

# robbit システムマニュアル

吉瀬研究室  
制作日：2025年9月

- このプログラムはrobbitのハードウェア，ソフトウェア，ゲートウェアの内容を説明している（ただし，ゲートウェアとはFPGA内に実装する回路を指す）
- この資料を参考にすることで，robbitのシステムを理解することができ，ソフトウェアやゲートウェアの改善を行いやすくなる

# robbit : ハードウェア

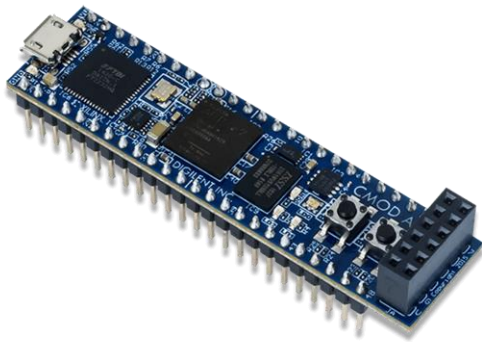
## ■ FPGA ボード: Cmod A7-35T

- サイズ: 69.9mm × 18.8mm
- 重量: 10.7g

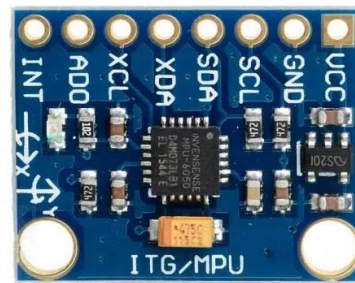
## ■ センサ: MPU-6050(IMU)

- 加速度, 角速度(ジャイロ)センサ

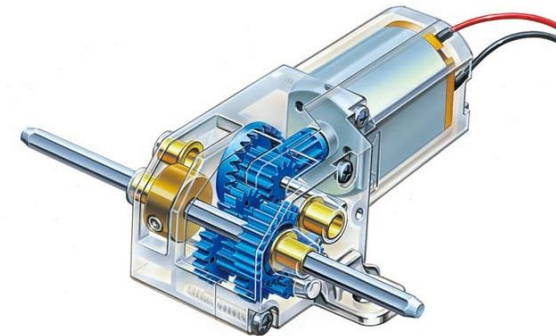
## ■ ギアモータ: ミニモータ標準ギヤボックス(TAMIYA)



Cmod A7-35T



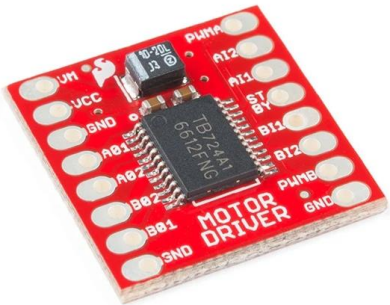
MPU-6050



ミニモータ標準ギヤボックス(TAMIYA)

# robbit : ハードウェア

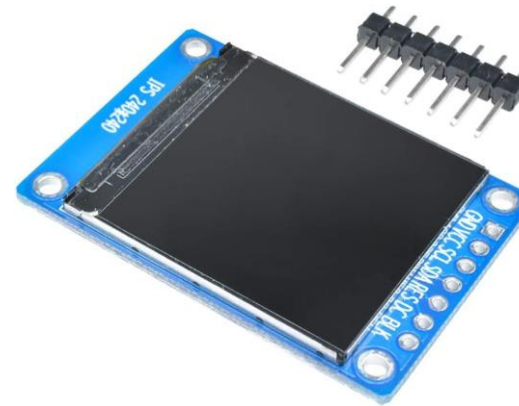
- モータドライバ: TB6612FNG搭載 モータドライバ
- タイヤ:スリムタイヤセット(55mm 径) 70193
- ディスプレイ: ST7789搭載 240×240 ミニディスプレイ
- バッテリー: EEMB 653042(3.7V)



