Chương 1: Tổng quan về đề tài và các phần mềm hỗ trợ

1.1. Tổng quan đề tài

Việc lái một động cơ có vẻ như là một nhiệm vụ dễ dàng: mắc động cơ vào một nguồn thay đổi áp được là xong. Nhưng đây không phải là cách hoàn hảo để điều khiển một động cơ , đặc biệt là khi có các thành phần khác đến mạch. Do đó sự cần thiết của mạch cầu H là không thể bàn cãi: đóng ngắt nhanh, hiệu suất cao.

Ứng dụng được sủ dụng trong đề tài này chính là điều khiển động cơ qua mạng không dây thông qua ứng dụng Blynk. Ứng dụng này tuy đơn giản nhưng thể hiện được sự kết hợp của điện tử công suất và IOT

Trong giới hạn đề tài này: chương 2 sẽ trình bày cách thiết kế và thi công mạch in cụng như giới thiệu thiết bị và kết nối phần cứng, trong khi đó chương 3 sẽ trình bày những ý tưởng cốt lõi để lập trình cho hệ thống hoạt động khi đã có phần cứng hoàn chỉnh và chương 4 sẽ trình bày kết quả.

1.2. Các phần mềm hỗ trợ

1.2.1. Phần mềm thiết kế mạch Altium designer

Altium Designer trước kia có tên gọi quen thuộc là Protel DXP, là một trong những công cụ vẽ mạch điện tử mạnh nhất hiện nay. Được phát triển bởi hãng Altium Limited. Altium designer là một phần mềm chuyên nghành được sử dụng trong thiết kế mạch điện tử. Nó là một phần mềm mạnh với nhiều tính năng thú vị, tuy nhiên phần mềm này còn được ít người biết đến so với các phần mềm thiết kế mạch khác như orcad hay proteus. Altium Designer có một số đặc trưng sau:

- Giao diện thiết kế, quản lý và chỉnh sửa thân thiện, dễ dàng biên dịch, quản lý file, quản lý phiên bản cho các tài liệu thiết kế.
- Hỗ trợ mạnh mẽ cho việc thiết kế tự động, đi dây tự động theo thuật toán tối ưu, phân tích lắp ráp linh kiện. Hỗ trợ việc tìm các giải pháp thiết kế

hoặc chỉnh sửa mạch, linh kiện, netlist có sẵn từ trước theo các tham số mới.

- Mở, xem và in các file thiết kế mạch dễ dàng với đầy đủ các thông tin linh kiện, netlist, dữ liệu bản vẽ, kích thước, số lượng…
- Hệ thống các thư viện linh kiện phong phú, chi tiết và hoàn chỉnh bao gồm tất cả các linh kiện nhúng, số, tương tự…
- Đặt và sửa đối tượng trên các lớp cơ khí, định nghĩa các luật thiết kế, tùy chỉnh các lớp mạch in, chuyển từ schematic sang PCB, đặt vị trí linh kiện trên PCB.
- Mô phỏng mạch PCB 3D, đem lại hình ảnh mạch điện trung thực trong không gian 3 chiều, hỗ trợ MCAD-ECAD, liên kết trực tiếp với mô hình STEP, kiểm tra khoảng cách cách điện, cấu hình cho cả 2D và 3D
- Hỗ trợ thiết kế PCB sang FPGA và ngược lại.

Từ đó, chúng ta thấy Altium designer có nhiều điểm mạnh so với các phần mềm khác như đặt luật thiết kế, quản lý đề tài mô phỏng dễ dàng, giao diện thân thiện,…

1.2.2. Phần mềm lập trình cho vi điều khiển Arduino IDE

Arduino IDE là chữ viết tắt của Arduino Integrated Development Environment, một công cụ lập trình với các board mạch Arduino. Nó bao gồm các phần chính là Editor (trình soạn thảo văn bản, dùng để viết code), Debugger (công cụ giúp tìm kiếm và sửa lỗi phát sinh khi build chương trình), Compiler hoặc interpreter (công cụ giúp biên dịch code thành ngôn ngữ mà vi điều khiển có thể hiểu được và thực thi code theo yêu cầu người dùng). Do đối tượng lập trình trong đề này là ESP8266 nên arduino IDE cần tải thêm thư viện mở rộng lập trình cho ESP8266.

