robbit-esp システムマニュアル

吉瀬研究室
制作日：2025年9月

概要

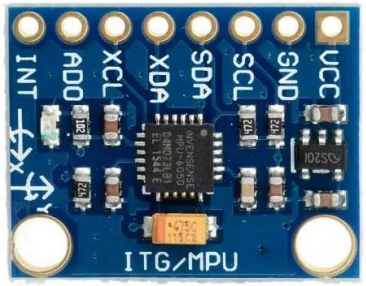
- このプログラムはrobbit-espのハードウェア，ソフトウェアの内容を説明している

- この資料を参考にすることで，robbit-espのシステムの理解が容易になり，性能改善を行いやすくなる

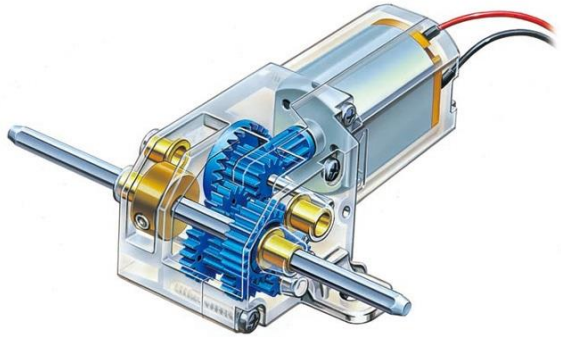
robbit-esp : ハードウェア

- マイコンボード: XIAO ESP32-C3
- センサ: MPU-6050(IMU)
  - 加速度，角速度(ジャイロ)センサ
- ギアモータ: ミニモータ標準ギヤボックス(TAMIYA)

Cmod A7-35T



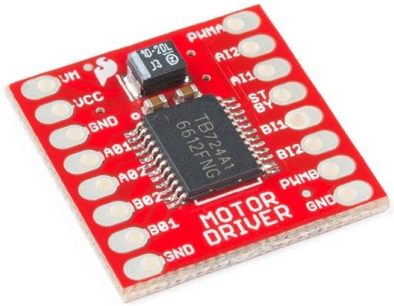
MPU-6050



ミニモータ標準ギヤボックス(TAMIYA)

robbit-esp : ハードウェア

- モータドライバ: TB6612FNG搭載 モータドライバ
- タイヤ:スリムタイヤセット(55mm 径) 70193
- バッテリー: EEMB 653042(3.7V)