TB6612FNG搭載  
モータドライバ



スリムタイヤセット

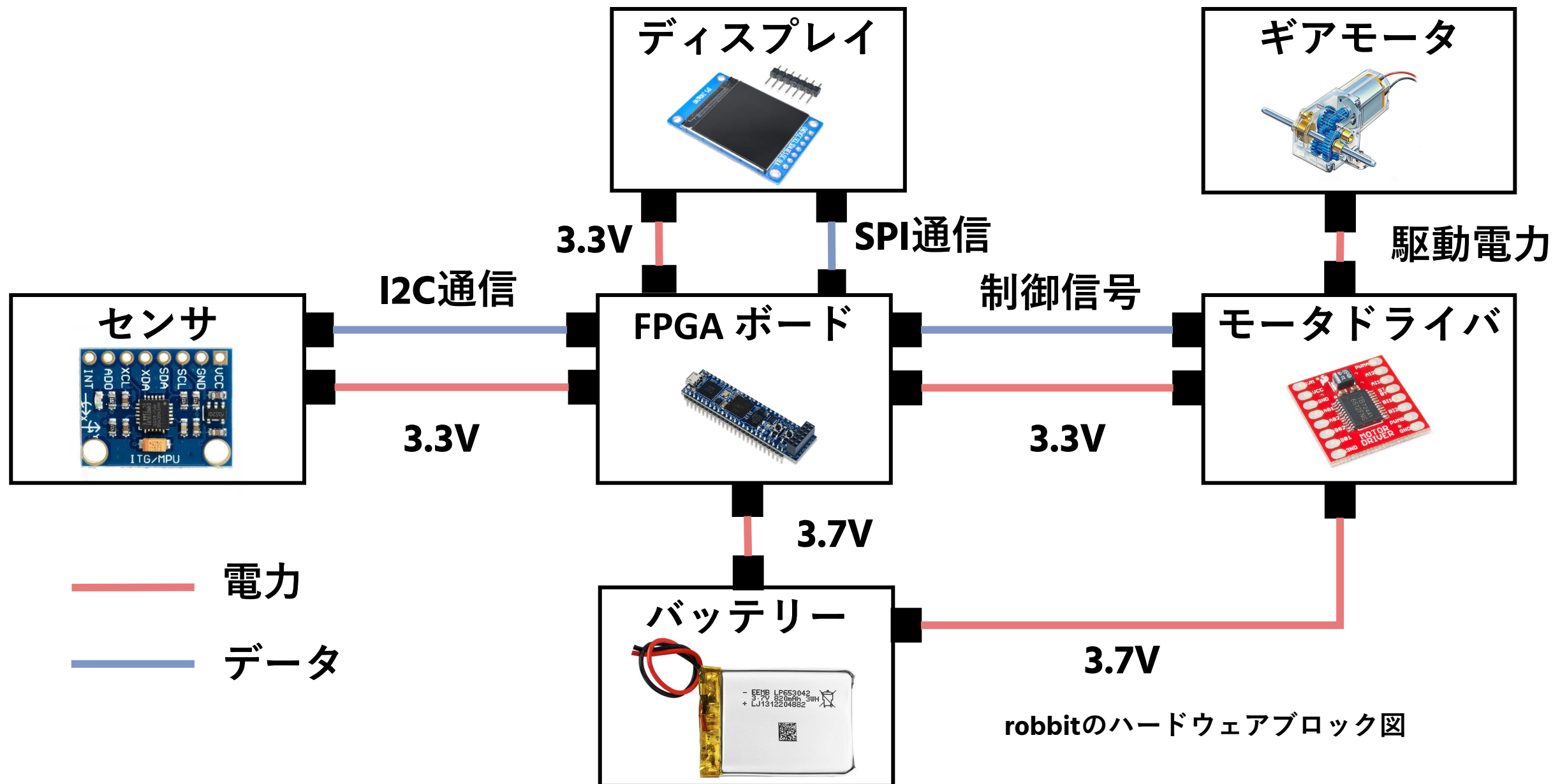


ST7789搭載  
ミニディスプレイ



EEMB 653042  
充電式バッテリー

# robbit : ハードウェア

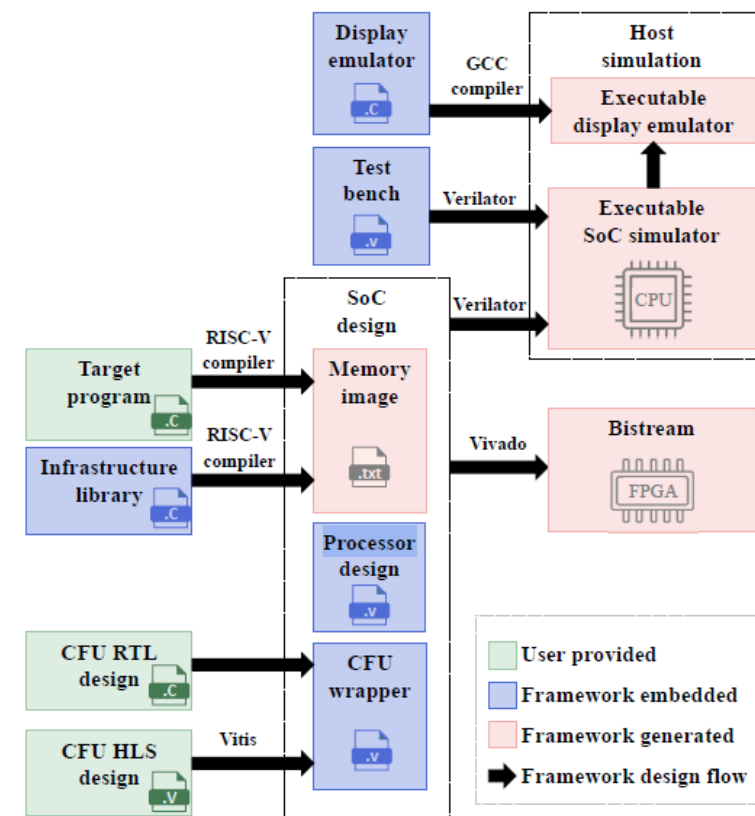


# ゲートウェア: 概要

## ■ CFU-Proving-Groundを使用する

- 175MHzまでの高速制御をサポート
- ミニディスプレイをサポート済み
- Memory Mapped IO(MMIO)を柔軟に設計可能

CFU-Proving-Groundにモータドライバモジュール, IMUモジュールを追加



# ゲートウェア: 概要

各verilogファイルで行われている処理内容は以下の通り

- main.v  
ディスプレイとのspi通信,
- mpu.v  
IMUとのI2C通信によるデータの送受信
- proc.v  