1.2.3. Ứng dụng BLYNK trên AppStore/Google Play

Blynk là một phần mềm mã nguồn mở được thiết kế cho các ứng dụng IoT(Internet of Things). Ứng dụng giúp người dùng điều khiển phần cứng từ xa , có thể hiển thị dữ liệu cảm biến , lưu trữ dữ liệu , biến đổi dữ liệu hoặc làm nhiều việc khác. Nền tảng Blynk có ba phần chính:

- Blynk App – Ứng dụng Blynk cho phép khởi tạo giao diện cho các dự án của mình
- Blynk Server – Chịu trách nhiệm giao tiếp qua lại hai chiều giữa điện thoại và phần cứng. Bạn có thể sử dụng server của Blynk nhưng sẽ bị giới hạn điểm Enegry. Trong các hướng dẫn sau này mình sẽ sử dụng Server riêng của mình! Và bạn cũng có thể sử dụng nó
- Blynk Library – Thư viện chứa các nền tảng phổ biến , giúp việc giao tiếp phần cứng với Server dễ dàng hơn

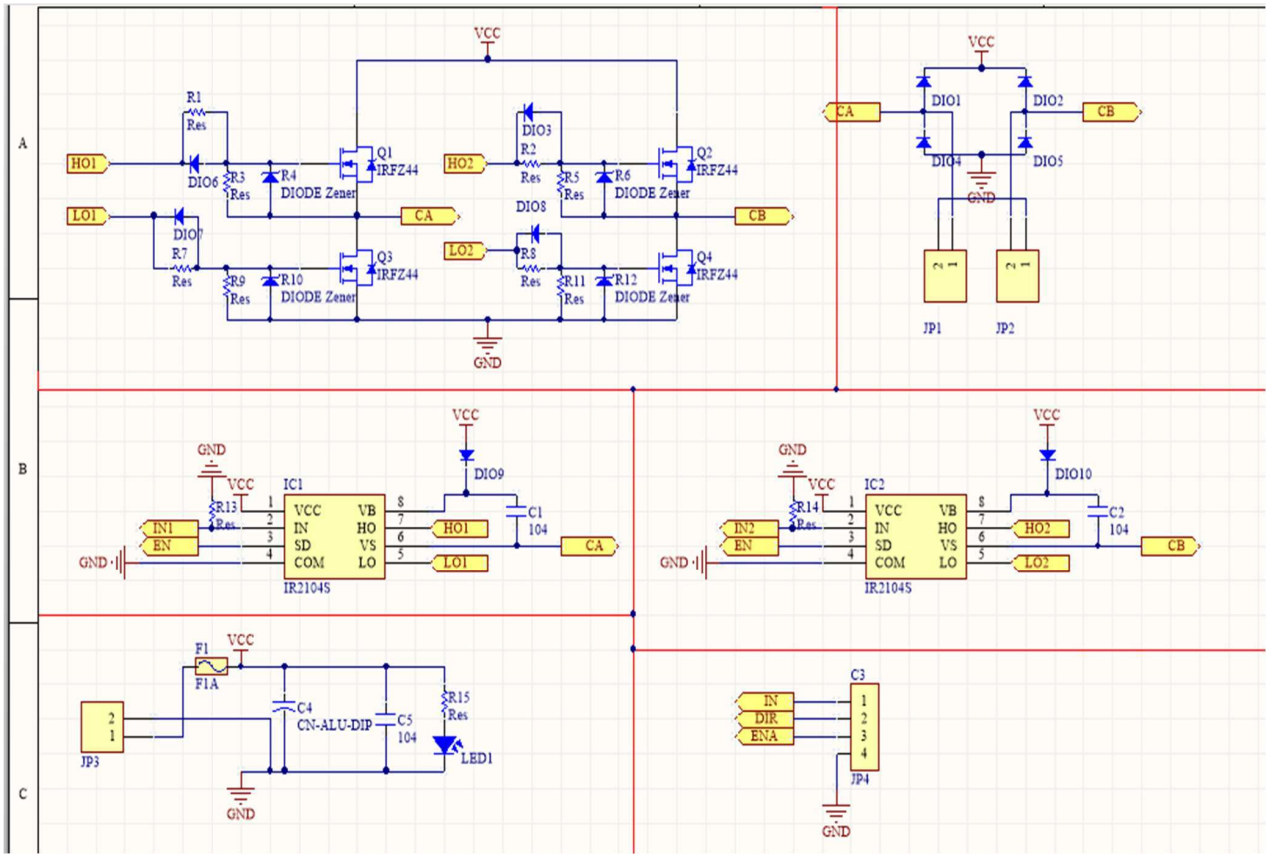
Chương 2: Thiết kế phần cứng

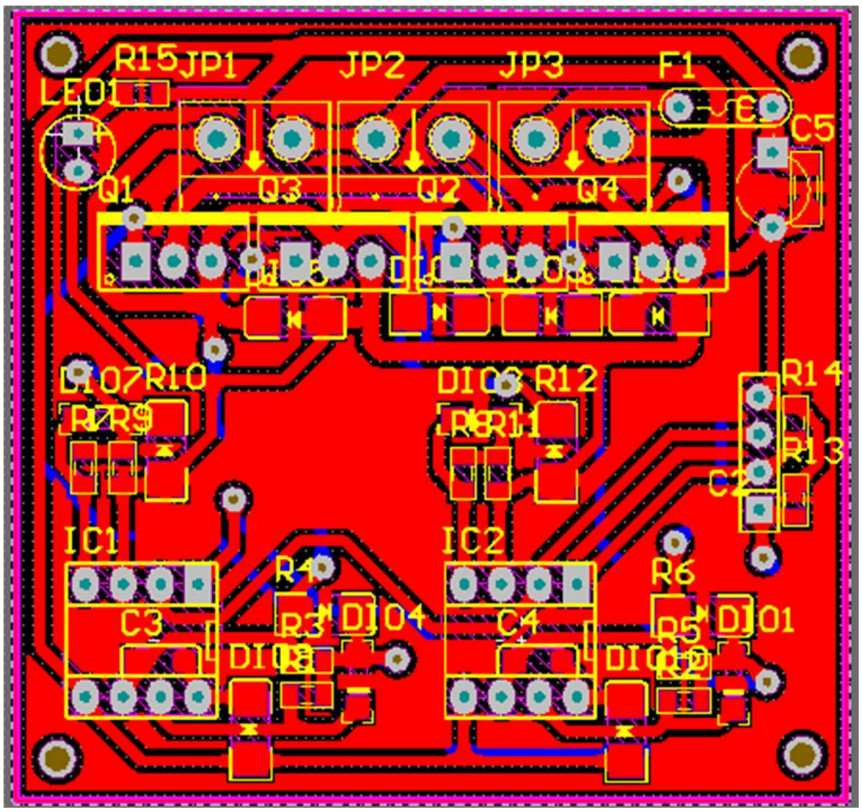
2.1. Mạch cầu H

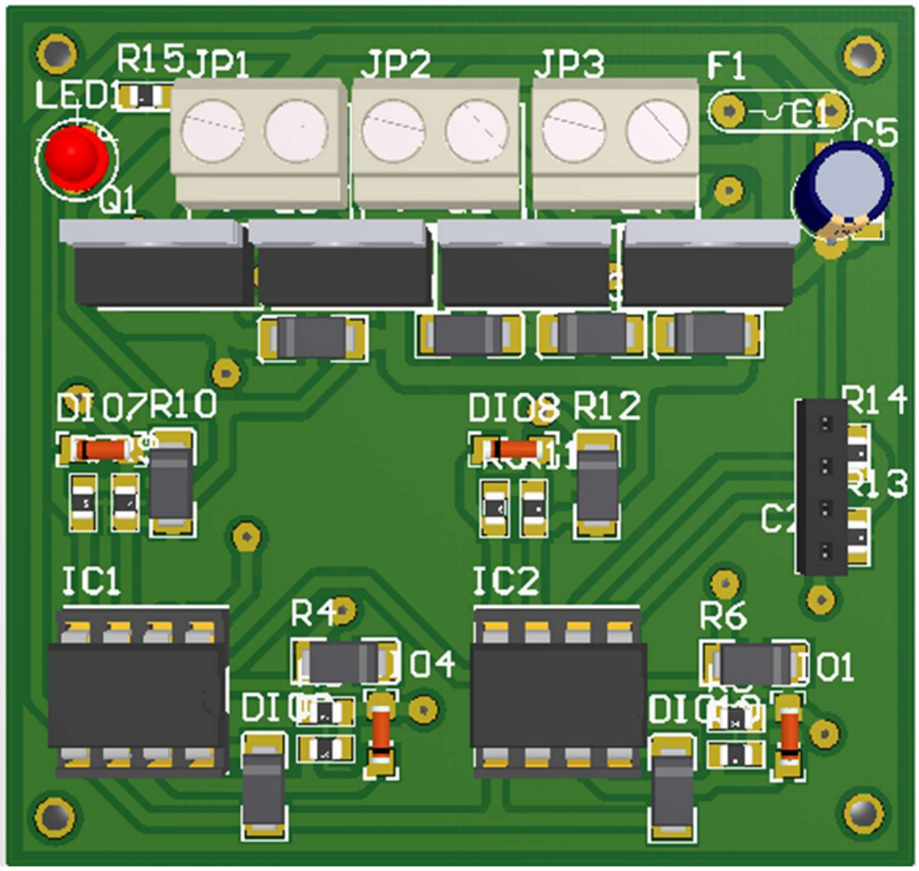
Mạch cầu H sử dụng IC lái là IR2104, các tụ C1 và C2 nối ở mỗi IR2104 là tụ boosttrap