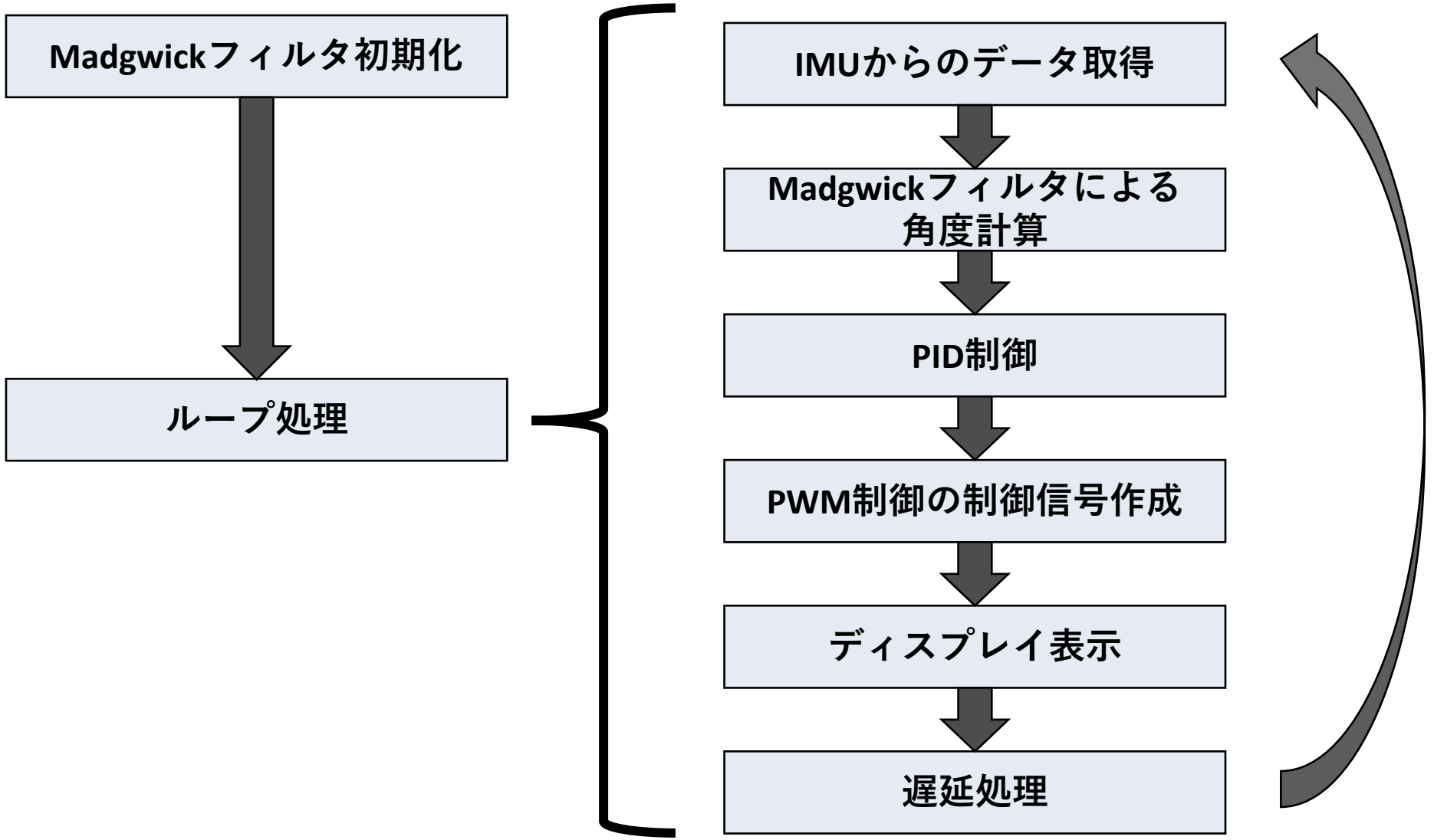
TB6612FNG搭載
モータドライバ

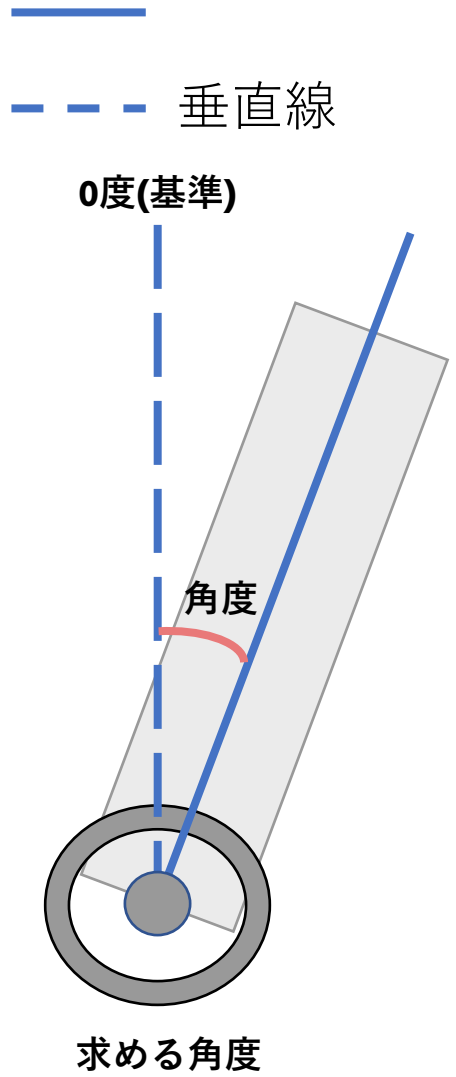
スリムタイヤセット

EEMB 653042
充電式バッテリー

ソフトウェア: 概要

ソフトウェア処理はmain.cppに書いてある。いかにフローチャートを示す。





ソフトウェア: IMUからのデータ取得

- IMUから3軸の加速度と角速度を取得する
- IMUとの通信には既存のライブラリを用いる

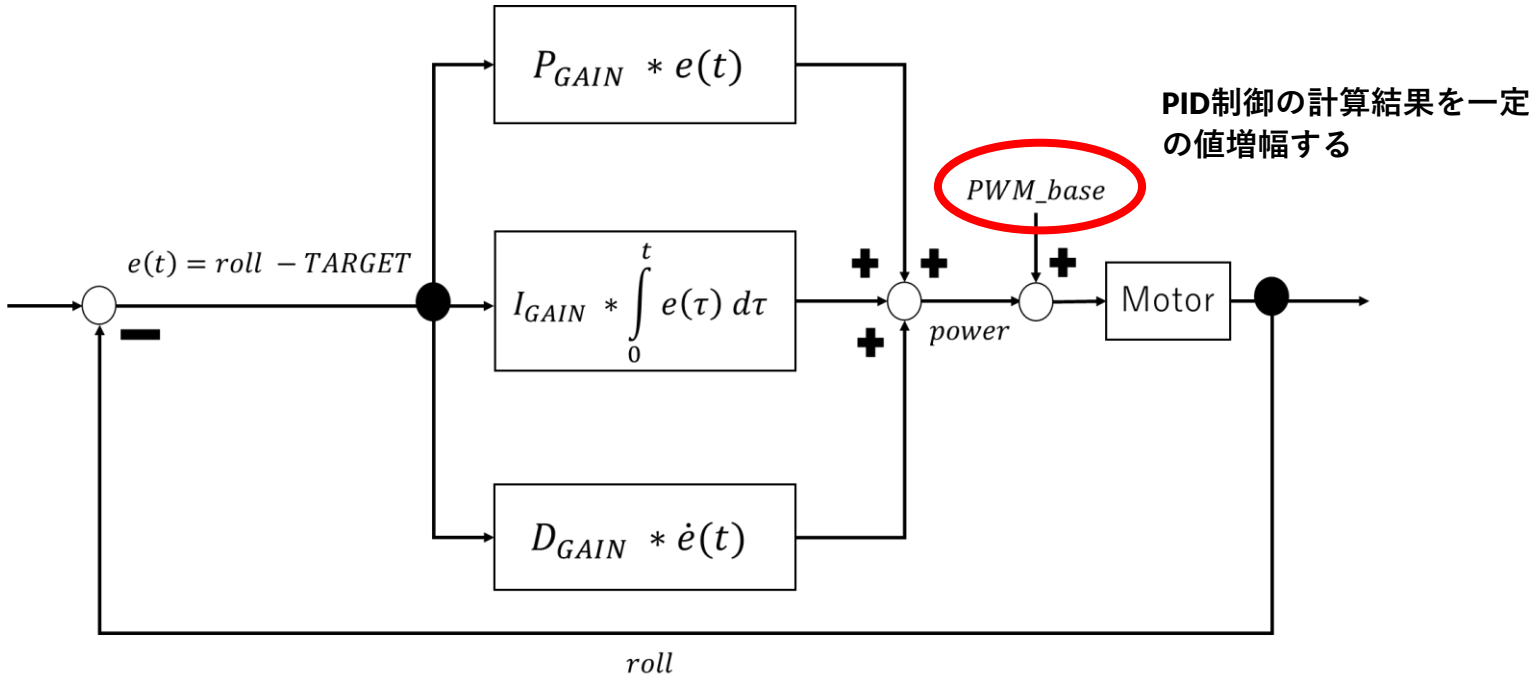
- 3軸の加速度と角速度とMadgwickフィルタを利用することで角度を算出する
- Madgwickフィルタのライブラリにゲインの調整機能を追加して実装する

```
mpu.getMotion6(&ay, &ax, &az, &gy, &gx, &gz);
MadgwickF.updateIMU(gz/131.0, gy/131.0, gx/131.0, az/16384.0, ay/16384.0, ax/16384.0);
roll = MadgwickF.getRoll() - 90.0; // Note
```

