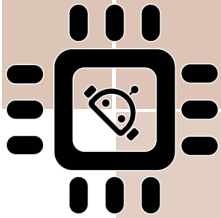


ARDUINO CHO NGƯỜI MỚI BẮT ĐẦU

Quyển không còn là cơ bản

minht57 lab

Lời nói đầu

Arduino được xem là nền tảng điện tử thông dụng và dễ dàng tiếp cận bậc nhất hiện nay. Nhờ vào bộ thư viện được chuẩn hóa và cộng đồng Arduino siêu lớn hiện nay thì việc lập trình cũng không quá phức tạp.

Tiếp theo hai quyển sách đầu tiên **"Arduino cho người mới bắt đầu"** thì đây là quyển thứ 3 bàn về các vấn đề xung quanh cách lập trình như phong cách code, một vài cách thiết kế chương trình phổ biến, một số thuật toán thường dùng,... Quyển sách này sẽ cho bạn một cái nhìn code không chỉ là code mà code là một nghệ thuật được thiết kế bởi lập trình viên. Quyển sách này không phải hướng dẫn cách bạn lập trình Arduino mà chủ yếu là giới thiệu ý tưởng và Arduino để thực hành ý tưởng đó.

Quyển sách sẽ giới thiệu các phần như sau **một số khái niệm** cần biết, **phong cách lập trình** (coding style), **máy trạng thái**, **bộ điều khiển PID**, **bộ lọc Kalman**, **GIT** và một **quy trình tự phát triển sản phẩ**m trong giai đoạn luyện tập.

Lưu ý quyển sách này mang hơi hướng cá nhân và sẽ được bổ sung thêm nhiều phần mới trong các kỳ tái bản tiếp theo.

minht57 lab

Hướng dẫn đọc sách

Bộ sách **Arduino dành cho người mới bắt đầu** gồm 3 quyển đi từ mức độ cơ bản đến chuyên sâu bao gồm **quyển cơ bản**, **quyển rất cơ bản** và **quyển không còn là cơ bản**. Bộ sách này sẽ cung cấp cho bạn một tư duy làm việc với một vi điều khiển hơn là chỉ thực hành Arduino đơn thuần.

Bộ 3 quyển sách sẽ cung cấp cho bạn rất nhiều thông tin ở mức căn bản dưới dạng từ khóa và tóm tắt vấn đề (vì nếu giải thích vấn đề rõ ràng, chuyên sâu thì sẽ rất lan man và dài dòng). Nếu bạn quan tâm một vấn đề nào cụ thể thì cứ dựa vào những từ khóa đã được nêu ra và tìm hiểu thêm trên internet hoặc sách vở.

Vì mục tiêu của quyển sách là hướng đến những bạn học lập trình Arduino định hướng chuyên sâu nên sách sẽ không tập trung vào từng module cảm biến, thiết bị chấp hành hay một dự án nào cụ thể. Mà cấu trúc sách đi theo hướng học một vi điều khiển tổng quát mà Arduino chỉ là một ví dụ cụ thể.

Quyển này sẽ giới thiệu từng bài riêng lẻ và không có sự liên kết với nhau nên bạn đọc có thể chọn bất kỳ chương nào bạn quan để đọc.

Nếu bạn cảm thấy văn phong hoặc cách tiếp cận của quyển sách không phù hợp với bạn thì bạn có thể bỏ qua.

Trong quá trình viết và biên soạn sách không thể tránh khỏi những sai sót về mặt nội dung cũng như hình thức. Nhóm tác giả rất mong nhận được sự góp ý của các bạn đọc để quyển sách ngày càng hoàn thiện hơn và có thể truyền tải nội dung đến với nhiều bạn đọc hơn.

Xin cảm ơn bạn đọc.

minht57 lab

Mục lục

MỘT SỐ KHÁI NIỆM 1

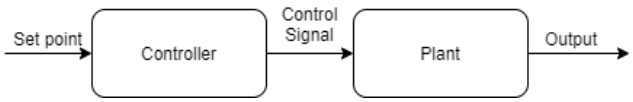
1.1 IDE

Integrated Development Environment (IDE) được dịch là môi trường phát triển tích hợp. IDE là một phần mềm kết hợp nhiều công cụ khác nhau như trình soạn thảo text (Text Editor), trình biên dịch/thông dịch (Compiler/Interpreter), trình kiểm tra lỗi (Debugger), chương trình nạp code (Deploy). IDE rất là tiện dụng và dễ dàng thao tác vì thường được thiết kế bằng Graphical User Interface (GUI).

Thông thường một IDE sẽ có một vài tính năng làm cho người dùng cảm giác gắn bó hơn với IDE như làm nổi bật các cú pháp (syntax highlighting), gợi ý tên biến/tên chương trình/từ khóa (code completion), kiểm soát phiên bản (version control), gỡ lỗi (debugging), tìm kiếm (search),...

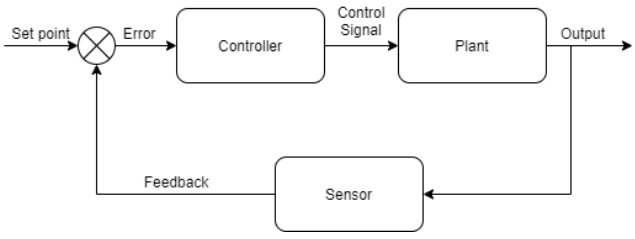
1.2 Phương pháp điều khiển

Phương pháp điều khiển vòng hở là điều khiển không phản hồi. Vì thế chất lượng có thể không đạt được như mong muốn vì bộ điều khiển không quan tâm đến ngõ ra.



Hình 1.1: Sơ đồ điều khiển vòng hở.

Phương pháp điều khiển vòng kín là phương pháp điều khiển có phản hồi. Bộ điều khiển sẽ kiểm tra sai số giữa ngõ ra và điểm đặt ngõ vào để điều chỉnh tín hiệu điều khiển hợp lý.



Hình 1.2: Sơ đồ điều khiển vòng kín.

Bộ điều khiển thì sẽ có rất nhiều loại khác nhau và tùy vào mục đích sử dụng mà chọn bộ điều khiển nào cho phù hợp. Trong quyển này sẽ lấy ví dụ về bộ điều khiển phổ biến nhất là PID.

PHONG CÁCH LẬP TRÌNH (CODING STYLE) 2

2.1 Giới thiệu

Việc lập trình có một phong cách là một điều rất quan trọng vì nó có rất nhiều điều có ích lợi. Nếu bạn là một lập trình viên thực thụ và khi bạn đi làm cho các dự án thì bạn phải lập trình theo một cái quy định gọi là coding convension, nó là một loạt các quy tắc mà bạn phải tuân thủ theo khi lập trình.

