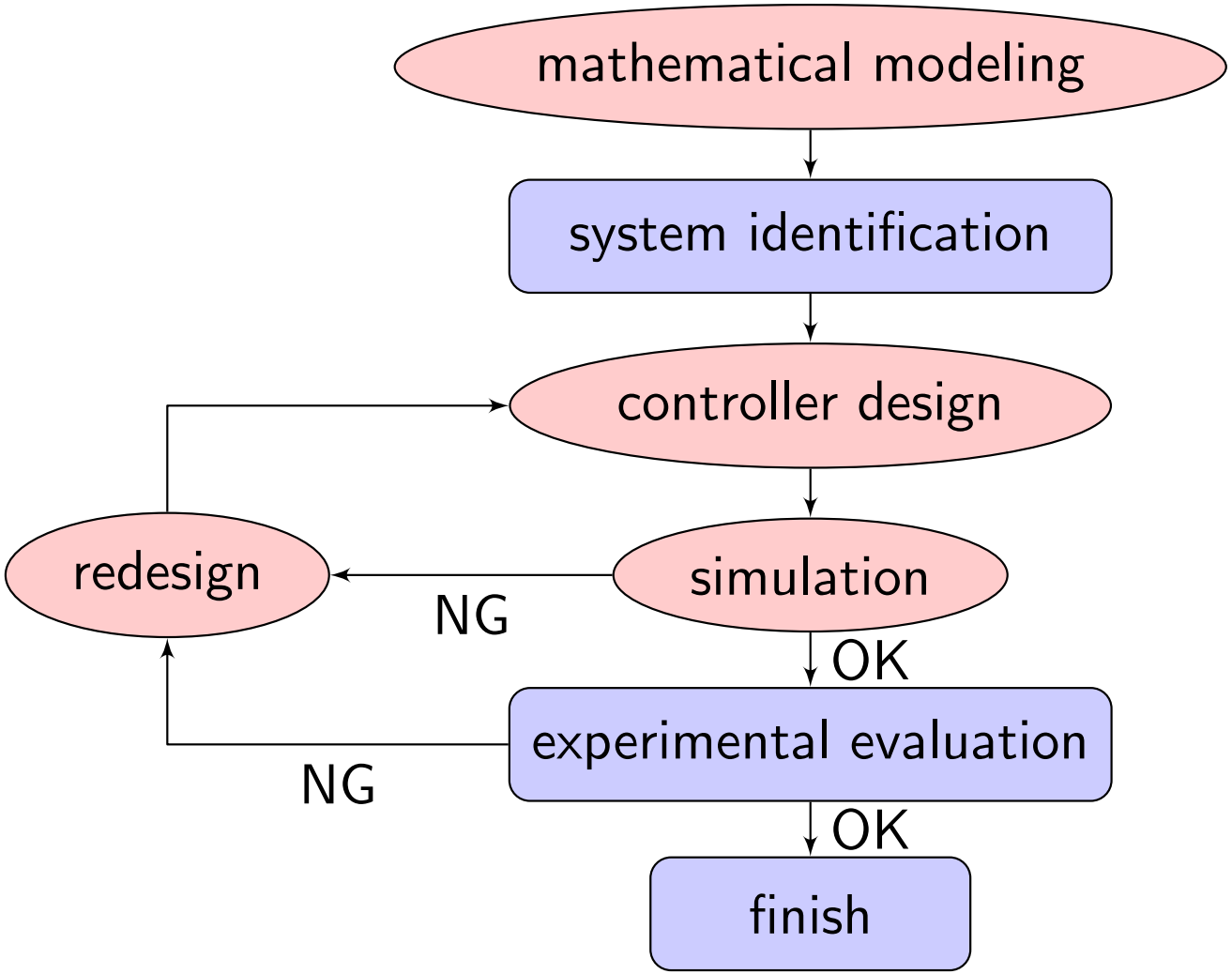
不安定系！

* 閉ループ制御されたシステム



実験では制御器以外はすべて実物

2-2.モデルに基づく”制御”の流れ



赤：パソコン
青：実験

今日はモデリング・制御器設計・シミュレーションに挑戦する

2-3. ドローンの仕組み

* 実験で扱う“ドローン”：クアドコプター
  - ４つのアクチュエータ
  - ６つの自由度

    ⇒underactuation(駆動不足)

    制御が難しい

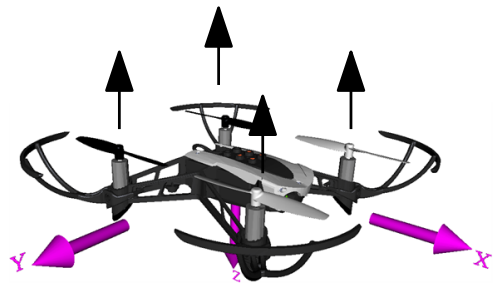
* センサ（制御に必要不可欠）
  - 超音波センサ：地表からの距離
  - カメラ：水平の動きと速度
  - 圧力センサ：高度
  - IMU(慣性観測装置)：加速度・角速度

2-3. ドローンの仕組み

* 推力の発生
  プロペラのモータが回転すると推力発生，浮ける

* 移動
  プロペラの回転数の違いで移動できる

* 回転方向
  向かいあうプロペラは同じ向き，２組のプロペラは違う向き
  ⇒推力・ロール・ピッチ・ヨーを別々に動かせる



まずはz軸に絞って制御をしてみよう！

2-4. ドローンz軸のモデリング

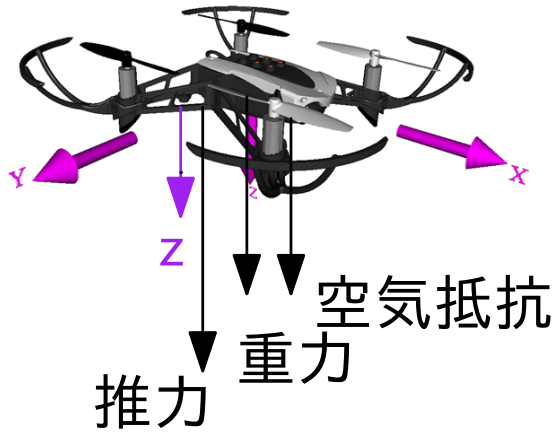
* z軸のプラントの定義
  各プロペラにあるモータの推力は指令値通りになっていると仮定
  ⇒推力からz軸位置まで