ソフトウェア: PID制御

■ PID制御

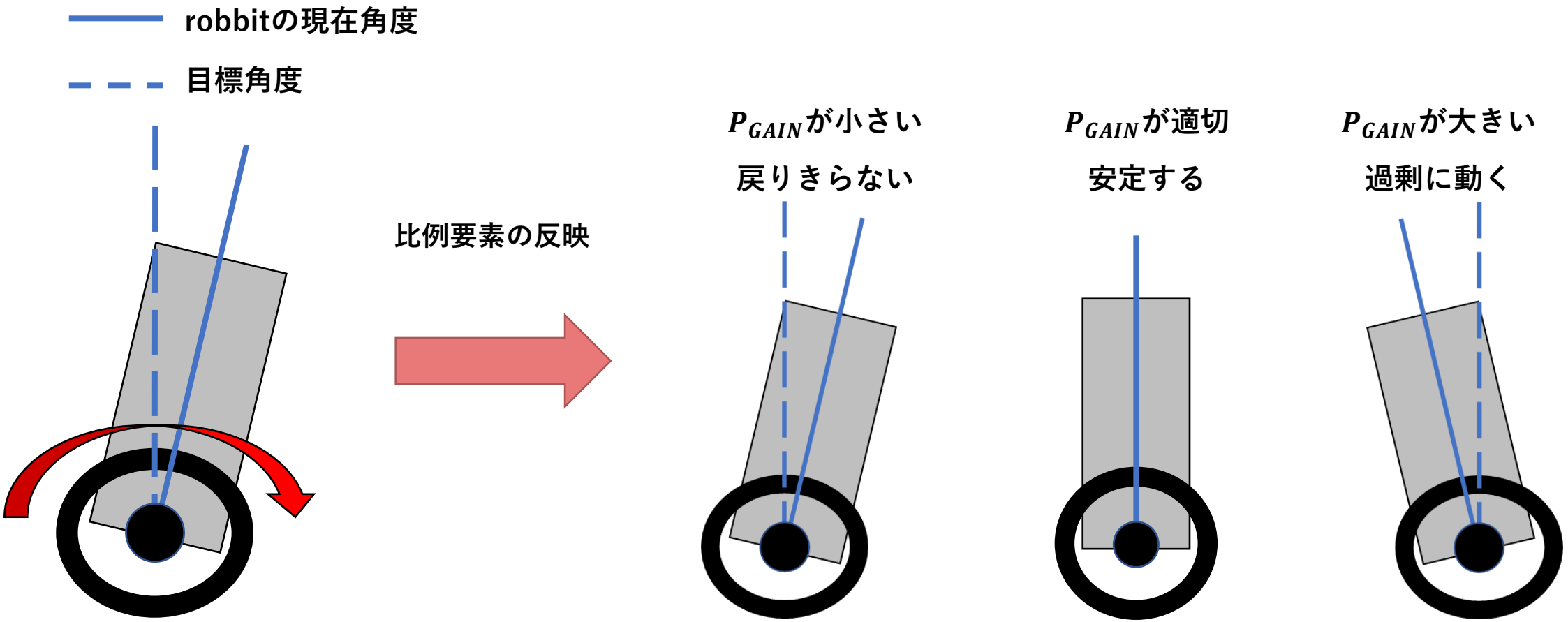
- 比例，積分，微分要素を組み合わせて操作量(角度)を決める
- 各要素の影響度合いは，ゲインによって決定される($P_{GAIN}, I_{GAIN}, D_{GAIN}$)
- 偏差が小さい場合の動力不足を解消するため，PWM_baseを加算



robbitのPID制御のブロック図

PID制御: 比例要素(P成分)