32bitのRISC-Vプロセッサを実装
- robbit.v  
モータドライバ制御モジュール(tb6612fng) : PWM制御を担当  
MMIOモジュール : robbitで新たに使用するメモリ領域にデータを書き込む

mpu.vとrobbit.vはCFU Proving Groundから新たに追加したプログラムである

# ゲートウェア：I2C通信モジュール

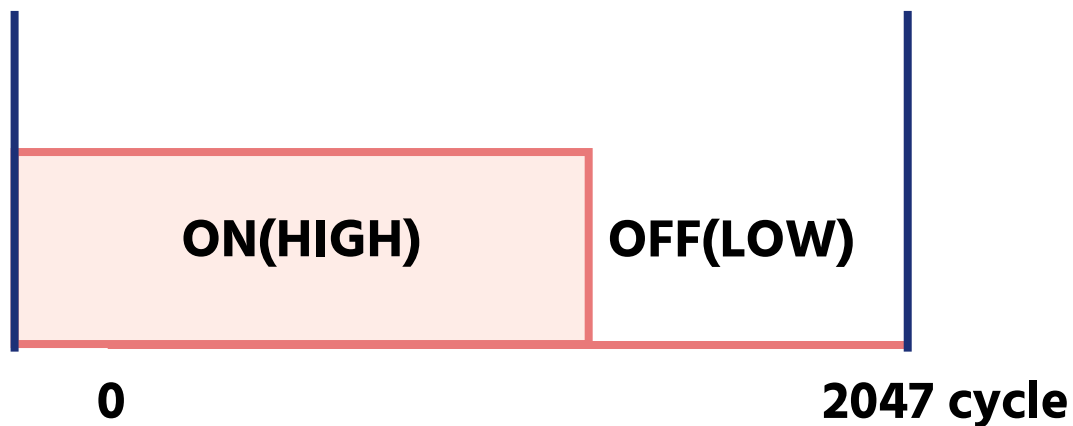
- MPU-6050との通信はI2C通信で行う
- mpu.vにあるmpu6050モジュールとi2c\_masterモジュールが担当している  
mpu6050モジュール: I2C通信でのデータやり取りのタイミング制御  
i2c\_masterモジュール: IMUとFPGAのデータのやり取りを担う
- I2C通信はsclとsdaという2本の通信線でデータのやり取りを行う  
sclがクロックの役割を果たし， sdaがデータバスの役割を果たす
- sclに合わせてsdaにデータを1bitずつ送受信する