外乱がない場合のドローンz軸のプラント

* z 軸のプラントの伝達関数

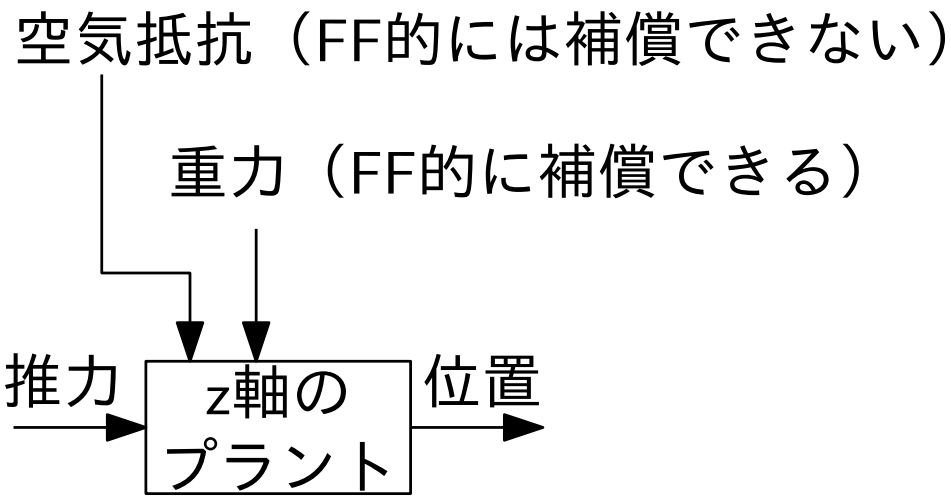
$\frac{z}{T} =$ ⇒P_n（z軸プラントモデル）とする

ヒント：運動方程式
伝達関数は入力から出力までなので,
他の力は含まれない。
従って他の力は外乱となる

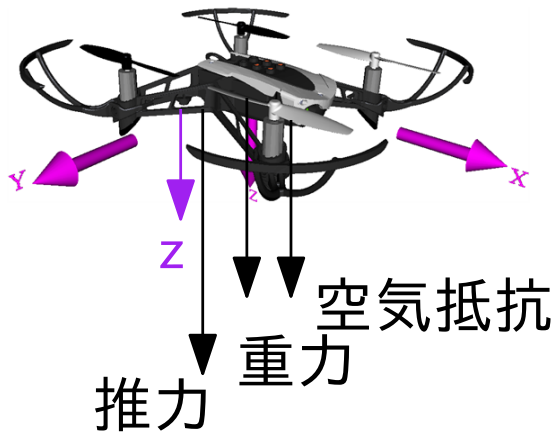


2-5-1. ドローンの重力補償

* 重力補償の意義
  重力があると，PD制御では偏差が残ってしまう
  ⇒あらかじめわかっているのでFF的に補償した力を入力すれば
  重力分は補償できる

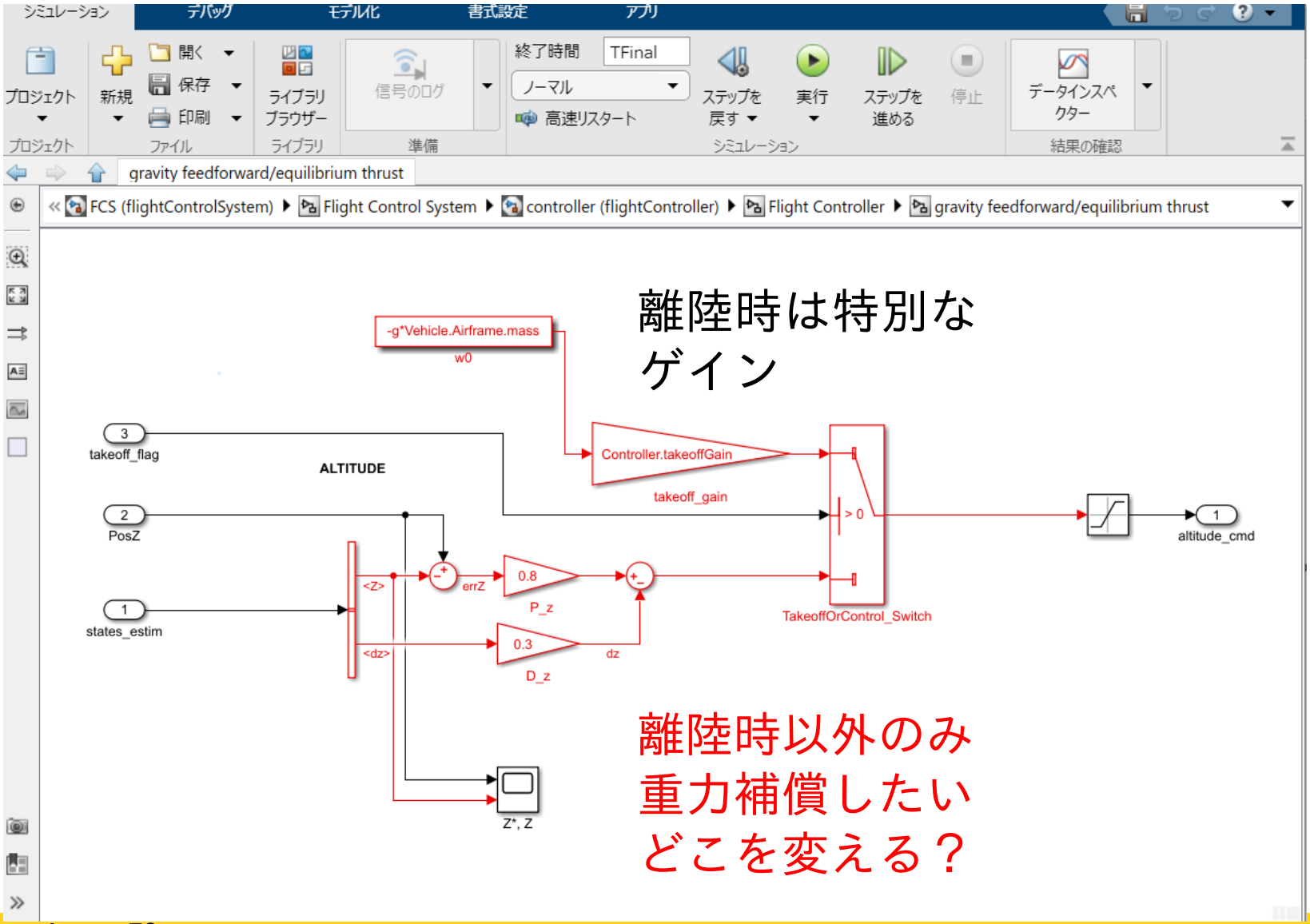


実際のドローンz軸のプラント



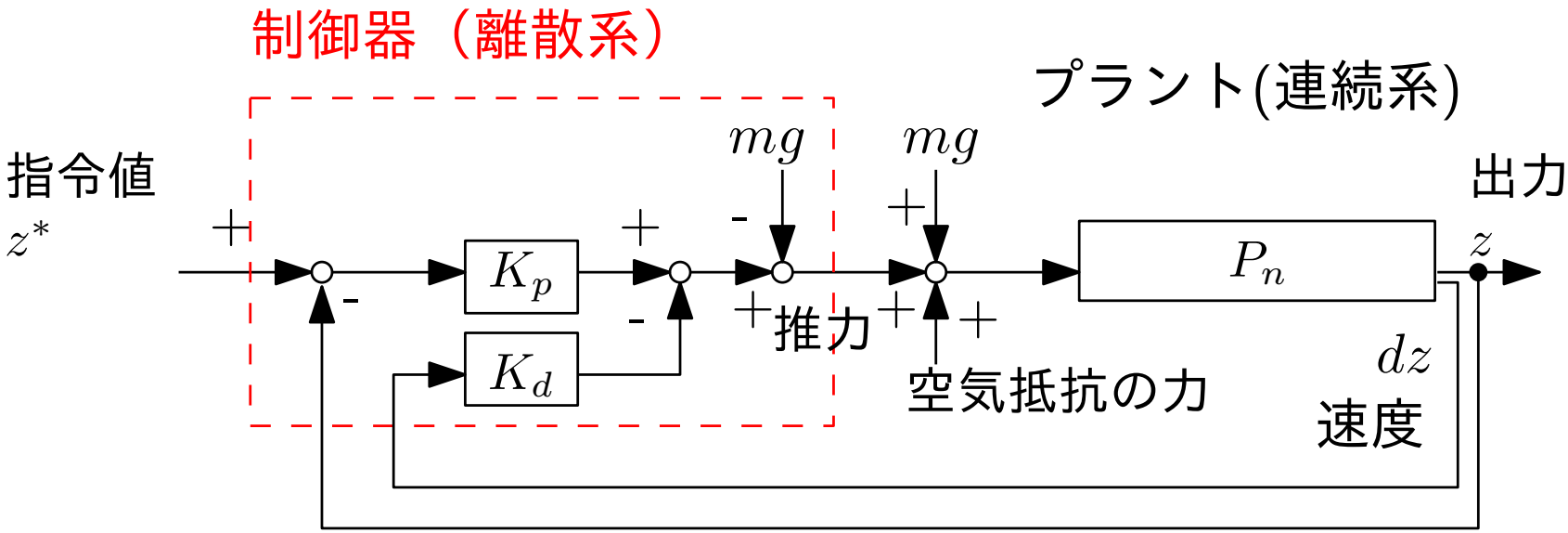
2-5-1. ドローンの重力補償

* z 軸の制御器ブロック



2-5-2. ドローンz軸のPID制御器設計

* z軸のPD制御系のブロック線図



注意点：微分キック（指令値の急変による悪影響）を防ぐために
微分先行型PDになっている

* PD制御の問題点

空気抵抗があるとステップ指令値に追従できない（偏差が残る）

⇒PID制御にしたい

2-5-2. ドローンz軸のPID制御器設計

* 「動く」PID制御器を作るには