Một số lợi ích khi lập trình theo một phong cách (style) nào đó:

- Code sẽ rõ ràng, trong sáng, sạch sẽ.
- Dễ đọc, dễ bảo trì code.
- Người khác đọc code sẽ nắm bắt nhanh hơn.
- Tối ưu được code trong một số trường hợp. Tránh được một số lỗi không nên xảy ra.
- <còn nhiều lắm>

2.2 Một số nguyên tắc

Project

Startup code

- Startup code thường được cung cấp bởi nhà sản xuất và chỉ khi cần thiết thì mới cần chỉnh sửa. Nếu có chỉnh sửa thì phải comment lại những gì đã chỉnh sửa, version của các công cụ như compiler, assember, linker, processor.
- Không nên xóa code trong file startup code mà chỉ nên comment khi thấy không cần thiết.

Stack và Heap

- Luôn khởi tạo stack là một số chẵn (và nên là lũy thừa của 2).
- Luôn khởi tạo giá trị ban đầu cho stack (ví dụ 0x55) để khi debug xem có tràn stack hay không.
- Không nên sử dụng các hàm liên quan đến cấp phát bộ nhớ động (malloc()). Vi điều khiển thường gặp vấn đề với Heap. Việc cấp phát bộ nhớ động và giải phóng bộ nhớ sẽ gây phân mảnh bộ nhớ qua thời gian, điều này ảnh hưởng xấu đến RAM.

Biến

Tên

- Nên đặc tên biến đầy đủ và thể hiện được ý nghĩa của biến.
- Chọn cách viết tên biến theo kiểu thisIsVariable (gọi là Camel Case) hoặc this_is_variable (gọi là Snake Case).
- Không nên viết in hoa hết cả tên biến (chữ viết in hoa thường dành cho hằng số, macro, define).
- Nên có tiền tố cho biến

s8_	8-bit signed integer
u8_	8-bit unsigned integer
s16_	16-bit signed integer
u16_	16-bit unsigned integer
s32_	32-bit signed integer
u32_	32-bit unsigned integer
s64_	64-bit signed integer
u64_	64-bit unsigned integer
f_	float
d_	double
b_	boolean
stu_	structure
e_	enumeration
str_	string
c_	character

- Thêm chữ a nếu là mảng

as8_	Array of 8 bit signed integer
au32_	Array of 32 bit unsigned integer
ad_	Array of double

- Thêm chữ p nếu là con trỏ

pu16_	Pointer to 16 bit unsigned integer
ps64_	Pointer to 64 bit signed integer
pv_	Pointer to void
pstu_	Pointer to structure
pstr_	Pointer to string

- Biến toàn cục nên có tiền tố là g_ để biết rằng đang sử dụng biến toàn cục.
- Kiểu dữ liệu nên sử dụng một số kiểu dữ liệu đã được định sẵn chiều dài.

	signed	**unsigned**
8-bit	int8_t	uint8_t
16-bit	int16_t	uint16_t
32-bit	int32_t	uint32_t
64-bit	int64_t	uint64_t

Hàm

- Một hàm không nên quá dài. Có thể chia thành nhiều hàm nhỏ để dễ quản lý. Rất nhiều bạn khi mới lập trình là viết tất cả mọi

thứ trong hàm main(), điều này sẽ rất khó kiểm soát chương trình. Vì thế bạn nên chia nhỏ chương trình bằng cách viết các chương trình con.
- Nên comment thông tin của hàm, các biến đầu vào phía trên hàm đó.

Ngắt

- Luôn giữ cho ngắt được thực thi thật ngắn.

Comment

- Nên được viết bằng tiếng anh.
- Viết ngắn gọn, đơn giản, xúc tích.

Một vài nguyên tắc chung

- Một hàng code nên tối đa là 80 ký tự và không vượt quá 120 ký tự.
- Không sử dụng đường dẫn tuyệt đối khi include header files.
- Không bao giờ sử dụng một con số mà không có ý nghĩa hoặc không có công thức gì cả (magic number). Nếu nó là hằng số thì hãy biến nó thành hằng số hoặc define và nên giải thích rõ ràng là tại sao là giá trị đó. Magic number thường hay gặp khi bạn cân chỉnh (calib) cảm biến, thuật toán để cho nó chạy. Khi nó chạy rồi thì bạn không hiểu tại sao "số đó" lại xuất hiện trong code.
- Nên sử dụng space thay vì tab.
- Đặt dấu cách sau mỗi dấu "," hoặc ";".
- Luôn đặt code được bao bọc bởi dấu {} sau mỗi từ khóa if, else, switch, do, while, for.
- Đối với các if lồng nhau thì không nên có độ sâu quá 3 điều kiện. Nếu bắt buộc có nhiều điều kiện lồng nhau thì nên chuyển thành cách code khác, ví dụ như switch...case.
- Nên sử dụng const bất cứ khi nào có thể.
- Nên dùng biến static để giới hạn vùng hoạt động của nó.
- Không được so sánh biến có dấu với biến không dấu.
- Nên khai báo giá trị cho tất cả các biến trước khi sử dụng.

Trên đây chỉ là một vài nguyên tắc cơ bản, bạn có thể tìm hiểu thêm và chọn cho mình một phong cách. Nếu sau này bạn tham gia dự án nào thì bạn phải theo conding convention của team, nhưng nếu bạn đã nắm cơ bản thì việc tiếp cận cái mới sẽ nhanh thôi.

MÁY TRẠNG THÁI 3

3.1 Giới thiệu

"Máy trạng thái hữu hạn (finite-state machine FSM) hoặc Máy tự động trạng thái hữu hạn (finite-state automaton FSA), hoặc là máy tự động hữu hạn, hoặc gọi đơn giản là máy trạng thái, là một mô hình tính toán toán học. Nó là một máy trừu tượng luôn có trạng thái nằm trong tổng hữu hạn các trạng thái tại bất kỳ thời điểm nào. Máy trạng thái hữu hạn có thể chuyển từ trạng thái này sang trạng thái khác để phù hợp với đầu vào; sự thay đổi này được gọi là quá trình chuyển đổi. Máy trạng thái hữu hạn được xác định bởi danh sách các trạng thái của nó, trạng thái khởi đầu, và các điều kiện cho từng sự chuyển đổi trạng thái."[1]

1: Trích dẫn từ Wikipedia

3.2 Máy trạng thái cho vi điều khiển

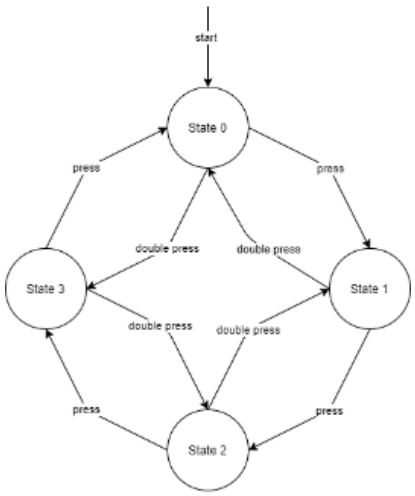
Khi chương trình của bạn có thể chia nhỏ thành nhiều trạng thái khác nhau hoặc chương trình của bạn chạy theo một chu trình nào đó thì bạn có thể áp dụng máy trạng thái.