Mosfet được kích từ IR2104 sẽ đóng ngắt để đưa áp ra động cơ, ở mỗi Mosfet đều có một Diode Zener 12V nối giữa 2 cực G và S để bảo vệ điện áp cực V_{GS} không vượt quá 12V. Ngoài ra còn có các diode mắc ngược với điện trở để giúp điện áp xả nhanh hơn giúp việc đóng ngắt diễn ra trong thời gian nhanh hơn.

Các điện trở nối cực chân G và S của Mosfet để giúp giải phóng các điện tích dư ở cực G, để mạch có thể hoạt động tốt trong thời gian dài mà không bị vấn đề gì.

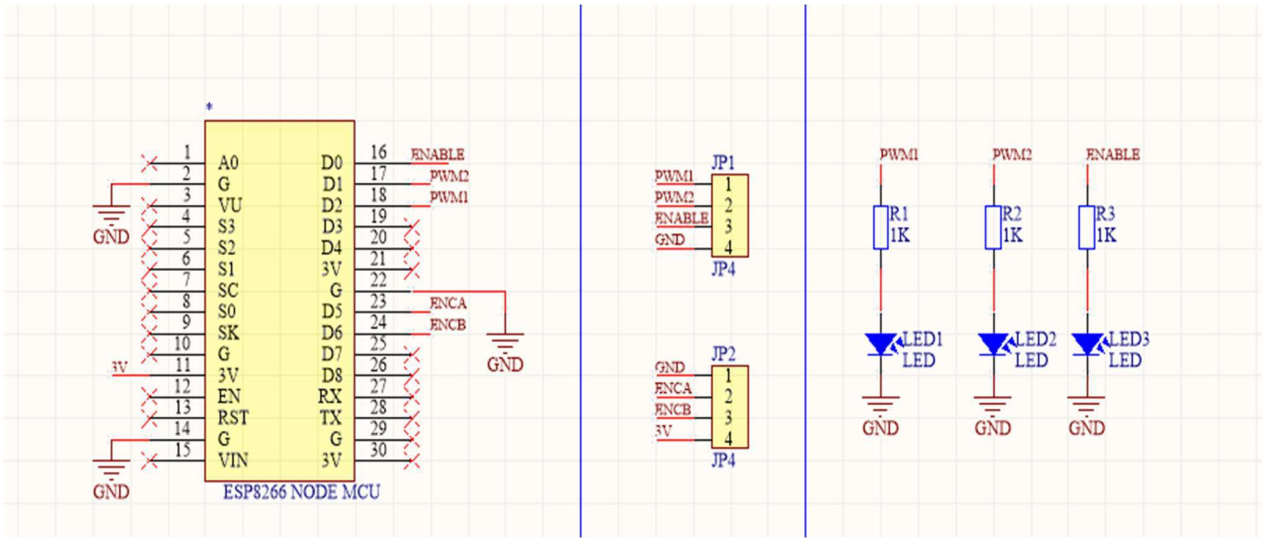

A
B
C
VCC
GND
HO1
HO2
LO1
LO2
CA
CB
R1
Res
DIO6
R3
Res
R4
DIODE Zener
Q1
IRFZ44
DIO3
R2
Res
R5
Res
R6
DIODE Zener
Q2
IRFZ44
DIO8
DIO7
R7
Res
R9
Res
R10
DIODE Zener
Q3
IRFZ44
R8
Res
R11
Res
R12
DIODE Zener
Q4
IRFZ44
DIO1
DIO2
DIO4
DIO5
JP1
JP2
IC1
IC2
IR2104S
VCC
VB
IN
HO
SD
VS
COM
LO
R13
R14
Res
IN1
IN2
EN
DIO9
DIO10
C1
C2
104
F1
F1A
JP3
C4
CN-ALU-DIP
C5
104
R15
Res
LED1
C3
DIR
ENA
JP4