  step1: PID制御器を設計する（微分先行型で）
  step2: アンチワインドアップをする

2-5-2. ドローンz軸のPID制御器設計

* z軸の閉ループ伝達関数

$\frac{z}{z^*} =$

導出：

プラントモデルを P_n とすると，

$z = P_n(s)u$

$u = (K_p + K_i/s)(z^* - z) - K_d s z$

が立てられる。

ここから$\frac{z}{z^*}$を求めて，$P_n = \frac{z}{T}$を代入

* z軸PID制御器の極配置設計

極が -2 rad/s になるようにK_p,K_i,K_dの値を求める

ヒント：極は（閉ループ伝達関数の分母）$= 0$の解である

$$m = \text{Vehicle.Airframe.mass} = 0.063$$

2-5-2. ドローンz軸のPID制御器設計

* 離散化

このシミュレーションではソルバーが離散なので，離散ブロックを使う

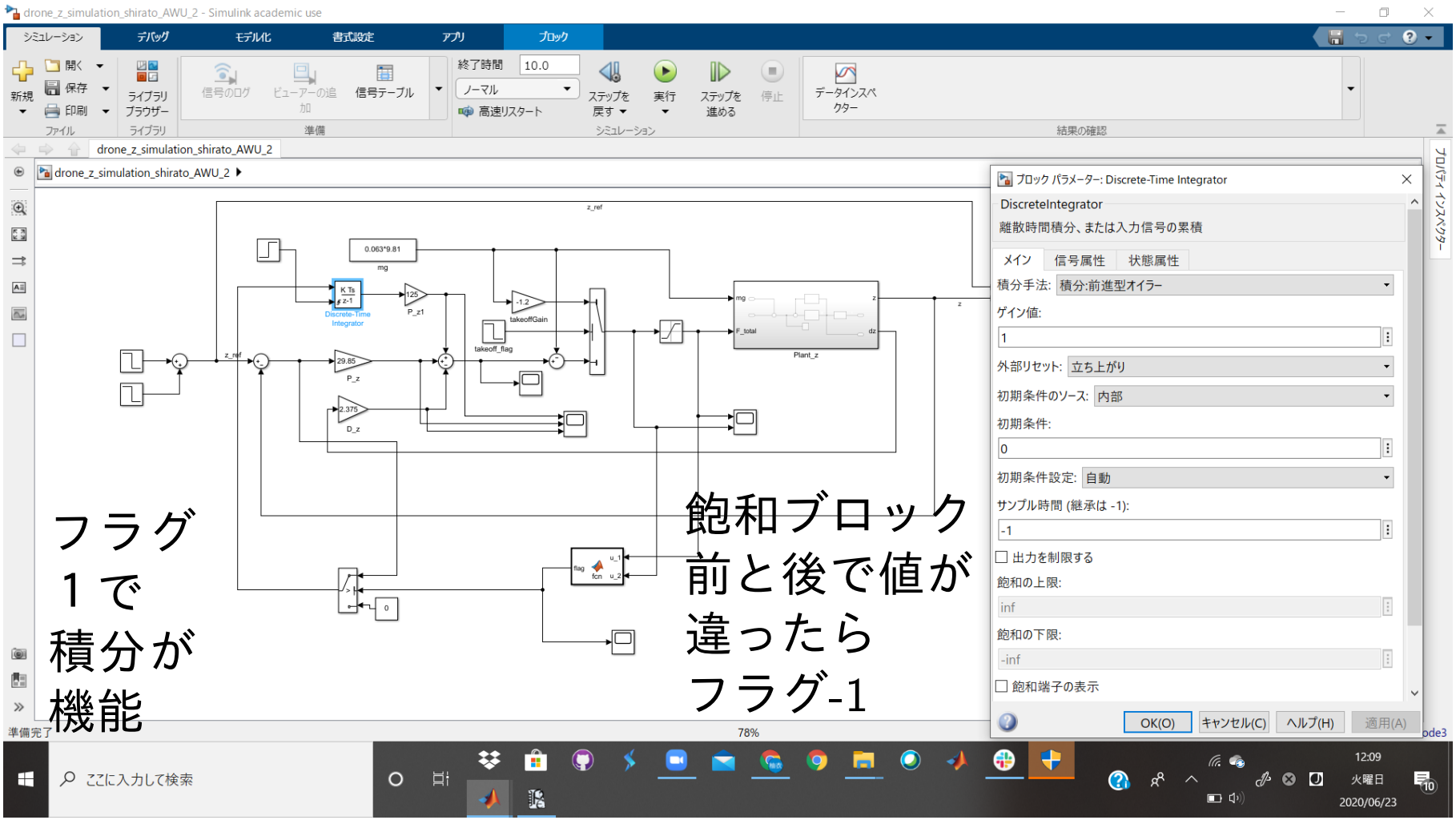
本物のドローン相手に実装する場合も離散化して実装する

* 制御器の実装

1. 連続の伝達関数でKp,Ki,Kdのゲインを求める
2. Simulink の Gain ブロックにその値を入れる
3. 積分はdiscrete-time integrator のブロックを使う