Để thực hiện máy trạng thái một cách đơn giản, chúng ta chỉ cần dùng switch…case trong ngôn ngữ C/C++.

Chúng ta cùng xem xét các ví dụ để hiểu rõ hơn về máy trạng thái.

Ví dụ 1

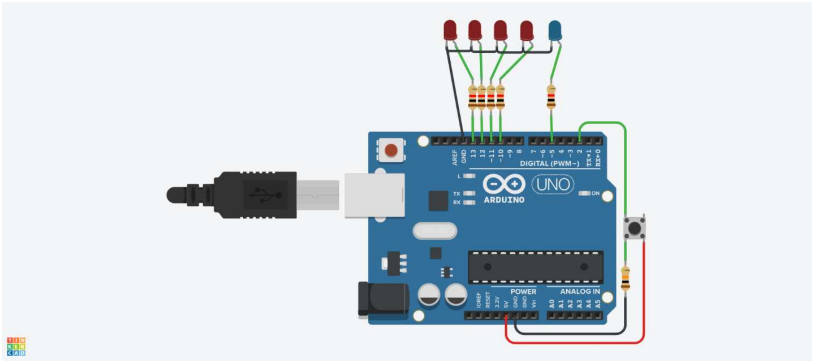
Ví dụ chúng ta có một chương trình gồm 4 trạng thái đơn giản gồm State 0..3. Chương trình sẽ thay đổi trạng thái khi bấm nút. Giá trị trạng thái sẽ tăng thêm 1 nếu nút nhấn được nhấn 1 lần, giá trị trạng thái sẽ giảm đi một nếu nút nhấn được nhấn 2 lần liên tiếp.



Hình 3.1: Sơ đồ máy trạng thái.

Bây giờ chúng ta sẽ thực hiện chương trình này trên Arduino. [2]

2: Bạn có thể mô phỏng chương trình tại https://www.tinkercad.com/things/dZWv2K1kDO5



Hình 3.2: Mạch Arduino ví dụ cho máy trạng thái.

Phần cứng:

- Mạch điện gồm 4 đèn LED đỏ để thay đổi trạng thái tương ứng với từng chế độ.
- Đèn xanh lá dùng để nhận biết có chương trình ngắt nút nhấn. Mỗi lần vào ngắt thì sẽ thay đổi trạng thái đèn.
- Nút nhấn dùng để thay đổi trạng thái chương trình.

Định nghĩa trạng thái

Tên trạng thái	Mô tả
State 0 | MODE_0	Hai đèn tại chân 11 và 13 sẽ nhấp nháy
State 1 | MODE_1	Hai đèn tại chân 10 và 12 sẽ nhấp nháy
State 2 | MODE_2	Đèn sẽ chạy từ phải sang trái
State 3 | MODE_3	Đèn sẽ chạy từ trái sang phải

Chương trình

```
// Author: minht57
// Date: Jul 25 2020

int buttonState = 0;

#define MODE_0 0
```

```
#define MODE_1 1
#define MODE_2 2
#define MODE_3 3

#define TIME_2_PRESSED        500

volatile byte machine_state = 0;
volatile unsigned long start_time = 0;
volatile byte press_twice = 0;
volatile byte button_pressed = 0;

volatile byte isr_state = 0;

void ext_isr_0()
{
  if (button_pressed && (millis() - start_time < TIME_2_PRESSED))
  {
    press_twice = 1;
  }

  button_pressed = 1;
  start_time = millis();

  isr_state = ~isr_state;
  digitalWrite(5,isr_state);
}

void setup()
{
  pinMode(2, INPUT);
  pinMode(10, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);
  pinMode(5, OUTPUT);
  Serial.begin(9600);

  attachInterrupt(digitalPinToInterrupt(2), ext_isr_0, FALLING);
}

void loop()
{
  if (button_pressed)
  {
    if(press_twice)
    {
      press_twice = 0;
      button_pressed = 0;
      machine_state = (4 + machine_state - 1) % 4;
    }
    else if (millis() - start_time >= TIME_2_PRESSED)
    {
      button_pressed = 0;
      machine_state = (machine_state + 1) % 4;
    }
  }

  Serial.println(start_time);

  int idx = 0;
```

```
67  switch(machine_state)
68  {
69    case MODE_0:
70      digitalWrite(10,LOW);
71      digitalWrite(11,HIGH);
72      digitalWrite(12,LOW);
73      digitalWrite(13,HIGH);
74      delay(200);
75      digitalWrite(10,LOW);
76      digitalWrite(11,LOW);
77      digitalWrite(12,LOW);
78      digitalWrite(13,LOW);
79      delay(200);
80    break;
81    case MODE_1:
82      digitalWrite(10,HIGH);
83      digitalWrite(11,LOW);
84      digitalWrite(12,HIGH);
85      digitalWrite(13,LOW);
86      delay(200);
87      digitalWrite(10,LOW);
88      digitalWrite(11,LOW);
89      digitalWrite(12,LOW);
90      digitalWrite(13,LOW);
91      delay(200);
92    break;
93    case MODE_2:
94      for(idx = 10; idx < 14; idx++)
95      {
96        digitalWrite(idx, HIGH);
97        delay(50);
98      }
99      for(idx = 10; idx < 14; idx++)
100     {
101       digitalWrite(idx, LOW);
102       delay(50);
103     }
104   break;
105   case MODE_3:
106     for(idx = 13; idx >= 10; idx--)
107     {
108       digitalWrite(idx, HIGH);
109       delay(50);
110     }
111     for(idx = 13; idx >= 10; idx--)
112     {
113       digitalWrite(idx, LOW);
114       delay(50);
115     }
116   break;
117 }
118 }
```

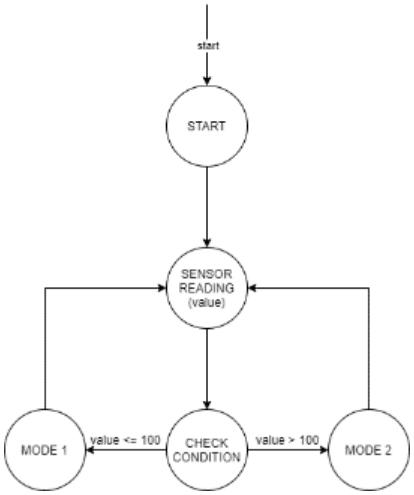
- Hàng 20..32: đây là chương trình ngắt ngoài để xác nhận nút nhấn được nhấn một lần hay hai lần.
- Hàng 49..62: đây là đoạn chương trình thay đổi giá trị trạng thái của máy trạng thái. Nếu nhấn một lần thì biến trạng thái sẽ tăng lên một. Nếu nhấn hai lần liên tiếp thì giảm biến trạng thái cho

một.

- Hàng 67..117: định nghĩa chương trình của các trạng thái.