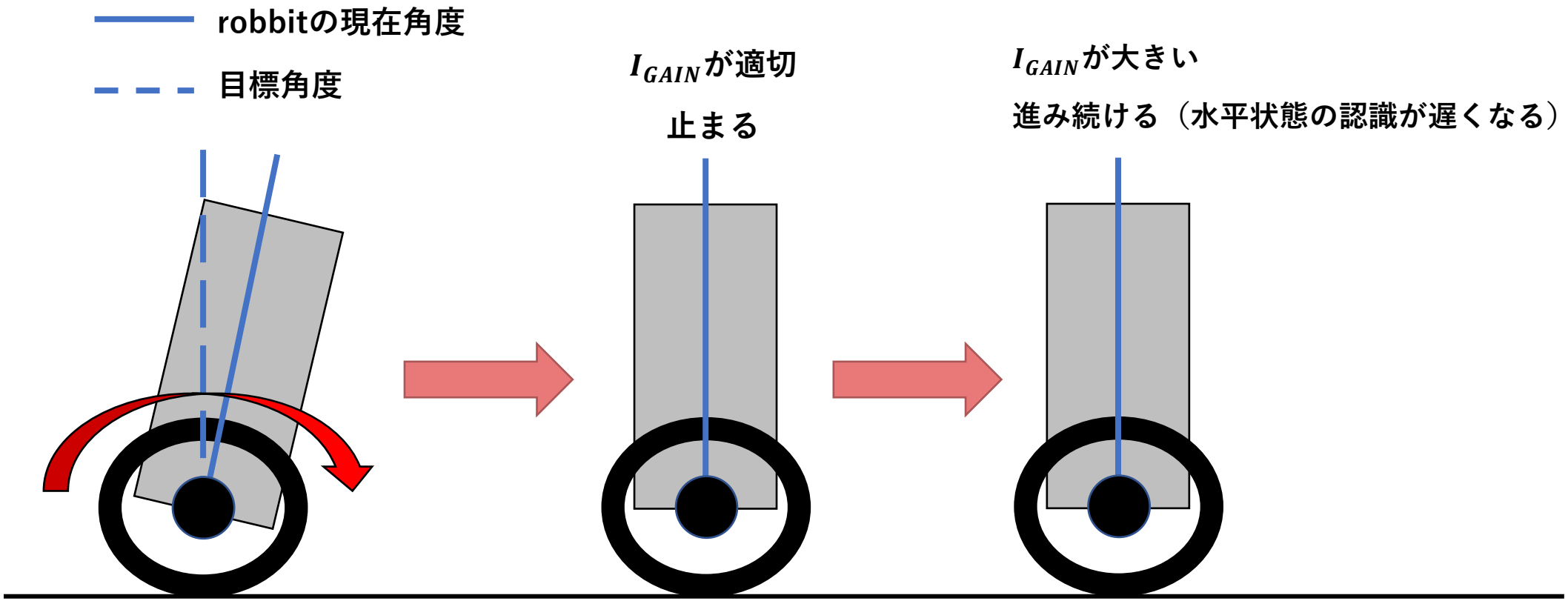
- 比例要素は操作量と目標値の誤差で表される
- P_{GAIN}が大きいと，偏差に対して敏感に反応する



比例要素の影響によるrobbitの動作イメージ

PID制御: 積分要素(I成分)

- 積分要素は操作量と目標値の誤差を時間積分している
- I_{GAIN}を大きくすると偏差を小さくできるが，大きすぎるとI成分の蓄積が大きくなり，システムの速応性や安定性が悪くなる