2-5-2. ドローンz軸のPID制御器設計

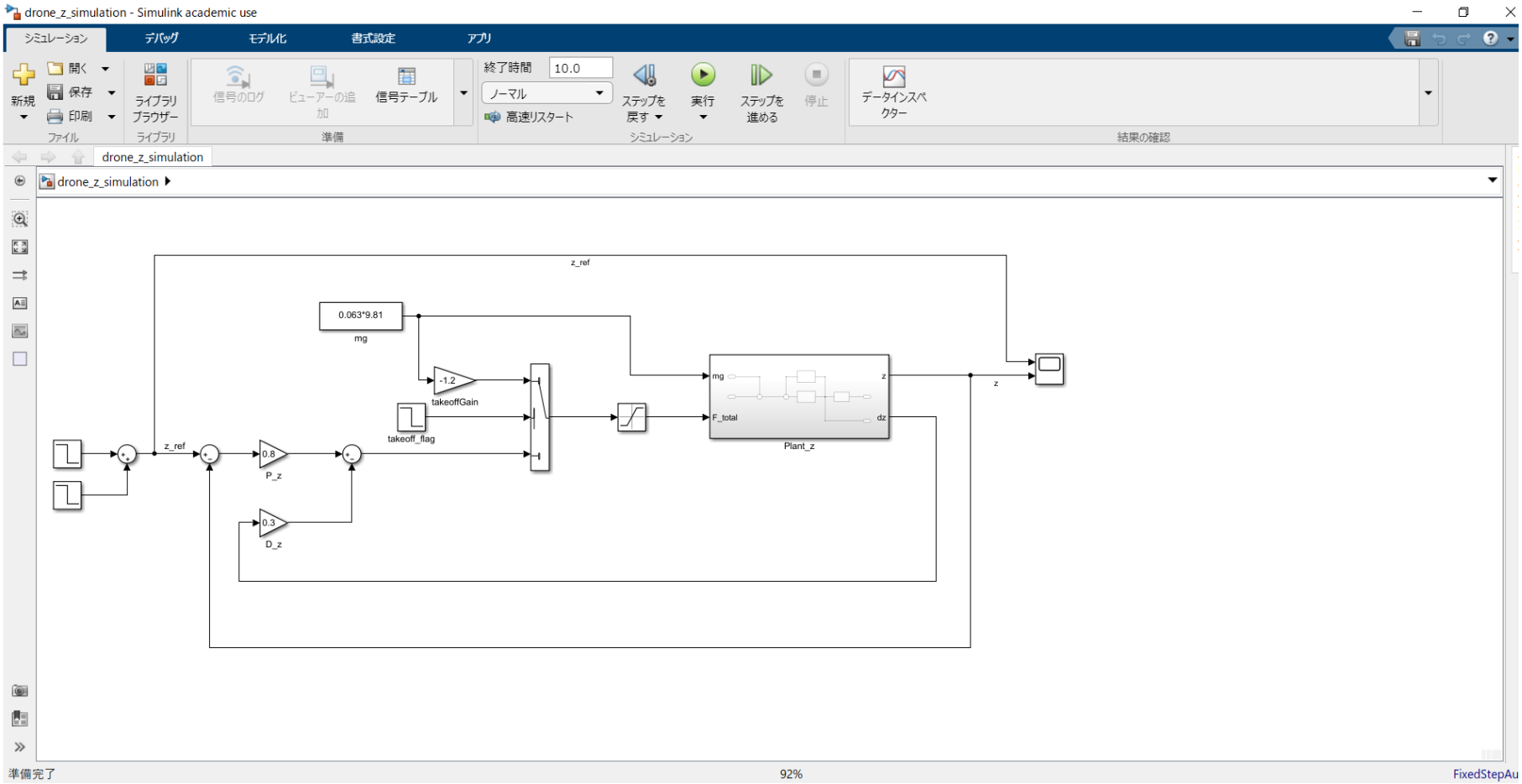
* アンチワインドアップの設計



2-6. ドローンのシミュレーション

* z軸のシミュレーションに使うプロジェクトファイル：

  `drone_z_simulation.slx`



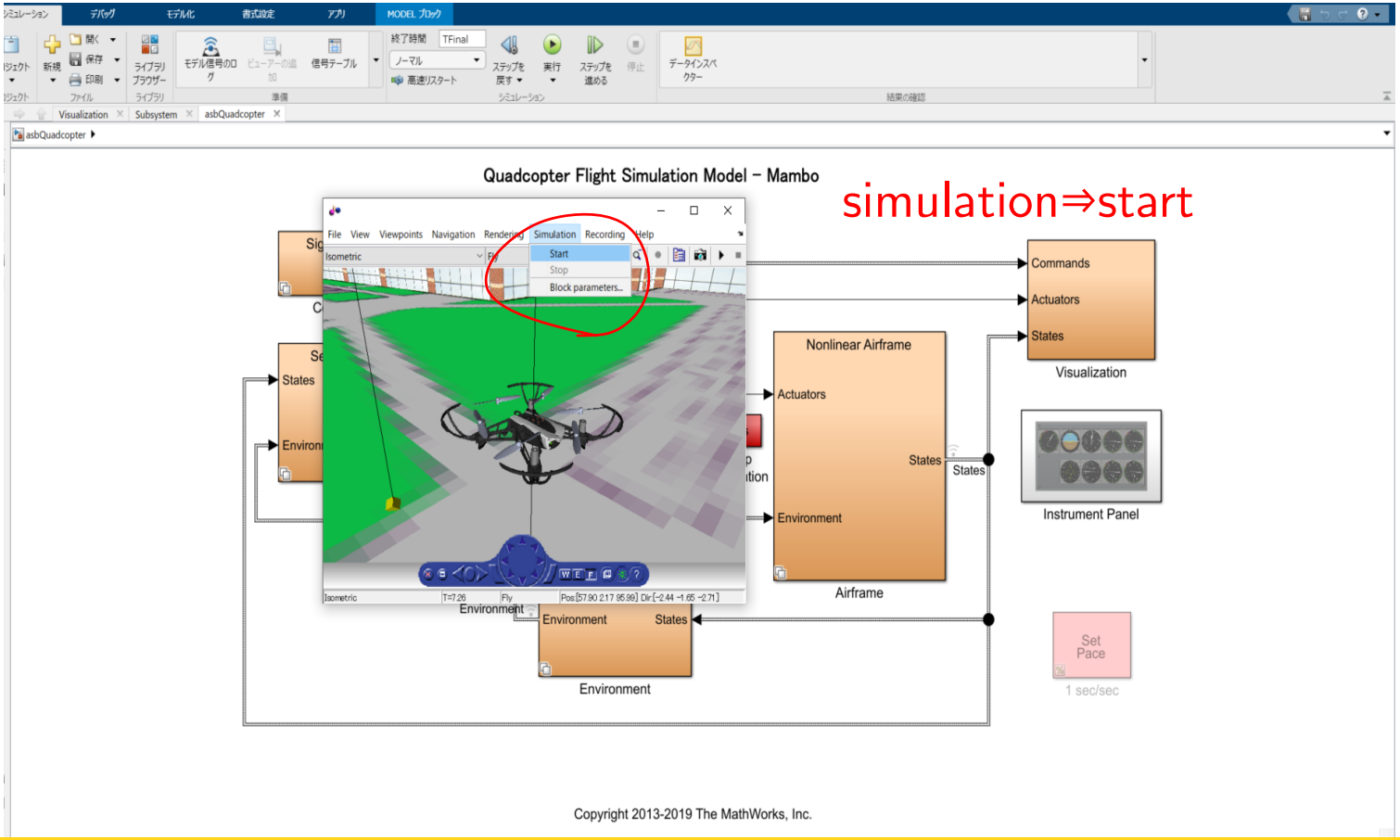
2-6. ドローンのシミュレーション

* 全体のシミュレーションに使うプロジェクトファイル：
  Mathworks社のデモ `asbQuadcopter` をTAが改造した
  `asbQuadcopter_exp.prj`

* プロジェクトファイルの開き方
  - `asbQuadcopter_exp.zip` を解凍
  - `asbQuadcopter.prj` をMATLAB2020で立ち上げる
  - `asbQuadcopter.prj` をMATLAB内でクリック