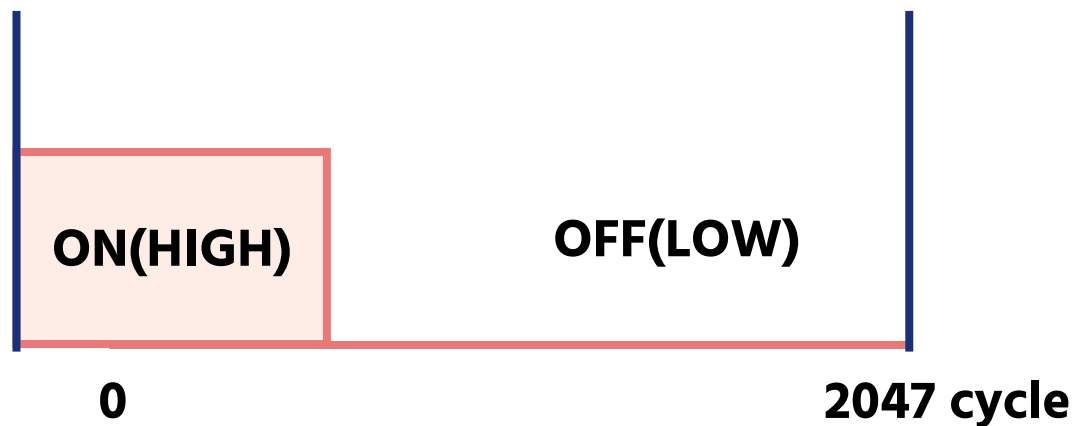


# ゲートウェア：tb6612fngモジュール

- robbit.v内に、モータドライバを制御するtb6612fngモジュールが存在する
  - モータドライバにPWM制御の制御信号を送る
  - PWM制御
    - 電気信号のON/OFF(デューティー比)の割合でモータの強さを制御する
- ONの割合が大きい = モータの回転速度が速い  
ONの割合が小さい = モータの回転速度が遅い