Ví dụ 2

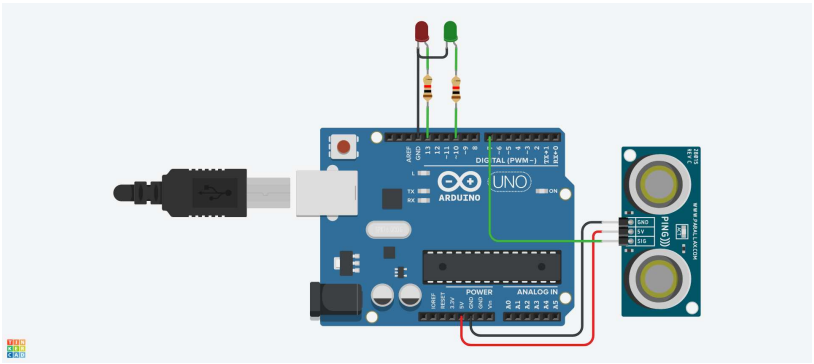
Ví dụ chúng ta có một cảm biến siêu âm đo khoảng cách để tránh vật cản. Nếu cảm biến đọc được khoảng cách nhỏ hơn một giá trị đặt trước thì thông báo có vật cản, ngược lại thì không có vật cản.



Hình 3.3: Sơ đồ máy trạng thái.

Bây giờ chúng ta sẽ thực hiện chương trình này trên Arduino. [3]

3: Bạn có thể mô phỏng chương trình tại https://www.tinkercad.com/things/k52C4ZDr3KI



Hình 3.4: Mạch Arduino ví dụ cho máy trạng thái.

Phần cứng

- Đèn LED màu đỏ thông báo nếu có vật cản. Đèn LED màu xanh dùng để thông báo không có vật cản.
- Cảm biến đo khoảng cách.

Định nghĩa trạng thái

Tên trạng thái	Mô tả
State 0 START	Trạng thái bắt đầu, chuyển sang trạng thái SENSOR_READING
State 1 SENSOR_READING	Đọc giá trị cảm biến khoảng cách, chuyển sang trạng thái CHECK_CONDITION
State 2 CHECK_CONDITION	Nếu khoảng cách nhỏ lớn hơn 100cm thì chuyển sang MODE_2, ngược lại thì chuyển sang MODE_1
State 3 MODE_1	Thực hiện chương trình khi có vật cản (bật LED đỏ) và quay lại trạng thái SENSOR_READING
State 4 MODE_2	Thực hiện chương trình khi không có vật cản (bật LED xanh) và quay lại trạng thái SENSOR_READING

Chương trình

```
// Author: minht57
// Date: Jul 25 2020

#define START          0
#define READ_SENSOR    1
#define CHECK_CONDITION 2
#define MODE_1         3
#define MODE_2         4

int distance = 0;
byte state = START;

long readUltrasonicDistance(int triggerPin, int echoPin)
{
  pinMode(triggerPin, OUTPUT);  // Clear the trigger
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(2);
  // Sets the trigger pin to HIGH state for 10 microseconds
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);
  pinMode(echoPin, INPUT);
  // Reads the echo pin, and returns the sound wave travel time in microseconds
  return pulseIn(echoPin, HIGH);
}

void fsm()
{
  switch(state)
  {
    case START:
      Serial.println("Start!!!");
      state = READ_SENSOR;
    break;
    case READ_SENSOR:
      // measure the ping time in cm
      //Serial.println("Sensor reading...");
            distance = 0.01723 * readUltrasonicDistance(7, 7);
      state = CHECK_CONDITION;
    break;
    case CHECK_CONDITION:
      if(distance > 100)
```

```
43        {
44          state = MODE_2;
45        }
46        else
47        {
48          state = MODE_1;
49        }
50      break;
51      case MODE_1:
52        // Do action
53        Serial.println("\t\tWarning!!! Obstacle!!!");
54        digitalWrite(13, HIGH);
55        digitalWrite(10, LOW);
56        state = READ_SENSOR;
57      break;
58      case MODE_2:
59        Serial.println("Free to move...");
60        digitalWrite(13, LOW);
61        digitalWrite(10, HIGH);
62        state = READ_SENSOR;
63      break;
64    }
65  }
66
67  void setup()
68  {
69    Serial.begin(9600);
70    pinMode(13, OUTPUT);
71    pinMode(10, OUTPUT);
72  }
73
74  void loop()
75  {
76    fsm();
77    delay(100); // Wait for 100 millisecond(s)
78  }
```

- Hàng 13..25: hàm này dùng để đọc giá trị khoảng cách.
- Hàng 27..65: đây là chương trình kiểm tra và thực hiện máy trạng thái. Việc đổi trạng thái được thực hiện trực tiếp trong từng trạng thái.

3.3 Tổng kết

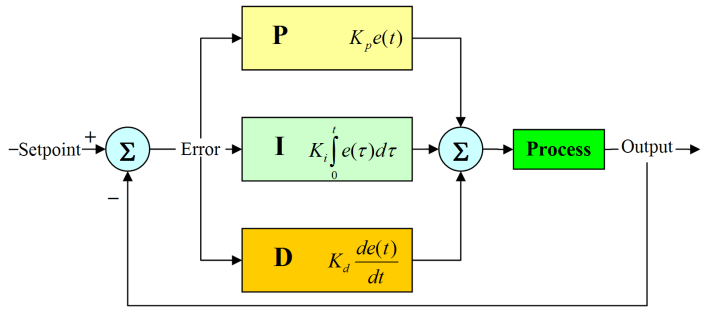
Máy trạng thái sẽ cho bạn một cái nhìn tổng quát hơn về chương trình, giúp bạn kiểm soát code tốt hơn khi chương trình bạn càng lớn.

BỘ ĐIỀU KHIỂN PID 4

4.1 Giới thiệu

Bộ điều khiển PID là bộ điều khiển tự động dùng để đưa ra giá trị điều khiển tới phần tử chấp hành để cho ngõ ra gần đúng với ngõ vào nhiều nhất với các yêu cầu về đáp ứng có thể chấp nhận được. Các yêu cầu về mặt đáp ứng như sai số giữa ngõ ra và ngõ vào, tốc độ đáp ứng, độ vọt lố, sự dao động của tín hiệu ngõ ra.

Bộ điều khiển PID là sự kết hợp của 3 khâu tỉ lệ (Proportional), khâu tích phân (Integral) và khâu vi phân (Derivative).



Hình 4.1: Bộ điều khiển PID.

- Khâu tỉ lệ (P) là tín hiệu ngõ ra tỷ lệ thuận với tín hiệu ngõ vào với hệ số là Kp.
- Khâu tích phân (I) là tổng tất cả các sai số từ thời điểm bắt đầu đến thời điểm hiện tại nhận với hệ số Ki.
- Khâu vi phân (D) là tốc độ thay đổi của sai số tại thời điểm t.