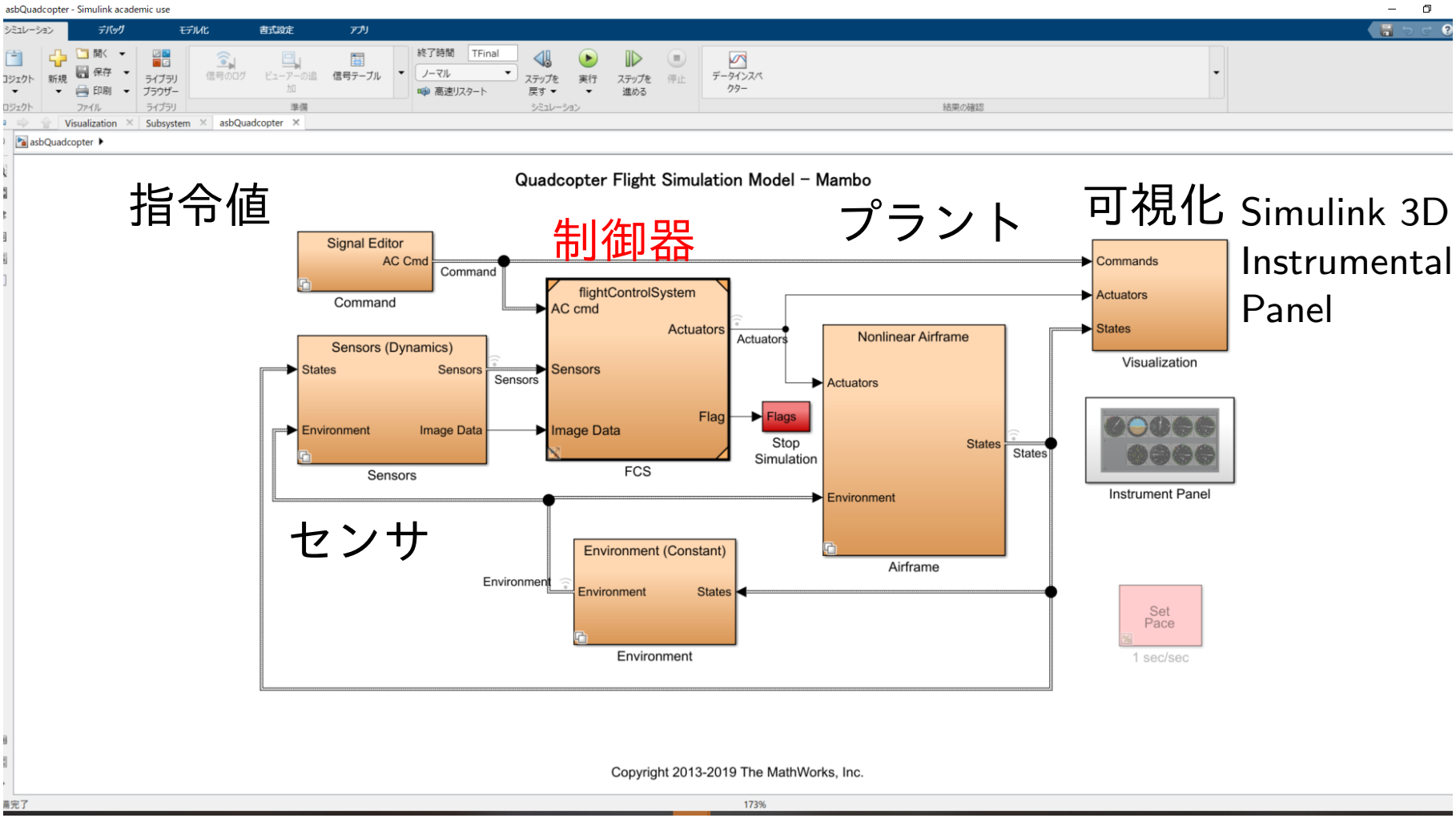
2-6. ドローンのシミュレーション

* 設計した制御器がまともなものか検証
  z 軸の制御器を変更して回してみる



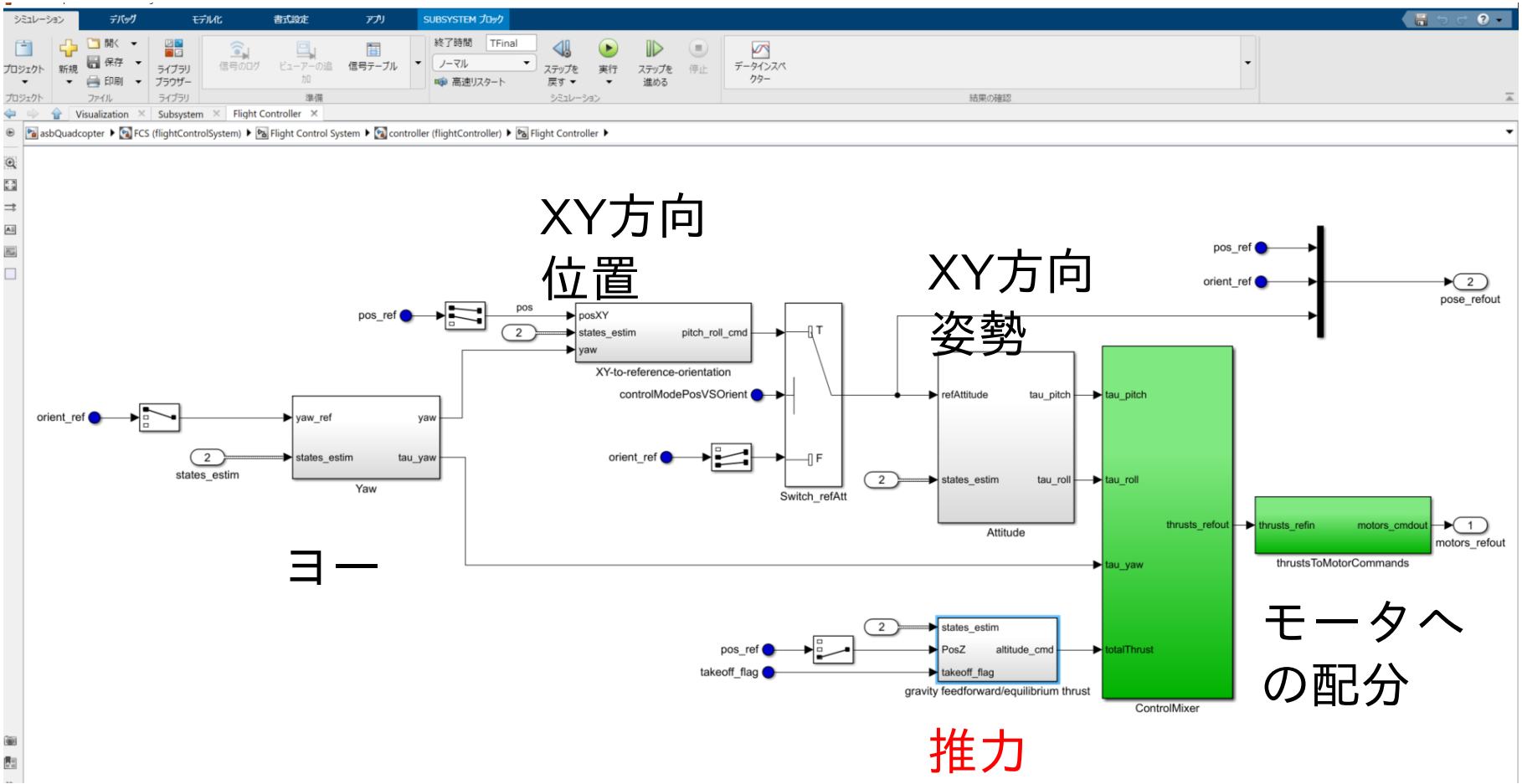
2-6補足　デモファイルの説明

* 全体のブロック線図



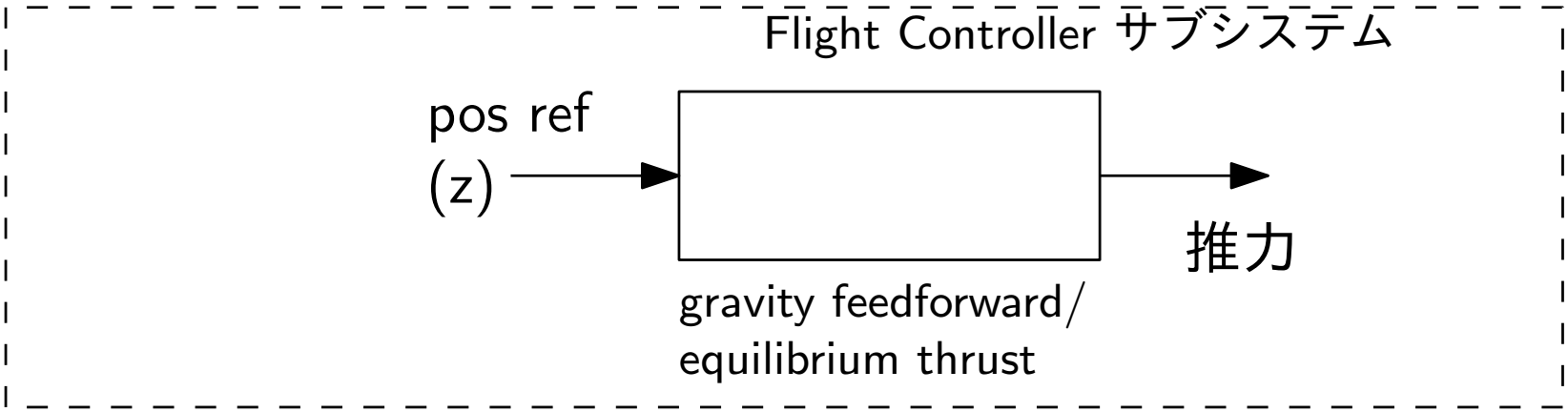
2-6補足　デモファイルの説明

* Flight Controller のブロック線図



2-6補足　デモファイル制御器の説明

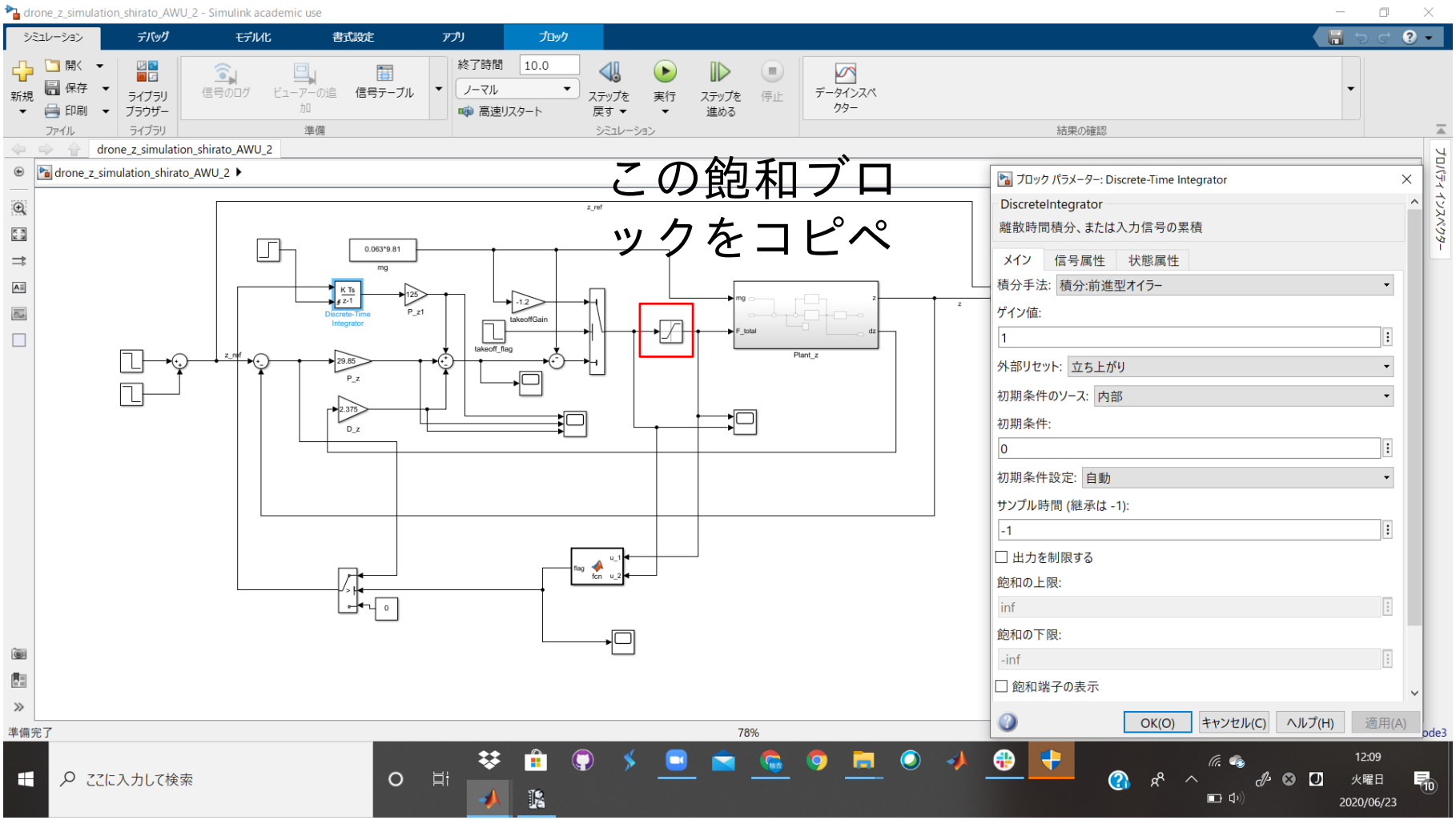
* 推力の制御



- gravity feedforward/equilibrium thrust ブロック
  ドローンが浮き上がるための総推力を制御
  中はPD制御

2-6. 補足　飽和ブロックの値

* 全体のシミュレーション制御器部分で飽和ブロックの変更



z軸だけのスクショ

発展課題1/2