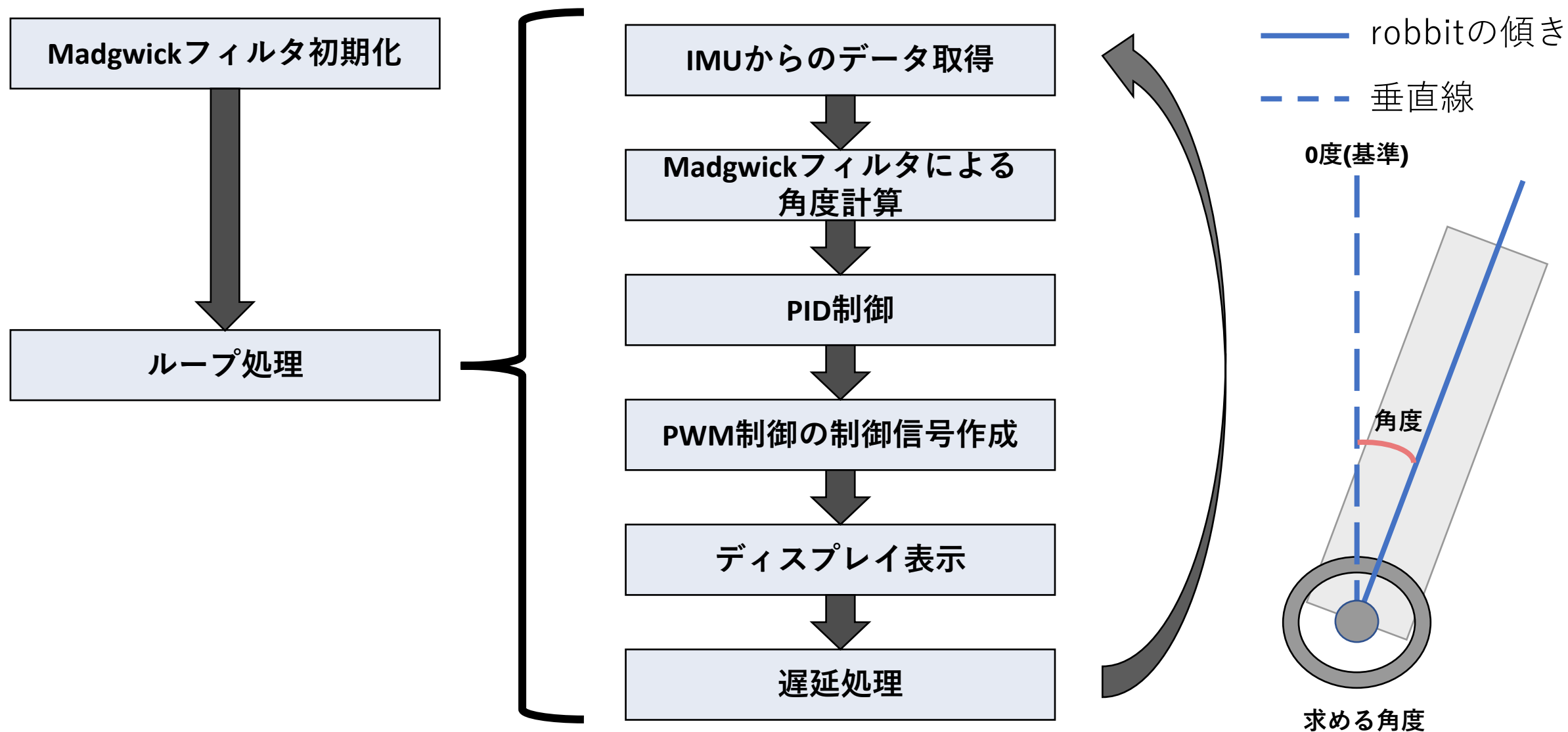


モータの回転速度が速い場合



モータの回転速度が遅い場合

# robbit : ソフトウェア



# 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
- 実際にはゲートウェア上でMemory Mapped IO(MMIO)によって加速度と角速度が書き込まれた後に、ソフトウェアで該当するアドレスにアクセスしてデータを読み込んでいる

```
//get acceralation and angular velocity
int16_t ax, ay, az, gx, gy, gz;
unsigned int data;
data = *(MPU_ADDR_ayax);
ax = data & 0xffff;           x,y軸の加速度取得
ay = data >> 16;

data = *(MPU_ADDR_gxaz);
az = data & 0xffff;           z軸の加速度, x軸の角速度取得
gx = data >> 16;

data = *(MPU_ADDR_gzgy);
gy = data & 0xffff;           y,z軸の角速度取得
gz = data >> 16;

//calculation of roll
MadgwickFilter.updateIMU(-gz / 131.0, gy / 131.0, gx / 131.0,
                        -az / 16384.0, ay / 16384.0, ax / 16384.0);
roll = MadgwickFilter.getRoll();
```

# ソフトウェア: Madgwickフィルタ

- 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フィルタの処理は, `main.cpp`にMadgwickクラスとして定義している

