

# 1 四軸飛行器介紹



Figure 1: 一個常見的四軸飛行器

## 2 物理模型

要維持四軸飛行器的平衡，須要考慮以下幾個物理量。

- 外力造成的加速度，即  $a_x, a_y, a_z$ 。
- 力矩造成的角加速度，即  $\alpha_x, \alpha_y, \alpha_z$ 。

可以藉由控置馬達來調整四軸飛行器所受的力與力矩。我們不妨假設四個馬達的推力分別為  $F_1, F_2, F_3, F_4$ 。

### 2.1 升力

四軸飛行器所受的升力即為四個馬達推力的總合，即

$$F = F_1 + F_2 + F_3 + F_4$$

### 2.2 力矩- $x, y$ 方向

兩個對角的螺旋槳的升力差會使四軸飛行器受到一個  $x$  或  $y$  方向的力矩，即

$$\tau_x = (F_2 - F_4)r, \quad \tau_y = (F_3 - F_1)r$$

其中  $r$  是四軸的臂長（馬達到質心的距離）。

### 2.3 力矩- $z$ 方向

四軸飛行器的螺旋槳由 1 號到 4 號是正反轉交錯的，而螺旋槳加速轉動時，或即使是等速轉動也會因空氣的摩擦力，產生一個反方向的力矩。

$$\tau_z = F_1 - F_2 + F_3 - F_4$$

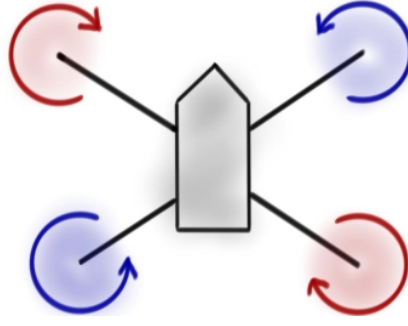


Figure 2:

### 3 PID

四軸飛行器是一個不隱定的系統，我們必須藉助電腦控置來達到平衡。PID 控置就是一種方法。首先我們設定一個目標態，比如若要四軸飛行器平衡，目標應該設為

$$z = 0, \theta_x = \theta_y = \theta_z = 0$$

對於上例的每一項，我們都有對應的控制方法。比如若要  $z$  增加，我們應該提高升力，即各個馬達的轉速。而若要  $\theta_x$  減少，我們便應該提高 4 號馬達的轉速，降低 2 號馬達的轉速。我們把這種調整稱作對應的反應 (action)。假設  $x$  是目標的值，而  $x'$  是實際的值，即誤差為  $e = x - x'$ 。現在我們必須跟據誤差來調整反應  $y$  的大小。

PID 就利用比例項 (P)，積分項 (I) 以及微分項 (D) 來調控反應。公式為

$$y(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}$$

其中  $K_p, K_i, K_d$  分別為比例項 (P)，積分項 (I) 以及微分項 (D) 的增益。

#### 3.1 比例項 (P)

比例項顧名思義，就是和誤差正比的那一項，即  $K_p e(t)$ 。 $K_p$  太低會使得系統反應太慢，太高會使得系統不穩定。

#### 3.2 積分項 (I)

積分項可以消除系統的偏差，如馬達實際轉速不同等等。 $K_i$  太大也會使得系統不穩定。

#### 3.3 微分項 (D)

微分項可以有效的使系統穩定，但容易受感應器的誤差所影響。

#### 3.4 離散 PID

因為我們運算、感應器偵測都是需要時間的，通常 PID 只能每  $T$  單位的時間才計算一次。在這個情況下，

$$y[n] = K_p e[n] + K_i \sum_0^n e[i] \Delta t_i + K_d \frac{e[n] - e[n-1]}{\Delta t}$$

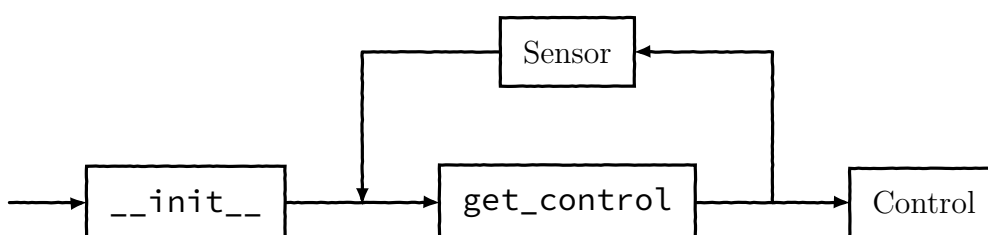
## 4 實作細節

### 4.1 概覽

你必須實作 `mymath/mypid.py` 中的 `class MyPID`。其中有兩個函數需要你修改

- `__init__(self, kp, ki, kd, *, imax)`: 這個函數是用來初始化 PID 用的。你可以在裡面宣告一些之後要用的變數。
- `get_control(self, t, err, derr)`: 這個函數是主要的函數。給定當前的時間 (`t`)、誤差 (`err`) 以及誤差的微分 (`derr`)，你必需回傳反應的大小。