以下の課題から１つ選んで発表してください。
自由に作っていただいても構いません。

- z 軸モデルでの定常誤差をなくす制御
  - （重力補償　標準課題）
  - （PID制御　標準課題）⇒Anti-windup制御
  - 外乱オブザーバ
- 速い応答をめざす制御
  - FF追加

発展課題2/2

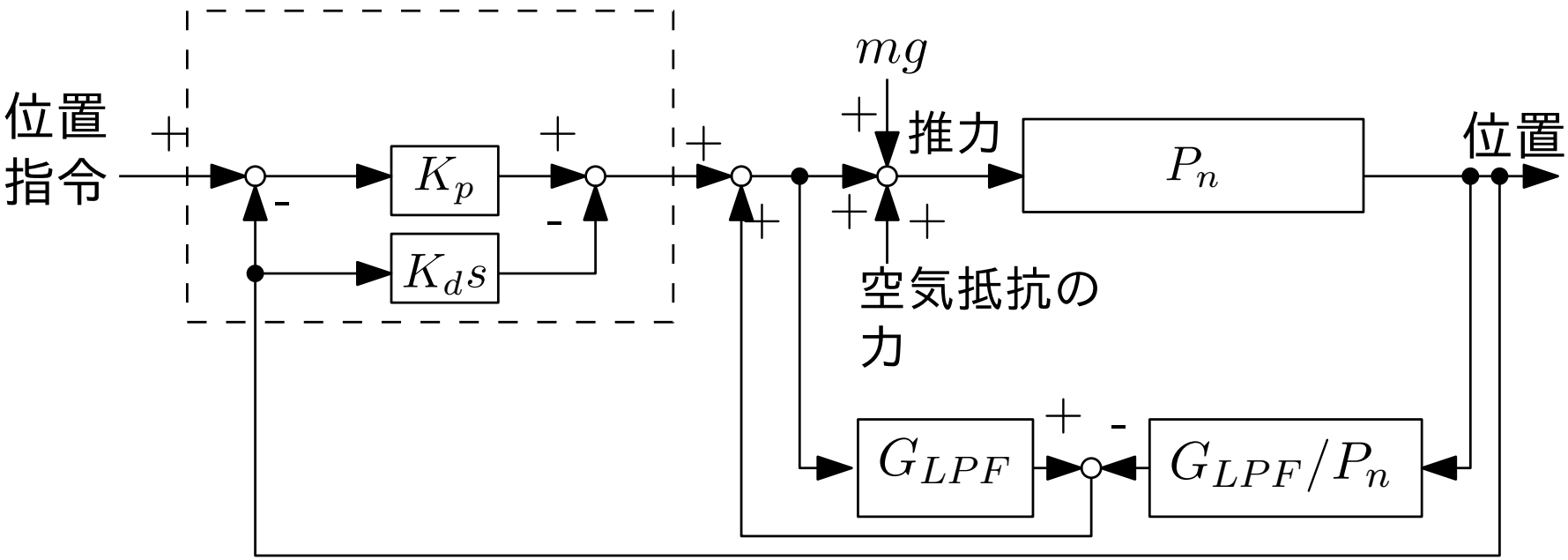
- 今あるドローンで楽しむ制御
  - くるくる回りながら上昇できるようにする
  - 他の変数（ヨー・ピッチ）の制御
  - x軸方向に進めるようにする
  - 力制御する

- ドローンの改造により広がる制御
  - ヘキサコプターへの改造
  - Fault tolerance 制御

Appendix
発展課題のヒント

z軸の外乱オブザーバ

* 外乱オブザーバのブロック線図（位置を使う）



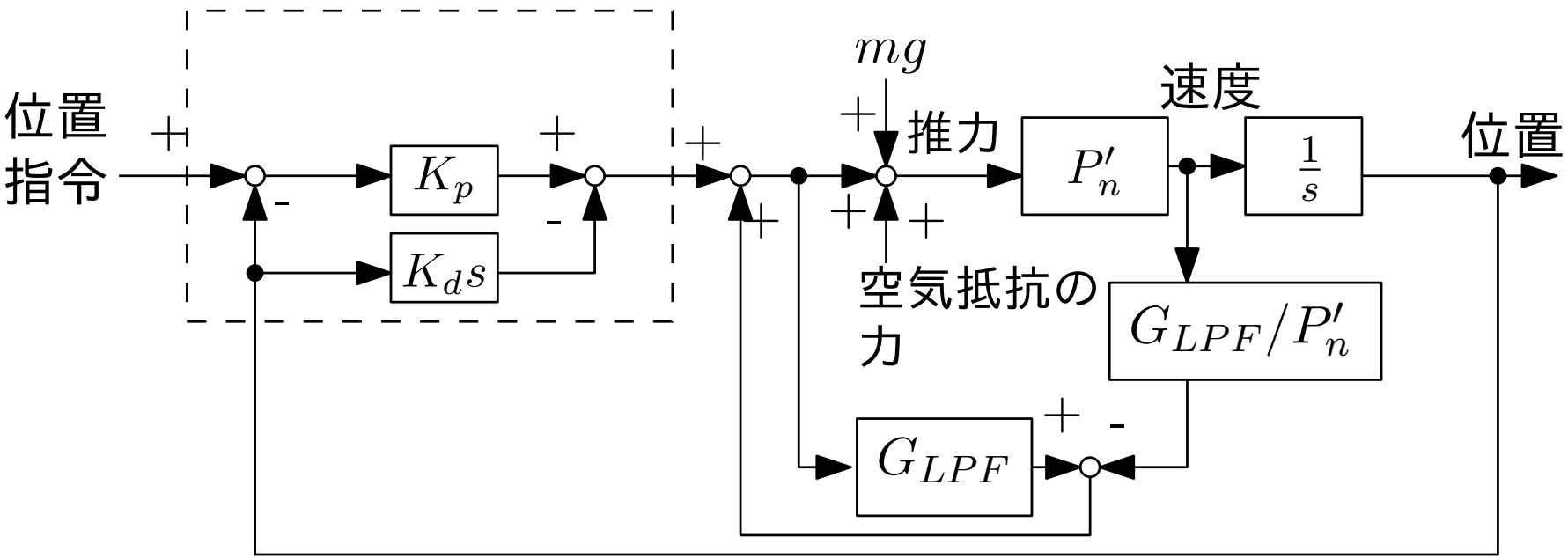
P_n： z軸プラントモデル（伝達関数）
G_{LPF}：外乱オブザーバ用のローパスフィルタ