Một số đặc tính khi thay đổi hệ số Kp, Ki, Kd:

- Khâu tỉ lệ:
  - Kp càng lớn thì tốc độ đáp ứng càng nhanh, sai số xác lập càng nhỏ (nhưng không bao giờ bằng 0), hệ thống dao động càng lớn, độ vọt lố càng cao. Nếu Kp tăng quá giới hạn thì hệ thống sẽ mất ổn định.
- Khâu tích phân:
  - Ki càng lớn thì đáp ứng quá độ càng chậm, sai số xác lập càng nhỏ, độ vọt lố càng cao.
- Khâu vi phân:
  - Kd càng lớn thì đáp ứng quá độ càng nhanh, độ vọt lố càng nhỏ, nhiễu tần số cao.

4.2 Thư viện

Tác giả Brett Beauregard đã có một gói thư viện PID chỉ cần cài đặt và chạy.[5]

5: Link: https://playground.arduino.cc/Code/PIDLibrary

Lớp PID

Giới thiệu: tạo đối tượng PID với con trỏ ngõ vào (input), con trỏ ngõ ra (output), con trỏ giá trị đặt (setpoint).

Cú pháp:

```
PID(&Input, &Output, &Setpoint, Kp, Ki, Kd, Direction)
PID(&Input, &Output, &Setpoint, Kp, Ki, Kd, POn, Direction)
```

- Input: tín hiệu vào (kiểu dữ liệu là double)
- Output: tín hiệu ra (kiểu dữ liệu là double)
- Setpoint: giá trị mong muốn (kiểu dữ liệu là double)
- Kp, Ki, Kd: Độ lợi từng khâu. (kiểu dữ liệu là double>=0)
- Direction: DIRECT hoặc REVERSE. Xác định hướng của ngõ ra đối với ngõ vào. Nếu thông số này là REVERSE thì dấu của Kp, Ki, Kd sẽ đổi dấu. Thông thường sẽ cấu hình là DIRECT.
- POn: P_ON_E (mặc định) hoặc P_ON_M. Đây là tính năng Proportional on Measurement (đọc trang blog của tác giả để biết thêm thông tin chi tiết).

Hàm Compute()

Giới thiệu: dùng để tính toán thuật toán PID. Hàm này nên gọi ở mỗi vòng loop(). Hàm sẽ chỉ tính toán với tần số được khai báo ở hàm SetSampleTime().

Cú pháp:

```
Compute()
```

Kết quả trả về:

- True: nếu output được tính toán
- False: output không được tính toán

Hàm SetMode()

Giới thiệu: dùng để bật/tắt module PID.

Cú pháp:

```
SetMode(mode)
```

- mode: AUTOMATIC (bật) or MANUAL (tắt)

Kết quả trả về: không.

Hàm SetTunings()

Giới thiệu: thay đổi độ lợi các khâu trong khi đang chạy (run-time).

Cú pháp:

```
SetTunings(Kp, Ki, Kd)
SetTunings(Kp, Ki, Kd, POn)
```

- Kp, Ki, Kd: bộ thông số muốn thay đổi.
- POn: P_ON_E (mặc định) or P_ON_M.

Giá trị trả về: không.

Hàm SetControllerDirection()

Giới thiệu: **Nếu giá trị ngõ vào (input) cao hơn giá trị đặt (setpoint) thì kết quả ngõ ra nên tăng hay giảm?** Kết quả là sẽ phụ thuộc vào loại đối tượng mà bạn đang điều khiển. Ví dụ là điều khiển xe, nếu ngõ vào đang lớn hơn giá trị đặt thì chúng ta cần giảm tốc. Đối với tủ lạnh, nếu giá trị nhiệt độ ngõ vào lớn hơn giá trị đặt thì chúng ta cần tăng tốc độ điều khiển bộ phần làm lạnh để nhiệt độ giảm xuống. Trong hai ví dụ trên, điều khiển xe được cấu hình là DIRECT, tủ lạnh là REVERSE.

Cú pháp:

```
SetControllerDirection(Direction);
```

- Direction: DIRECT hoặc REVERSE

Kết quả trả về: không.

Một số hàm yêu cầu giá trị cấu hình từ module PID

Giới thiệu: yêu cầu các giá trị cấu hình hiện tại của module PID.

Cú pháp:

```
GetKp()
GetKi()
GetKd()
GetMode()
GetDirection()
```

Kết quả trả về: tương ứng với kiểu biến trả về.

Ví dụ

```
1  /********************************************************
2   * PID Basic Example
3   * Reading analog input 0 to control analog PWM output 3
4   ********************************************************/
5
6  #include <PID_v1.h>
7
8  //Define Variables we'll be connecting to
9  double Setpoint, Input, Output;
10
11 //Specify the links and initial tuning parameters
12 PID myPID(&Input, &Output, &Setpoint,2,5,1, DIRECT);
13
14 void setup()
15 {
16   //initialize the variables we're linked to
17   Input = analogRead(0);
18   Setpoint = 100;
19
20   //turn the PID on
21   myPID.SetMode(AUTOMATIC);
22 }
23
24 void loop()
25 {
26   Input = analogRead(0);
27   myPID.Compute();
28   analogWrite(3,Output);
29 }
```

- Hàng 9: khai báo biến giá trị đặt/giá trị mong muốn (setpoint), biến ngõ vào (input), biến ngõ ra (output).
- Hàng 12: khai báo đối tượng PID với con trỏ các biến vào/ra/giá trị đặt, bộ thông số Kp, Ki, Kd.
- Hàng 17: khởi tạo giá trị đầu tiên cho biến input.
- Hàng 18: cài đặt giá trị mong muốn.
- Hàng 21: bật PID.

- Hàng 26: đọc tín hiệu feedback.
- Hàng 27: tính toán ngõ ra từ ngõ vào. Hàm này sẽ tự động lấy giá trị từ các con trỏ đã gán ở bước khởi tạo. Khi tính toán xong thì giá trị sẽ được tự động lưu vào biến Output.
- Hàng 28: ghi giá trị điều khiển vào đối tượng. Đối tượng ở đây là module PWM xuất ra giá trị điện áp có thể thay đổi được.

4.3 Một số lưu ý

Trước khi sử dụng bộ điều khiển PID, bạn cần xác định một số thông tin cần thiết sau:

- Ngưỡng hoạt động của ngõ vào và ngõ ra.
- Xác định tầm giá trị của hệ số độ lợi các khâu.

Xác định mục đích sử dụng của đối tượng:

- Ví dụ cùng là điều khiển động cơ DC, nhưng nếu bạn điều khiển vị trí và điều khiển vận tốc thì cần bộ điều khiển hơi khác nhau.
- Ví dụ khi bạn điều khiển vị trí, bạn mong muốn động cơ quay 720 độ từ vị trí hiện tại. Qua độ điều khiển PID thì giá trị điều khiển u của bạn sẽ tăng lên rồi giảm về 0 khi động cơ bạn đã đủ 720 độ. Nghĩa là khi đạt được tới giá trị mong muốn thì ngõ ra là bằng 0.
- Ngược lại khi bạn điều khiển vận tốc, bạn mong muốn động cơ quay với tốc độ 50rpm (rounds per minute – vòng trên phút). Khi tốc độ ngõ ra đạt tới giá trị mong muốn thì giá trị điều khiển u của bạn phải được giữ để đảm bảo tốc độ động cơ không đổi.