#### 4.1.1 函式呼叫流程



如圖所示，程式一開始會呼叫初始函式 `__init__` 一次，接著每隔一小段時間就會從感測器得到資訊，計算誤差後傳進 `get_control` 裡得到控制的反應大小。

### 4.2 模擬器使用方法

請在瀏覽器網址列輸入 `http://localhost:8000` 來執行模擬器。如果瀏覽器找不到網頁，請在終端機輸入 `coffee server.coffee -n` 後再開啓瀏覽器。

### 4.3 提示

請跟著提示一步一步完成這份程式碼！

如果你遇到困難，不要害怕，趕緊問問身旁的人吧！

而如果你覺得太簡單了，也請你不要吝惜教教身旁的人吧！

#### 4.3.1 `imports` and `class`

---

```
1 from mymath.pid import BasePID
2 import numpy as np
3
4 class MyPID(BasePID):
```

---

- 第 1-2 行：引入一些模組。`BasePID` 是你的 PID 要繼承的母物件。而 `numpy ...` 是一個很猛的科学計算模組，只是因為一些歷史因素我們沒有把他刪掉。你應該不會需要用到他。

- 第 4 行：這裡定義了我們的物件 `MyPID`，他繼承了 `BasePID` 這個母物件。至於什麼是繼承呢？基本上你會擁有 `BasePID` 的所有東西，你還可以在他之上修改原有的一些東西。細節應該不太重要！

#### 4.3.2 `__init__`

---

```
5 def __init__(self, kp, ki, kd, *, imax):
6     super().__init__(kp, ki, kd, imax=imax)
7     self.last_time = None
8     # TODO:
9     # Init the variables you need...
```

---

這裡主要是放一些初始化需要做的事情。

- 第 5 行：定義我們的 `__init__` 函數。為什麼要取那麼怪的名子呢？其實這是 python 定義的，所有 python 物件的建構函數都要是這個名子！
- 第 6 行：初始化這個物件所繼承的母物件。詳細的細節可以不用理會，但要注意執行了這行後，會自動幫你設好這些變數。

- `self.kp`：即  $K_p$  的值。
- `self.ki`：即  $K_i$  的值。
- `self.kd`：即  $K_d$  的值。
- `self.int_restriction`：誤差累積的上限，後面會詳細說明。

這些變數你就不需要在自行設定了。

- 第 7..9 行：這邊需要你初始化一些之後你所需要的變數。請注意變數可能需要在 `self` 下定義，否則當這個函數結束後變數就會消失了。也就是說你的變數應該會長的像 `self.a` 等等。在這邊做為範例我們幫你初始化了一個變數 `self.last_time` 為 `None` 了 (你不需要更改這一行)。至於要初始化哪些變數，可以接下來需要的時後再決定！

#### 4.3.3 `get_control(self, t, err, derr)`

---

```
11 def get_control(self, t, err, derr):
12     if self.last_time is None:
13         self.last_time = t
14         return 0.
15
16     # TODO:
17     # What should up be ?
18     up = 0.
19
```

---

```

20     # TODO:
21     # What should ud be ?
22     ud = 0.
23
24     # TODO:
25     # Calculate dt. Remember to update
26     # some variables you defined
27     dt = 0.
28     self.last_time = t
29
30     # TODO:
31     # Calculate the sum of the error.
32     something = 0.
33
34     # TODO:
35     # Restrict the sum of the error to be within
36     # [-self.int_restriction, self.int_restriction]
37     if something > self.int_restriction:
38         pass
39
40     # TODO:
41     # What should ui be ?
42     ui = 0.
43
44     # TODO:
45     # Return the sum of up, ud and ui !
46     return 0.