* 課題

G_{LPF}を設計して外乱オブザーバで空気抵抗を抑圧しよう

z軸の外乱オブザーバ

* 外乱オブザーバのブロック線図（速度を使う）



P_n： z軸プラントモデル（伝達関数）
G_{LPF}：外乱オブザーバ用のローパスフィルタ

* 課題

G_{LPF}を設計して外乱オブザーバで空気抵抗を抑圧しよう

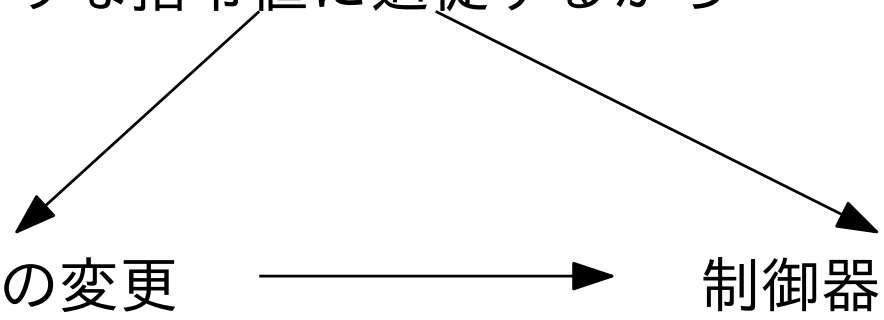
FFの追加

* FB制御の問題
  FB帯域より速い信号には追従できない
  ⇒FF制御を組み合わせて
  速い信号へのそこそこの追従をめざす

詳しくはTA吉清作成スライド：
（https://drive.google.com/drive/folders/1Q3xKDjV-_KqI4Vxdsce1mjbAch3q8Vcu?usp=sharing内）
を参照してください

回るドローン

* なぜ回るか
  回るような指令値に追従するから



徐々に回り始めたい

指令値を変えたので
そのままでは追従できない

定常偏差がない条件（最終値の定理）
ステップ：　制御器かプラントが$\frac{1}{s}$を持つこと
ランプ：　制御器かプラントが$\frac{1}{s^2}$を持つこと

回るドローン

* 注意点
ソルバーが離散なので，step など連続系の指令値を使えない

* 指令値の変え方
1. ホーム/環境/基本設定/Matlab/一般/MATファイルからVersion 7.3を選択[1]
2. 必要ならMATLAB2020を開き直す
3. mainModel中のreference.m を変える
4. MATLAB 2020aのウィンドウで，フォルダをmainModelsに変更
5. その階層でreference.mを回す（変更された値がref.matに入る）
6. asbQuadcopter.prjをクリック

[1] `https://jp.mathworks.com/help/matlab/import_export/mat-file-versions.html`

回るドローン

* 注意点

ソルバーが離散なので，連続系で制御器を作れない

* 制御器の実装

1. 連続の伝達関数を求める
2. sys = tf([ ], []); c2d(sys, 0.005, ‘tustin’)で変換
3. Discrete Transfer Fcnで実装

他の変数の制御

* ロール・ピッチ・ヨーの制御器設計

  推力の制御と同様に
  モデリング⇒制御器設計⇒シミュレーションしてみる

  詳しくはTA横田作成スライド：
  （https://drive.google.com/drive/folders/1Q3xKDjV-_KqI4Vxdsce1mjbAch3q8Vcu?usp=sharing内）
  を参照してください

モータが壊れたら？

* 故障に対しても丈夫な制御法

1. 冗長性を増やす（ヘキサコプターにするなど）
2. Fault tolerant control
   - 故障の検出
   - 故障状態の推定 (カルマンフィルタなど)
   - 故障状態の制御

ヘキサコプター（故障に強い制御）

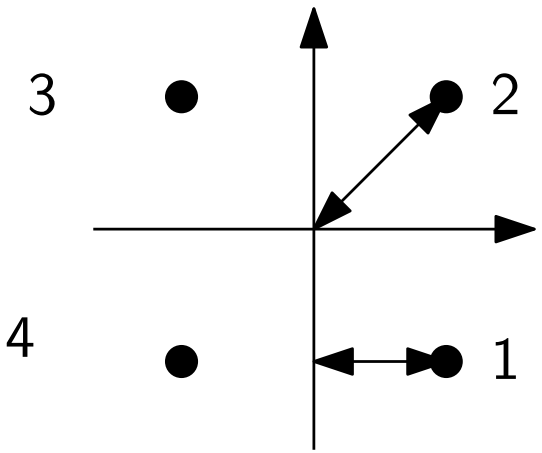
* 変えるもの
  - 羽の座標
  - トルクを各モータへ分配する行列のサイズ
  - トルクを各モータへ分配する行列の要素（ヨートルクなど）
  - その他の変更
  - Simulink 3D での羽の増やし方

ヘキサコプター（故障に強い制御）

* 羽の位置
  Airframe⇒Nonlinear Airframe⇒Nonlinear⇒AC Model⇒Motor Forces and Torques
  D1-D4 で指定
  Vehicle.Airframeの変数はtasksフォルダのvehicleVars.mにある

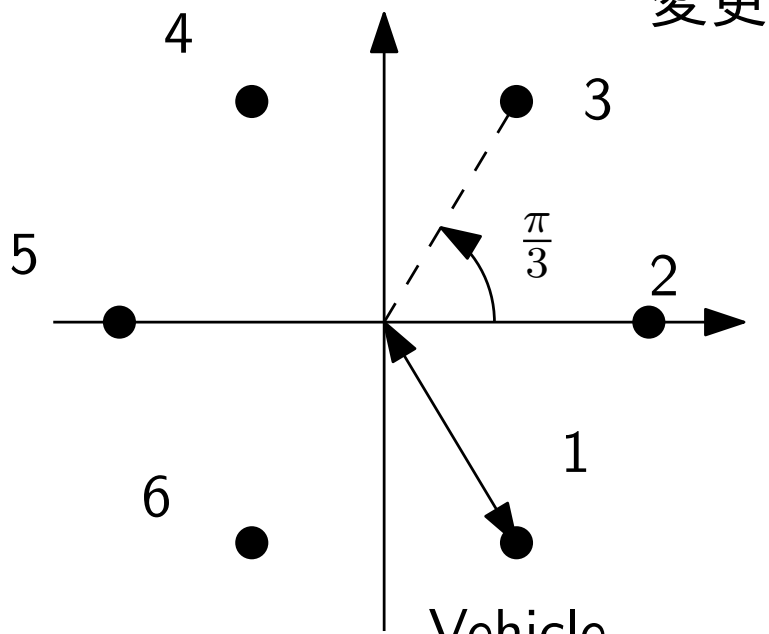
もともと

Vehicle.
Airframe.d



Vehicle.
Airframe.xy

変更後



Vehicle.
Airframe.d

ヘキサコプター（故障に強い制御）

* トルクの分配
  Flight controller⇒controlMixer
  Controller.Q2Ts

行列を使って分配する

$$\begin{pmatrix} Tz \\ \tau_{yaw} \\ \tau_{pitch} \\ \tau_{roll} \end{pmatrix} \longrightarrow \begin{pmatrix} f1 \\ f2 \\ f3 \\ f4 \end{pmatrix}$$

- 総推力
- ヨーのトルク
- ピッチのトルク
- ロールのトルク

モータが出すべき力

ヘキサコプター（故障に強い制御）

* トルクの行列と各モータの力の関係式（もともと）

$$\begin{pmatrix} Tz \\ \tau_{yaw} \\ \tau_{pitch} \\ \tau_{roll} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ d & -d & d & -d \\ -l & -l & l & l \\ -l & l & l & -l \end{pmatrix} \begin{pmatrix} f1 \\ f2 \\ f3 \\ f4 \end{pmatrix}$$