BỘ LỌC KALMAN CHO IMU 5

5.1 Giới thiệu

Inertial Measurement Unit (đơn vị đo quán tính - IMU) là một cảm biến thông dụng trong việc đo chuyển động của vật thể. IMU có thể đo được nhiều thông số khác nhau như vận tốc, hướng, gia tốc, lực, tốc độ góc hoặc từ trường tùy thuộc vào số lượng cảm biến trên IMU. Trên IMU có thể có các cảm biến sau:

- Accelerometer: đo gia tốc chuyển động theo 3 trục XYZ.
- Gyroscope: đo vận tốc xoay quanh trục XYZ.
- Magnetometer: đo từ trường theo 3 trục XYZ.
- Barometer: đo áp suất không khí.

Degree of Freedom (DOF) là bậc tự do. Ví dụ một IMU gồm cảm biến accel và gyro thì người ta gọi là 6-DOF IMU, một IMU gồm accel + gyro + mag thì được gọi là 9-DOF IMU,...

Các ứng dụng thường thấy là dùng IMU để đo góc theo 3 trục của vật thể hoặc kết hợp với cảm biến khác để bổ sung tính đúng đắn cho kết quả. Vì IMU khá là nhạy với nhiễu hoặc bị trôi theo thời gian, nên ta có thể kết hợp nhiều cảm biến lại với nhau để cho một kết quả tốt hơn (gọi là sensor fusion).

Kalman là một bộ ước lượng trạng thái khá nổi tiếng và việc dùng Kalman để ước lượng góc thường được nhắc đến. Ý tưởng của bộ lọc Kalman có thể tóm tắt như sau: [7]

- Giả sử bạn biết được mô hình chuyển động (motion model) và mô hình đo (measurement model). Đồng thời bạn cũng biết được nhiễu hệ thống (process noise) và nhiễu đo (measurement noise).
- Bạn biết được trạng thái hiện tại của bạn. Dựa vào mô hình chuyển động thì bạn sẽ suy đoán được trạng thái tiếp theo.
- Từ trạng thái ước lượng tiếp theo thì bạn sẽ ước lượng được giá trị trả về thông qua mô hình đo.
- Bạn sẽ tính toán ra được sự thay đổi của kết quả thực tế cảm biến thực tế trả về với giá trị ước lượng.
- Cập nhật giá trị trạng thái hiện tại bằng độ lợi Kalman (Kalman gain). Giá trị của độ lợi Kalman sẽ được tự động tính toán, giá trị này thể hiện việc bộ lọc Kalman đang tin vào giá trị ước lượng hơn hay là tin vào giá trị thực tế đo được hơn.

7: Chi tiết hơn bạn có thể đọc tại: http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-imp

5.2 Thư viện

Đây là một bộ thư viện Kalman filter được Kristian Lauszus phát triển. Bộ thư việc có thể được dùng cho bất kỳ vi xử nào và tác giả đã dùng Arduino để kiểm chứng kết quả.[8] [9]

8: Link thư viện: https://github.com/TKJElectronics/KalmanFilter

9: Code mẫu dành cho Arduino: https://github.com/TKJElectronics/KalmanFilter/blob/master/examples/MPU6050/MPU6050.ino

5.3 Một số thông tin cần biết

Bộ lọc Kalman dùng để ước lượng trạng thái rất tốt nhưng khối lượng tính toán khá là nhiều thì có thể gây ảnh hưởng đến những phần còn lại của chương trình vì thế bạn nên cân nhắc khi sử dụng bộ lọc Kalman.

Ngoài bộ lọc Kalman thì bạn có thể sử dụng bộ lọc bù (Complimentary Filter) để thay thế. Nhờ vào việc tính toán đơn giản mà kết quả chấp nhận được thì bạn có thể xem xét.

$$angle = \alpha(angle + data_{gyro} * dt) + (1 - \alpha) * data_{acc}$$

- α là hằng số để cân bằng giữa thông tin từ gyro và acc. α thường bằng 0.98 và có thể điều chỉnh tùy vào hệ thống.
- **data_gyro** là vận tốc góc được lấy từ gyro (đơn vị rad/s).
- **data_acc** là góc được tính từ acc (rad). Ta có thể tính góc từ acc bằng công thức sau:[10]
  - $roll = atan2(accY, accZ) * RAD\_TO\_DEG;$
  - $pitch = atan(-accX/sqrt(accY * accY + accZ * accZ)) * RAD\_TO\_DEG;$

10: Link tham khảo: http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf

Đặc tính của cảm biến acc và gyro:

- Giá trị trả về của cảm biến acc được thông qua tác dụng về lực nên cảm biến acc khá nhạy với những sự rung lắc (vibration). Do đó cảm biến acc khá là nhạy với nhiễu tần số cao. Bên cạnh đó, góc có thể được tính trực tiếp từ giá trị của acc nên giá trị đáng tin cậy. Vì vậy, giá trị góc tính ra từ giá trị acc đáng tin cậy trong một khoảng thời gian dài (long-time). Để khắc phục nhiễu tần số cao thì có thế áp dụng bộ lọc thông thấp.
- Khi tính toán góc từ gyro thì giá trị rất đáng tin cậy trong một khoảng thời gian ngắn (short-time) vì cảm biến gyro không chịu tác động của lực. Nhưng để tính toán được góc thì chúng ta cần tích phân giá trị của cảm biến theo thời gian nên sẽ gây hiện tượng trôi (drift). Vì vậy, giá trị tính góc từ gyro có độ tin cậy trong khoảng thời gian ngắn.
- Với đặc tính của hai cảm biến như vậy, chúng ta kết hợp cả hai cảm biến với nhau để có giá trị góc ước lượng chính xác hơn.

Nếu bạn muốn ước lượng góc yaw của vật thể thì bạn nên kết hợp với magnetometer để có góc quay chính xác.

Nếu muốn đo độ cao thì cảm biến barometer kết hợp với cảm biến nhiệt độ để cân chỉnh lại giá trị cho chính xác hơn. Vì áp suất khí quyển khác nhau từng vùng nên bạn cần cân chỉnh (calibrate) tại vị trí nơi bạn sử dụng.

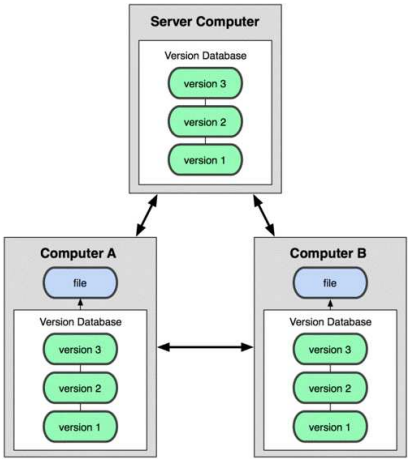
Bạn cũng nên tìm hiểu về các cách cân chỉnh (calibrate) các cảm biến trong IMU.