```

---

程式碼雖然看起來很長，但其實不會很複雜！我們一行一行來看！

- 第 11 行：定義這個函數，我們看一下他的參數。第一個 `self` 顧名思意就是代表自己！`t`, `err`, `derr` 則分別代表當前的時間、誤差以及微分向的誤差。也就是

$$\begin{aligned}
 t &= t[n] \\
 err &= e[n] \\
 derr &\approx \frac{e[n] - e[n-1]}{t[n] - t[n-1]}
 \end{aligned}$$

至於為什麼 `derr` 明明可以用 `t`, `err` 得出卻還要傳一個近似的值進來呢？其實是因為直接計算的話容易因為感測器誤差而出現不可遇期的結果，因此我們會用其他方法 (Kalman Filter) 幫你算好了。

- 第 12..14 行：如果這是第一次執行 ( $n = 0$ ) 我們直接回傳 0。這個部分全部幫你寫好了，不需要去動他。
- 第 16..18 行：還記得 PID 的公式嗎？忘記了也沒關係！我們再複習一次

$$y[n] = K_p e[n] + K_i \sum_0^n e[i] \Delta t_i + K_d \frac{e[n] - e[n-1]}{\Delta t}$$

$$= u_p + u_i + u_d$$

其中  $\Delta t = t[n] - t[n-1]$ ，而  $u_p, u_i, u_d$  分別是 P, I, D 項的反應。現在做為暖身請你修改第 8 行的值，算出  $u_p$  (也就是 `up`)，由上面的式子我們可以知道  $u_p = K_p e[n]$ 。

注意到  $e[n] = \text{err}$ ，且  $K_p$  我們已經幫你存在 `self.kp` 了。

- 第 20..22 行：與剛剛類似，請修改第 12 行的值，算出  $u_d$  (也就是 `ud`)。  $u_d$  的定義為

$$u_d = K_d \frac{e[n] - e[n-1]}{t[n] - t[n-1]}$$

但實際在計算時，請直接把  $(e[n] - e[n-1]) / (t[n] - t[n-1])$  用 `derr` 近似。

- 第 24..28 行：最後我們要開始算積分項。首先我們要算出  $\Delta t$  (`dt`)。請由  $\Delta t$  的定義

$$\Delta t = t[n] - t[n-1]$$

下手！注意到  $t[n-1]$  會被存在 `self.last_time` 裡。

計算完後也不要忘記更新 `self.last_time` 的值成當前的時間，下一次計算才可以使用！

- 第 30..32 行：計算積分的累計！也就是要計算

$$I = \sum_0^n e[i] \Delta t_i \approx \sum_1^n e[i] \cdot (t[i] - t[i-1])$$

咦？怎麼下標變成從 1 開始了？其實是因為  $n = 1$  時  $t[i-1] = t[-1]$  沒有定義。這也是為什麼我們在 2.4 行要直接跳出的原因！

至於要如何計算呢？其實就是把  $e[n] \cdot \Delta t_n$  一直累加而已。但是要累加在哪一個變數呢？我們似乎還沒定義這樣的一個變數呢！

嘿嘿！這時候就要請你在初始化的時候就定義好囉！具體一點，要做的事情如下：

1. 在 `__init__` 裡初始化一個你自己定義的變數為 0。

注意這個變數必須要定義在 `self` 之下，才會被保存下來。

2. 每一次 `get_control` 被乎叫時便累加他。

- 第 34..38 行：上一步我們計算出了  $I = \sum e[i] \Delta t_i$ ，但在實際中我們必須給這個數字一個閾值！否則  $I$  如果可以無限累積，最後一定會超出馬達可以容許的範圍！

在這裡我們會給他一個上限  $I_R$ ，也就是我們必須規定  $|I| \leq I_R$ ，或是說  $-I_R \leq I \leq I_R$ 。因此我們要做的就是









- Case 1:  $-I_R \leq I \leq I_R$  這個情況  $I$  沒有超過範圍，所以我們不用動他。
- Case 2:  $I \geq I_R$  這個情況  $I$  超過右界，所以我們把他拉回來，也就是令  $I = I_R$ 。
- Case 3:  $I \leq -I_R$  這個情況  $I$  超過左界，所以同樣的我們令  $I = -I_R$ 。

在我們的程式中  $I_R = \text{self.int\_restriction}$ 。

- 第 40..42 行：最後計算出  $u_i$ ， $u_i = K_i \sum e[j] \Delta t_j = K_i I$ ，其實就是乘一個係數而已！
- 第 44..46 行：最後的最後別忘了把答案傳回去！否則就像考試忘了把答案填在答案卡上一樣！

## 4.4 測試

如果你已經完成了以上程式，打開模擬器並執行，理論上四軸飛行器應該可以保持平衡了！模擬器有以下功能。

	逆時鐘旋轉		
	高度上升		
	高度下降		
	順時鐘旋轉	滑鼠左鍵	鏡頭旋轉
	往前傾斜	滑鼠右鍵	鏡頭平移
	往後傾斜	滑鼠滾輪	鏡頭前後
	往左傾斜		
	往右傾斜		