トルクの行列　　$\downarrow A$　　モータの力

* トルク行列を達成するために各モータが出すべき力

$$\begin{pmatrix} f1 \\ f2 \\ f3 \\ f4 \end{pmatrix} = A^{-1} \begin{pmatrix} Tz \\ \tau_{yaw} \\ \tau_{pitch} \\ \tau_{roll} \end{pmatrix}$$

A^{-1}(Controller.Q2Ts)
tasksフォルダのControllerVars.mで指定

ヘキサコプター（故障に強い制御）

* トルクの行列と各モータの力の関係式（羽６枚）

$$\begin{pmatrix} Tz \\ \tau_{yaw} \\ \tau_{pitch} \\ \tau_{roll} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ d & -d & d & -d & d & -d \\ -l_1 & -l_2 & -l_1 & l_1 & l_2 & l_1 \\ -l_3 & 0 & l_3 & l_3 & 0 & -l_3 \end{pmatrix} \begin{pmatrix} f1 \\ f2 \\ f3 \\ f4 \\ f5 \\ f6 \end{pmatrix}$$

トルクの行列　　A　　モータの力

* トルク行列を達成するために各モータが出すべき力

$$\begin{pmatrix} f1 \\ f2 \\ f3 \\ f4 \\ f5 \\ f6 \end{pmatrix} = pinv(A) \begin{pmatrix} Tz \\ \tau_{yaw} \\ \tau_{pitch} \\ \tau_{roll} \end{pmatrix}$$

6×1　　6×4　　4×1

ControllerVars.mで
疑似逆行列を作る

ヘキサコプター（故障に強い制御）

* Aの作り方（羽６枚）

$$\begin{pmatrix} Tz \\ \tau_{yaw} \\ \tau_{pitch} \\ \tau_{roll} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ d & -d & d & -d & d & -d \\ -l_1 & -l_2 & -l_1 & l_1 & l_2 & l_1 \\ -l_3 & 0 & l_3 & l_3 & 0 & -l_3 \end{pmatrix} \begin{pmatrix} f1 \\ f2 \\ f3 \\ f4 \\ f5 \\ f6 \end{pmatrix}$$

トルクの行列　　A

総推力　　1
ヨーのトルク　回転の向きに応じた符号　値は同じ
ピッチのトルク　y軸からの符号つき距離
ロールのトルク　x軸からの符号つき距離

「羽の位置」と合わせる

ヘキサコプター（故障に強い制御）

* 出力ポートの次元
  - Flight Control System の出力ポート
  - Nonlinear Airframe の入力ポート

* モータの数
  - Nonlinear Airframe⇒Nonlinear⇒AC Model⇒Motor Forces and Torques ⇒MotorsToW
    ゲインの要素数を追加
  - Flight Controller⇒thrust to Motor Commands
    要素数を追加

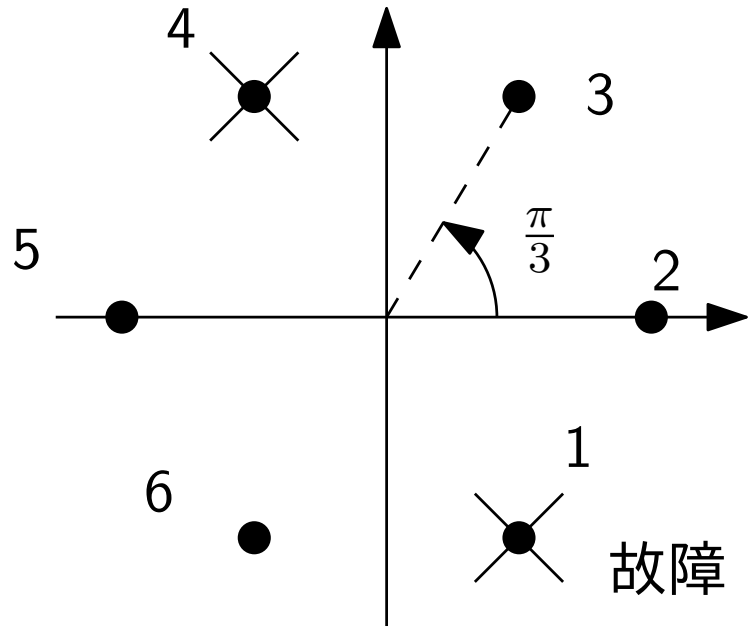
ヘキサコプター（故障に強い制御）

* SImulink 3D での羽の増やし方・座標の変え方
* 該当するブロック：VR Sink
  見せるための部分。シミュレーションには影響しない。
* 変更の仕方
  - VR Sink をクリック
  - File⇒Open in Editor
  - Quadbody⇒Children
  - Childrenをコピー
  - 座標にあたる translation を変える(x,z,yの順？)

Fault Tolerance 制御（故障に強い制御）

* Fault tolerance 制御
  - 故障の検出
  - 故障状態の推定 (カルマンフィルタなど)
  - 故障状態の制御

* 方針
  - １つのモータの故障を想定
  - 通常時　６つのモータで動かす
  - 故障時　４つのモータで動かす



Fault Tolerance 制御（故障に強い制御）

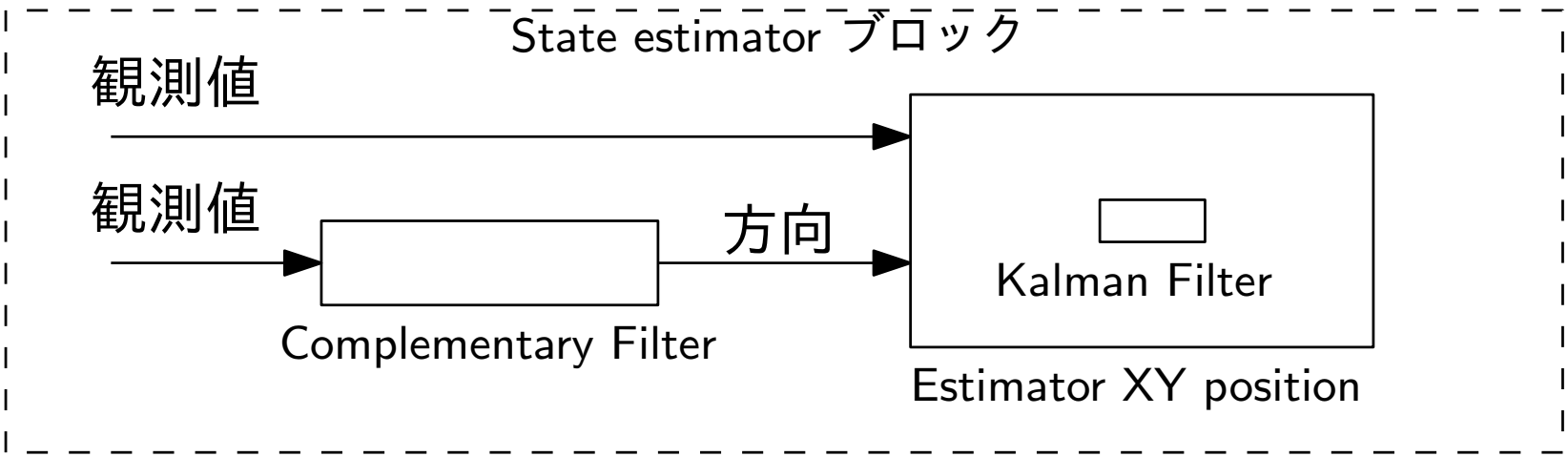
* 故障時の制御
  動いている４つのモータに対して推力配分

* 切り替えに使えるブロック
  - switch cases サブシステム
  - switch ブロック
  - 定数ブロック（フラグとして利用）

* 注意点
  - 可変サイズに対処できないブロックがある
  - 入力端子を増やして対応
  - switchブロックはクリックして「異なるデータの入力サイズを許可する」にチェック
  - 全体として重くなりがち

Fault Tolerance 制御（故障に強い制御）

* 推定器の構造
　全状態変数の値を推定したい
⇒ハイブリッド・フィルタ構成



- Complementary Filter
  方向（orientaion）を推定
- Estimator XY position
  状態変数の値を推定