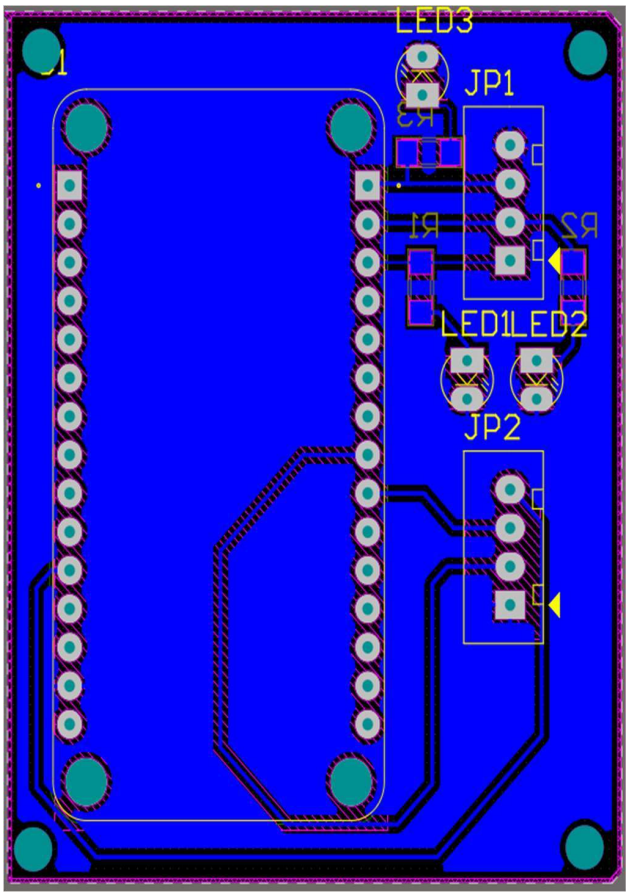

R15
JP1
JP2
JP3
F1
LED1
C5
Q1
Q3
Q2
Q4
R14
R13
C2
IC1
IC2
R4
R6
C3
C4
R5

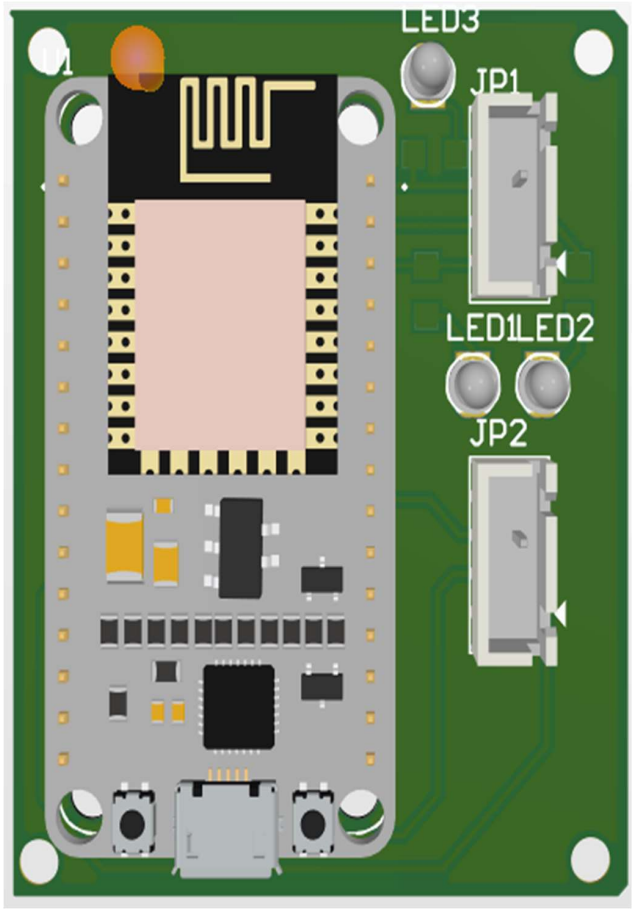

R15
JP1
JP2
JP3
F1
LED1
C5
Q1
R14
R13
C2
IC1
IC2
R4
R6

2.2. Mạch ra chân cho ESP8266

ESP8266 có nhiệm vụ chính là nhận tín hiệu từ điện thoại thông qua wifi, sau đó dựa vào bộ điều khiển PID để xuất xung ra chân được chọn. Ngoài ra cũng có 2 chân encoder để đọc tín hiệu trả về tự động cơ. Mạch còn có thêm 3 đèn led tượng trưng cho Enable, tín hiệu từ chân PWM1 và PWM2