CODE VERSION/GIT 6

6.1 Giới thiệu [12]

"Git là tên gọi của một Hệ thống quản lý phiên bản phân tán (Distributed Version Control System – DVCS) là một trong những hệ thống quản lý phiên bản phân tán phổ biến nhất hiện nay. DVCS nghĩa là hệ thống giúp mỗi máy tính có thể lưu trữ nhiều phiên bản khác nhau của một mã nguồn được nhân bản (clone) từ một kho chứa mã nguồn (repository), mỗi thay đổi vào mã nguồn trên máy tính sẽ có thể ủy thác (commit) rồi đưa lên máy chủ nơi đặt kho chứa chính. Và một máy tính khác (nếu họ có quyền truy cập) cũng có thể clone lại mã nguồn từ kho chứa hoặc clone lại một tập hợp các thay đổi mới nhất trên máy tính kia. Trong Git, thư mục làm việc trên máy tính gọi là Working Tree."

12: Bài viết được trích dẫn tại csc.edu.vn



Hình 6.1: Mô hình tổ chức dữ liệu phân tán trên Git.

Github là một dịch vụ máy chủ repository cho phép người dùng tạo tài khoản để tạo ra các kho chứa cá nhân để làm việc. Github hoạt động theo mô hình hệ thống Git.

6.2 Những lợi ích khi sử dụng Git

- Lưu lại được những phiên bản trong quá trình phát triển phần mềm và có thể khôi phục lại một vị trí bất kỳ.
- Dễ sử dụng, an toàn, tiện lợi, nhanh chóng.
- Có thể làm việc với nhiều thành viên cùng một lúc ở nhiều nơi khác nhau.
- Dễ dàng trong việc phát triển phần mềm khi hệ thống phần lớn.
- Dễ dàng kiểm soát code, review code.
- ...

6.3 Những đường dẫn bạn nên đọc để tìm hiểu về Git

Giải thích về mô hình hoạt động của git.

- Link: https://csc.edu.vn/lap-trinh-va-csdl/tin-tuc/kien-thuc-lap-trinh/Git-la-gi--Nhung-khai-niem-co-ban-khi-lam-viec-tren-Git-4133.

Học git từ cơ bản đến nâng cao

- Link 1: https://sociss.edu.vn/lessons/tu-hoc-git-tu-co-ban-den-nang-cao-phan-1
- Link 2: https://sociss.edu.vn/lessons/tu-hoc-git-tu-co-ban-den-nang-cao-phan-2

Thực hành với git (online)

- Link: https://learngitbranching.js.org/

PHÁT TRIỂN SẢN PHẨM 7

Hướng tiếp cận để phát triển sản phẩm cho các bạn tự học Lưu ý: bài viết này mang tính chất cá nhân và quan điểm riêng của mỗi cá nhân. Hãy chia sẻ và học hỏi để tìm ra phương pháp nào phù hợp nhất cho mình.

7.1 Suy nghĩ

Trước khi bạn làm bất cứ điều gì thì hãy luôn tìm cho mình một phương pháp, một hướng tiếp cận để bạn có thể đi đúng hướng và không bị lang mang khi gặp rắc rối.

Hãy luôn đặt tự đặt câu hỏi với bản thân và tự tìm ra câu trả lời để bạn có thể học nhiều nhất có thể chứ không hẳn là cái sản phẩm bạn cầm trên tay. Hãy đặt các câu hỏi "Cái gì", "Khi nào", "Làm thế nào", "Ở đâu", "Tại sao"... Một số ví dụ

- Khi bạn gặp một khái niệm, một thiết bị/linh kiện mới: cái này là cái gì?
- Khi bạn muốn gặp một xung nhiễu trên nút nhấn: Khi nào xung nhiễu xuất hiện? Tại sao lại có xung nhiễu ở đây? Cần giải quyết như thế nào?
- Khi bạn gặp một lỗi lập trình: Bug đang ở đâu? Phần nào có khả năng gặp bug nhiều nhất? Làm sao để tìm ra bug? Làm sao để fix bug?
- ...

Hầu như tất cả vấn đề bạn gặp phải đều có câu trả lời trên Google, vấn đề đặt ra là làm sao bạn tìm được đúng câu trả lời bạn mong muốn? Hãy tìm ra phương pháp đặt câu hỏi cho Google hoặc cách đặt vấn đề trên một diễn đàn.

- Nếu một vấn đề mà bạn không có bất kỳ một từ khóa (keyword) hay khái niệm (defination) nào, hãy mô tả cái bạn mong muốn cho google (đừng mong có câu trả lời ngay mà hãy tìm từ khóa liên quan) để bạn tìm những từ khóa đúng cho vấn đề của bạn. Sau khi đã có một vài từ khóa, khái niệm về vấn đề của bạn, hãy đặt lại câu hỏi với google với từ khóa đó, kết quả gần với vấn đề của bạn được hiển thị nhiều hơn.
- Nếu được thì bạn nên tìm kiếm bằng tiếng anh, kết quả trả về sẽ nhiều hơn rất nhiều và tài liệu chính xác nhiều hơn.
- Khi bạn hỏi một vấn đề trên các nền tảng xã hội: hãy mô tả đơn giản cái bạn đang làm, bạn đang gặp vấn đề ở đâu, bạn đang nghĩ nó bug ở chỗ nào, nên kèm hình ảnh, thông báo lỗi hoặc kể cả code nếu cần,... để người giúp đỡ sẽ nắm bắt vấn đề nhanh chóng mà có thể giúp bạn. Đừng đặt các câu hỏi chung chung ("Em đang lập trình con STM32F407, em đang gặp bug ở ngắt

nút nhấn. Có sư phụ nào chỉ em bug ở đâu không?"), ai tốt thì cũng sẽ trả lời cho bạn những cách giải quyết rất chung chung. Nếu như thế thì con đường của bạn đến câu trả lời sẽ hơi mất thời gian rất nhiều.

7.2 Ý tưởng đến kế hoạch

Khi bạn làm gì thì cũng cần có một kế hoạch. [14]

14: Đây là cách làm của một cá nhân, nó mang tính chất kinh nghiệm. Có thể đúng, có thể chưa đúng, nhưng nó cũng là một cách làm. Hãy tìm cái hay để học hỏi, bỏ qua cái không hay hoặc không phù hợp.

Mình giả sử bạn biết được mong muốn của mình là gì và sản phẩm sẽ hoạt động như thế nào. Hãy ngồi xuống và vẽ nó ra giấy. Bạn hình dung sản phẩm của mình hoạt động như thế nào thì hãy vẽ sơ đồ khối như thế ấy. Liệt kê các thiết bị, các module cần thiết cho hoạt động. Hãy phát thảo sơ mọi thứ từ cơ khí, điện tử, mạch điện, phần mềm.