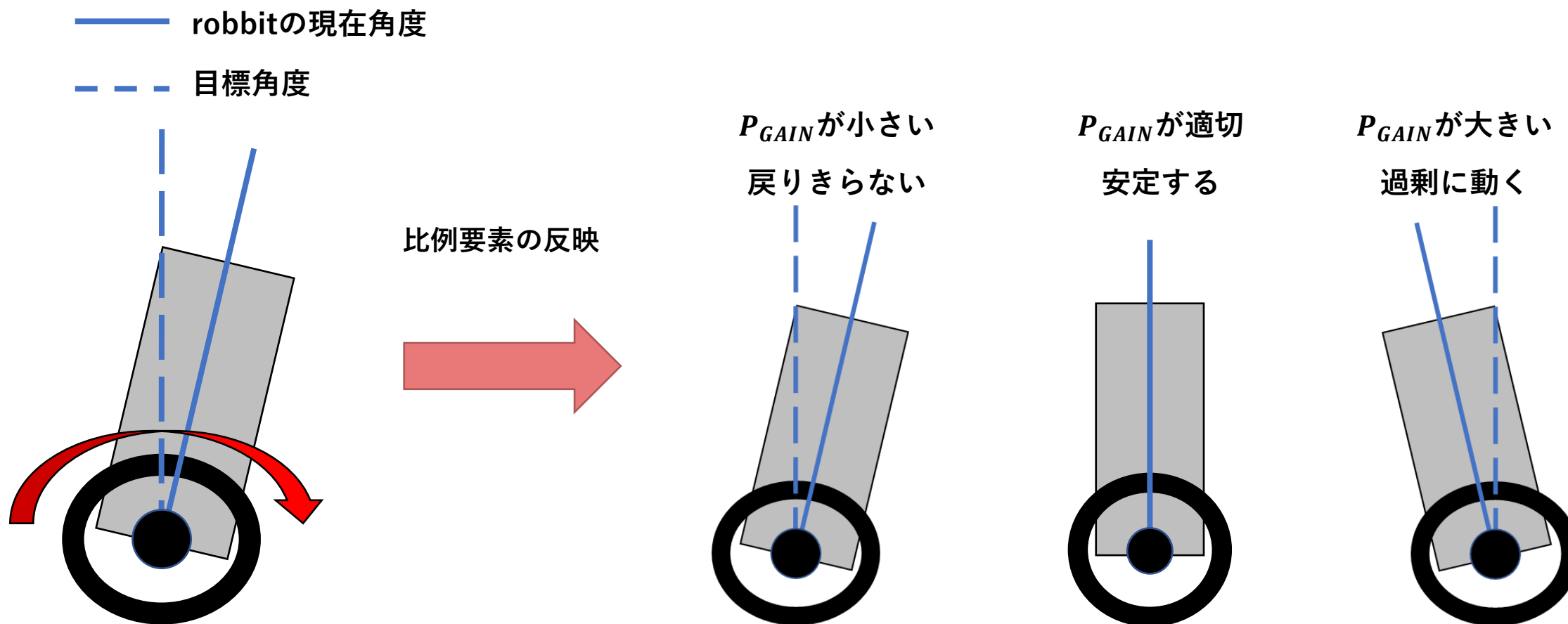
## ■ PID制御

- 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 ( $P_{GAIN}, I_{GAIN}, D_{GAIN}$ )

$$Power = P_{GAIN} \times P + I_{GAIN} \times I + D_{GAIN} \times D$$

# PID制御: 比例要素(P成分)

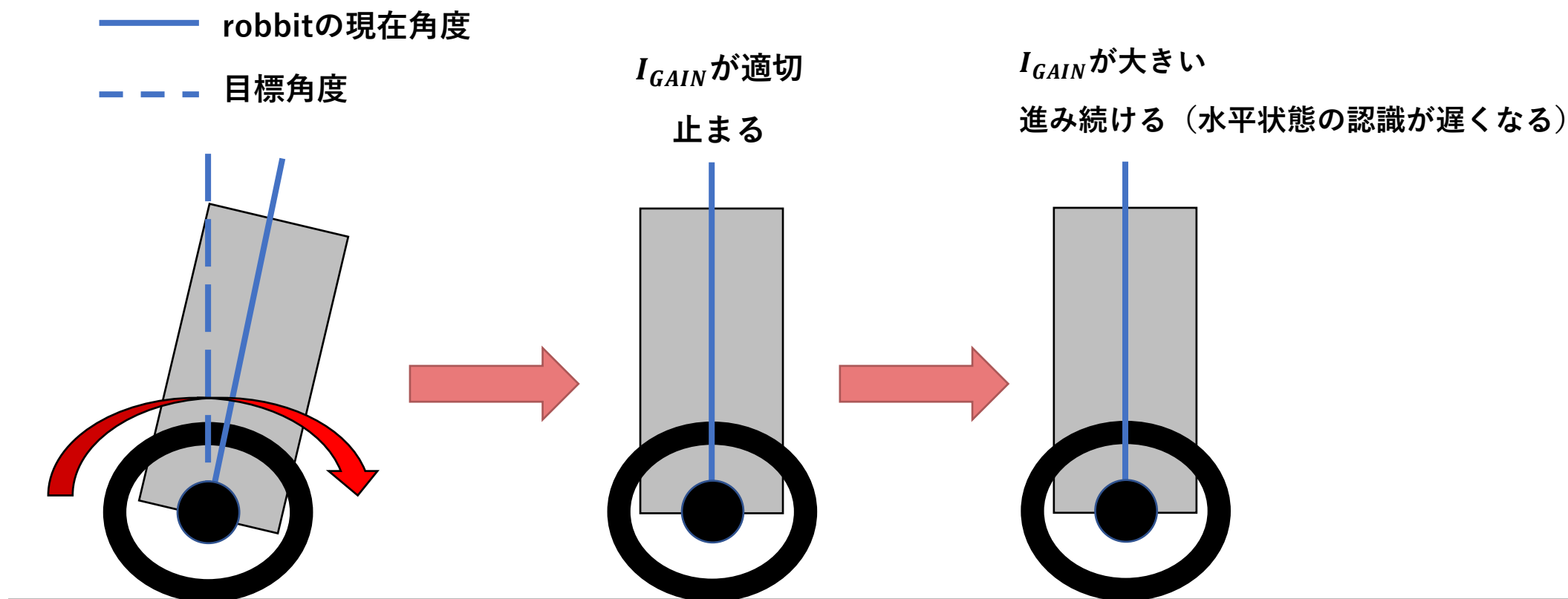
- 比例要素は操作量と目標値の誤差で表される
- $P_{GAIN}$  が大きいと、偏差に対して敏感に反応する