積分要素の影響によるrobbitの動作イメージ

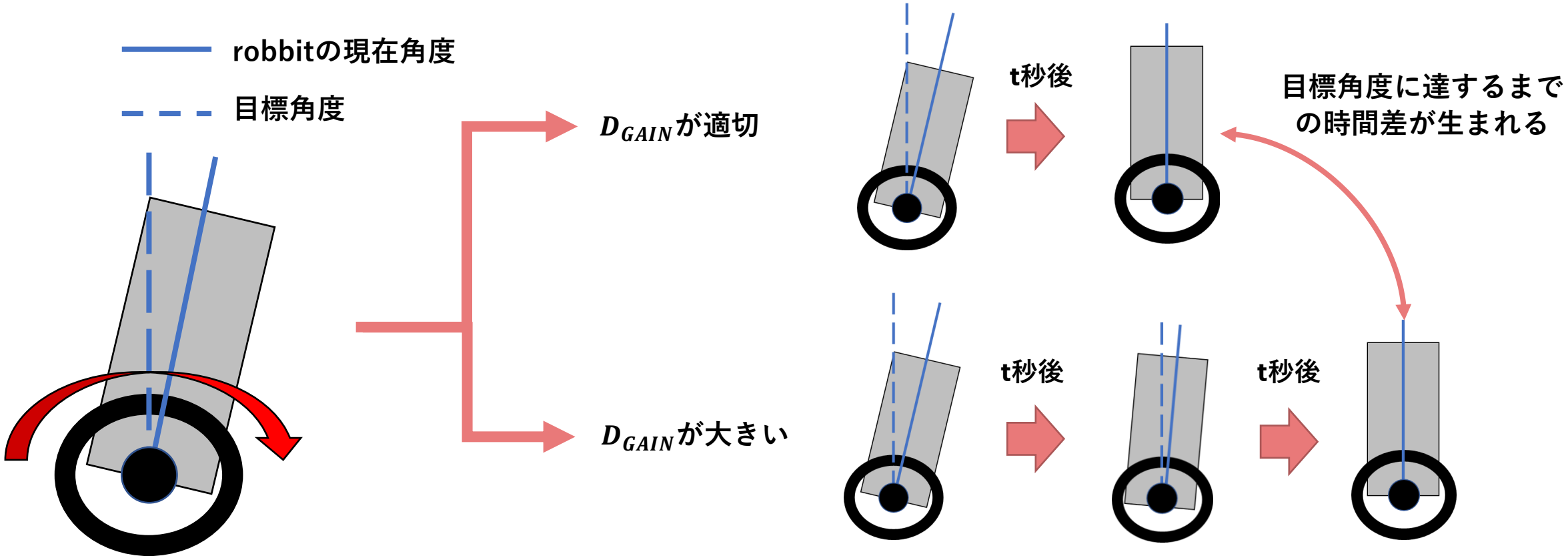
PID制御: アンチワインドアップ

- 前ページで述べた積分要素の蓄積を抑える手段としてアンチワインドアップが存在する
- 実装方法は，積分要素の計算結果に対して上限と下限を設定し，範囲内だったら積分要素の値を更新する

$$I = \begin{cases} I_{MAX} & (|I + P * dt| > I\_MAX) \\ -I_{MAX} & (|I + P * dt| > I\_MAX) \\ I + P * dt & else \end{cases}$$

PID制御: 微分要素(D成分)

- 微分要素は操作量と目標値の誤差を時間微分している
- D_{GAIN}を大きくすると，目標角度を行き過ぎるオーバーシュートが抑制される。大きすぎると目標角度に到達する時間が長くなる。



積分要素の影響によるrobbitの動作イメージ

ソフトウェア: PID制御

Institute of SCIENCE TOKYO

PID制御の実装は以下の通り

```
// PID control
P = (parameter.target - roll) / 90.0;          比例要素の計算
if(fabsf(I + P * dt) < I_MAX)  I += P * dt;  // cap    積分要素の計算（アンチワインドアップ）
D = (P - preP) / dt;     微分要素の計算
preP = P;

power = parameter.Kp * P + parameter.Ki * I + parameter.Kd * D; 出力の計算
```

- 変数Pでは比例要素を計算するため，目標角度(TARGET)と現在角度(roll)の偏差を導出する
- 変数Iでは積分要素を求めるため，比例要素を長方形近似で積分している
  If(fabsf(I+P*dt))の部分がアンチワインドアップの条件式に該当する
- 変数Dでは微分要素を求めるため，微分の公式を用いて計算を行っている
- 最後にpowerにすべての要素の値を足し合わせることで出力を計算する

ソフトウェア: PWM信号の決定

PWM制御の制御信号として，波形がHIGHの割合をPWM_{base}からV_{max}で表現する
式で表すと以下の通りである
$power$はPID制御の計算結果を表し，pwm_{base}はPWM信号の増加分を表す(詳しくは次ページ)

$$PWM = \begin{cases} power + PWM_{base} & (|power| < V_{MAX}) \\ V_{MAX} & (|power| \geq V_{MAX}) \end{cases}$$

また，powerの符号はモータの回転方向を決める制御信号として使われる

ソフトウェア: BLE通信

- PCとrobbit-espのBLE接続は既存のライブラリを用いて行う
- PCからの書き込み要求と読み込み要求をが来た時に，非同期にデータの送受信を行うため，コールバック関数として実装する

```
class MyCallbacks: public BLECharacteristicCallbacks {
  void onWrite(BLECharacteristic *pCharacteristic) {
    String status = pCharacteristic->getValue();
    int spaceindex = status.indexOf(' '); //index of space

    String parameter = status.substring(0, spaceindex); //parameter name
    float value = status.substring(spaceindex, status.length()).toFloat();

    if (parameter == "1") {
      target = value;
    } else if(parameter == "2"){
      Kp = value;
    } else if(parameter == "3"){
      Ki = value;
    } else if(parameter == "4"){
      Kd = value;
    } else if(parameter == "5"){
      pwm_base = value;
    } else if(parameter == "6"){
      vmax = value;
    }
  }

  void onRead(BLECharacteristic *pCharacteristic) {

  }
```

コールバック関数の処理

・onWrite : PCからの書き込み要求に対応する関数

・ onRead : PCからの読み込み要求に対応する関数

今回のプログラムではloop関数でPCに送信する値の設定を決めているので何もしない