Ví dụ, mình muốn làm một chiếc xe đi theo line.

- Phân tích: chiếc xe theo line.
  - Chúng ta cần một chiếc xe, mà xe thông dụng đơn giản hiện nay là xe điều khiển bởi 2 động cơ và một bánh xe tự do (xe có ba bánh).
  - Line: vậy chúng ta phải cần một cái gì đó để phát hiện được cái line (→ cảm biến). Nhiều gợi ý trên mạng là dùng cảm biến hồng ngoại để phát hiện được line.
  - Theo: khi đã có chiếc xe và một dãy cảm biến hồng ngoại để phát hiện xe còn trong line hay không, ta cần thuật toán để đi theo cái line đó. Nếu đơn giản thì chỉ cần if…else: nếu xe còn trong line thì chỉ cần đi thẳng; nếu xe lệch khỏi line bên trái thì quẹo phải để tìm lại line; nếu xe lệch khỏi line bên phải thì quẹo trái để tìm lại line. Nếu phức tạp hơn thì có thể dùng PID để điều khiển bám line.
- Cơ khí và thiết bị
  - Khung xe.
  - 2 động cơ. Mà để điều khiển được động cơ thì cần cầu H.
  - Hệ thống các đèn hồng ngoại.
  - Pin.
  - Mạch Arduino để điều khiển xe.
  - Công tắc, trụ đỡ, dây điện,…
- Phần lập trình
  - Thuật toán cần ngõ là giá trị của dãy cảm biến, ngõ ra là 2 tín hiệu điều khiển động cơ.
  - Chọn cảm biến loại digital thì chỉ cần đọc digital.
  - Cần module PWM để điều chỉnh tốc độ động cơ.
  - Liệt kê sơ phần hoạt động của xe: xe chạy thẳng thì cấp xung 2 động cơ như nhau; nếu rẻ trái thì động cơ bên phải chạy nhanh hơn; nếu rẻ phải thì động cơ bên trái chạy nhanh hơn.

Sau khi liệt kê hết tất cả các phần cần làm, hiểu sơ hoạt động của hệ thống như thế nào. Đơn giản phía trên chỉ là ý tưởng, phát thảo ra những gì bạn nghĩ, hiểu biết về sản phẩm bạn muốn làm.

Đến đây, hãy lập kế hoạch cho sản phẩm. Bạn hãy suy nghĩ về số lượng thành viên, năng lực từng thành viên, mục đích thực hiện sản phẩm, và thời gian bạn thực hiện thì đưa ra một kế hoạch sơ lược về ai làm việc gì, mỗi phần nào làm trong bao lâu và làm sao để các thành viên hợp tác với nhau để có thể cho luồng công việc mượt mà. Nếu bạn làm một mình thì đơn giản thôi → ÔM HẾT.

7.3 Bắt đầu làm và giải quyết vấn đề

Khi đã có kế hoạch thì hãy bắt tay làm mọi thứ. Thông thường thì không có cái gì theo kế hoạch cả vì có nhiều vấn đề bạn không thể lường trước được. Khi gặp vấn đề, hãy bình tĩnh và tìm hướng giải quyết vấn đề.

Khi gặp vấn đề về cơ khí: hãy xem xét lại vấn đề bạn đang gặp và so sánh với thiết kế (trên giấy hoặc trong đầu) để tìm ra làm sao cải thiện, sửa đổi thiết kế như thay đổi cấu trúc frame/chasis cho phù hợp, gắn thêm đế,...

Khi gặp vấn đề về điện: hãy đọc kỹ các tài liệu liên quan về IC, module,... Hãy luôn cấu hình đúng mức điện áp, nối đúng dây,... Nếu gặp vấn đề về nhiễu thì hãy xem xét lại cách đặt các thiết bị, sơ đồ thiết kế mạch,...

Khi gặp vấn đề về code:

- Nếu lỗi cấu trúc: hãy google lỗi và sẽ chắc chắn có cách giải quyết.
- Nếu lỗi logic: hãy xem xét lại sơ đồ thiết kế hệ thống (trên giấy hoặc trong não).

Khi gặp một vấn đề mang tính chất hệ thống:

- Hãy tách riêng từng phần và đảm bảo từng phần chạy đúng như ý bạn mong muốn. Sau đó ghép từng phần một với nhau và kiểm tra từng chức năng một. Không nên ghép tất cả một lần rồi kiểm tra một thể, như vậy thì khó mà phát hiện bug ở phần nào.
- Nếu từng module hoạt động rất tốt nhưng khi kết hợp với nhau không chạy được, thì hãy suy nghĩ cách kết hợp của bạn không đúng hoặc bạn nên thay đổi framework khác để có thể chạy.

Khi gặp vấn đề về con người:

- Hãy bình tĩnh và giải quyết dựa vào trình độ thuyết phục, hòa giải của bạn.

7.4 Cách suy nghĩ về hướng giải quyết vấn đề

Hãy thử đặt nhiều câu hỏi khác nhau về vấn đề của bạn.

Hãy thử nói vấn đề của bạn với người khác hoặc rubber duck. Đôi lúc họ không giúp được bạn, mà bạn tự nói vấn đề của mình để bạn xem

lại (review) lại quá trình làm thì đôi lúc bạn sẽ tự tìm thấy bug có khả năng ở đâu.

Luôn giới hạn phạm vi của bug nhỏ nhất có thể để tìm ra chính xác vị trí bug.

- Bug đang ở module nào?
- Bug đang ở file nào?
- Bug đang ở hàm nào?
- Bug đang ở đoạn code nào?
- Bug đang ở line(s) code nào?

LỜI KẾT

Quyển sách đã giới thiệu một số vấn đề mà phần lớn chúng ta đều gặp trong quá trình phát triển phần mềm cho vi điều khiển. Đây chắc chắn không thể đủ về số lượng cũng như chiều sâu của từng vấn đề một. Như ý tưởng ban đầu, quyển sách chỉ đặt cho bạn đọc một số vấn đề, giới thiệu một số từ khóa để bạn đọc có thể dựa trên đó mà tìm hiểu chuyên sâu.

Nếu bạn đọc có bất cứ góp ý/thắc mắc liên quan đến nội dung quyển sách thì đừng ngần ngại gửi mail về nhóm tác giả theo thông tin tác giả bên dưới.

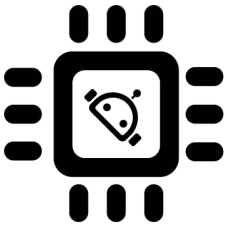
THÔNG TIN TÁC GIẢ VÀ BẢN QUYỀN

Tác giả

Người biên soạn: minht57 lab

Email: minht57.lab@gmail.com

Định hướng lĩnh vực nghiên cứu: lập trình nhúng (embedded software) và robotics.



Bản quyền

minht57 lab