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

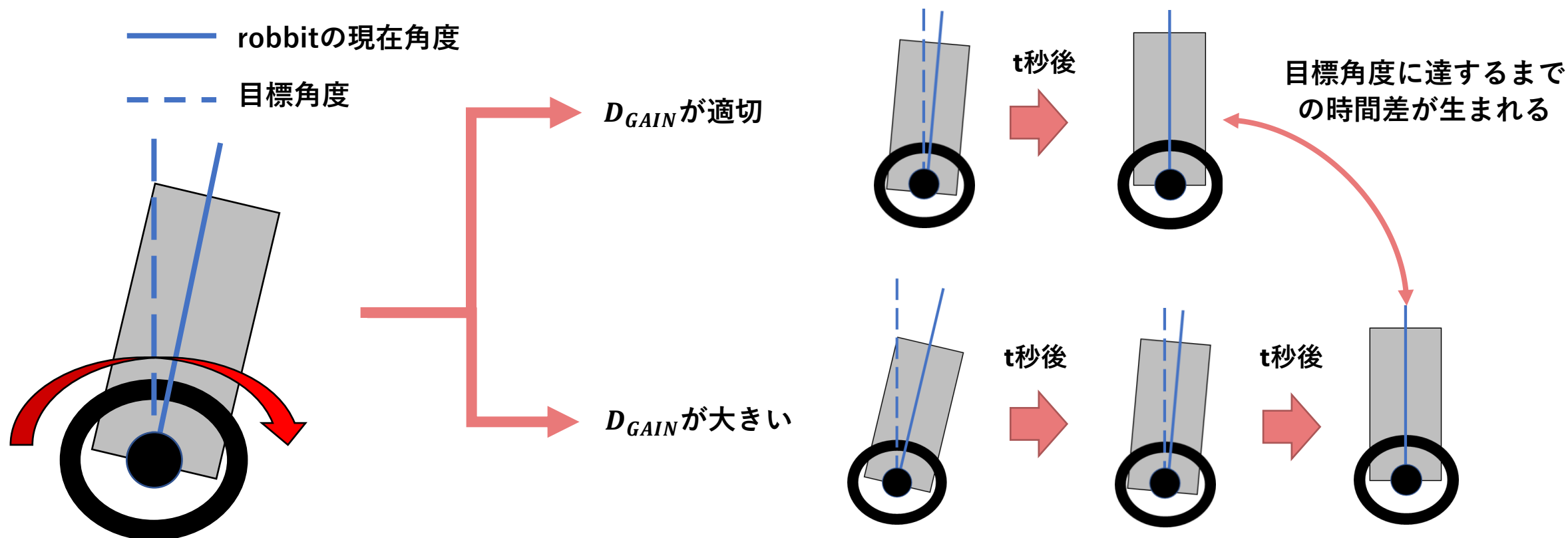
- 前ページで述べた積分要素の蓄積を抑える手段としてアンチwindアップが存在する
- 実装方法は、積分要素の計算結果に対して上限と下限を設定し、範囲内だったら積分要素の値を更新する

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



# 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

## 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wind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wind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