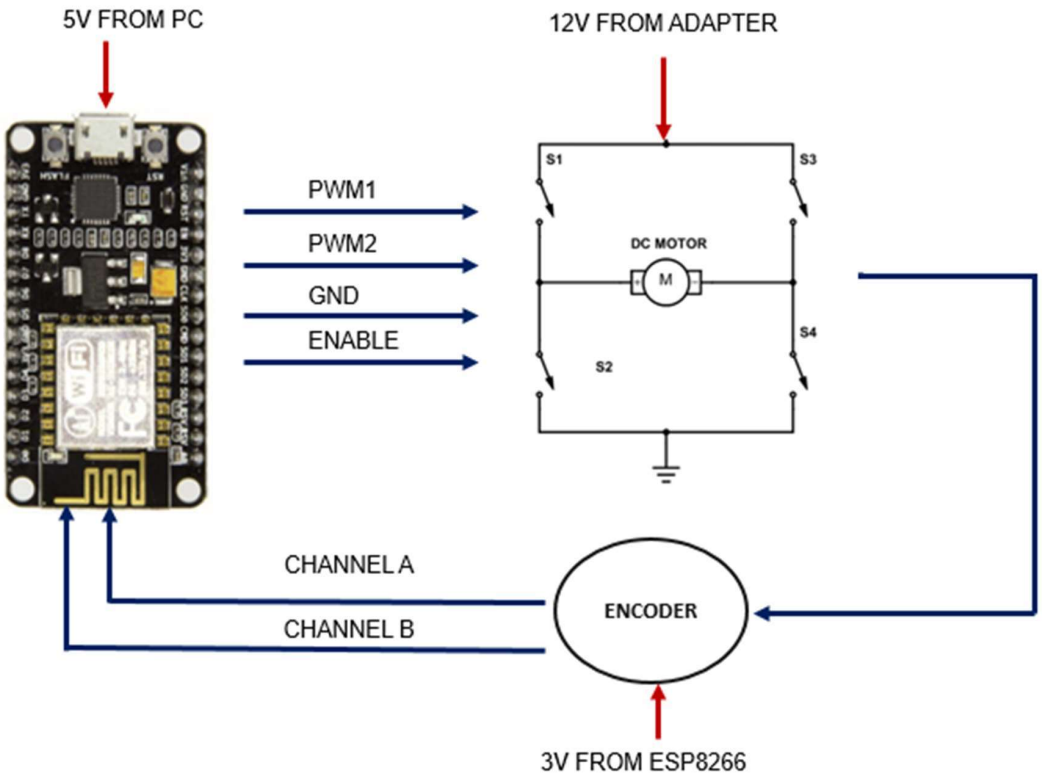
2.3. Kết nối phần cứng

Động cơ được xài trong đề tài là động cơ DC GA25 180 rpm có các tính chất sau :

+ Tỉ số truyền 45:1 (động cơ quay 45 vòng trục chính hộp giảm tốc quay 1 vòng).
+ Dòng không tải: 150mA
+ Dòng chịu đựng tối đa khi có tải: 750mA
+ Tốc độ không tải: 180RPM (180 vòng 1 phút)
+ Tốc độ chịu đựng tối đa khi có tải: 140RPM (140 vòng 1 phút)
+ Lực kéo Moment định mức: 4.3KG.CM
+ Lực léo Moment tối đa: 5.2KG.CM
+ Chiều dài hộp số L: 21mm
+ Số xung Encoder mỗi kênh trên 1 vòng quay trục chính: 11 x 45 = 495 xung.

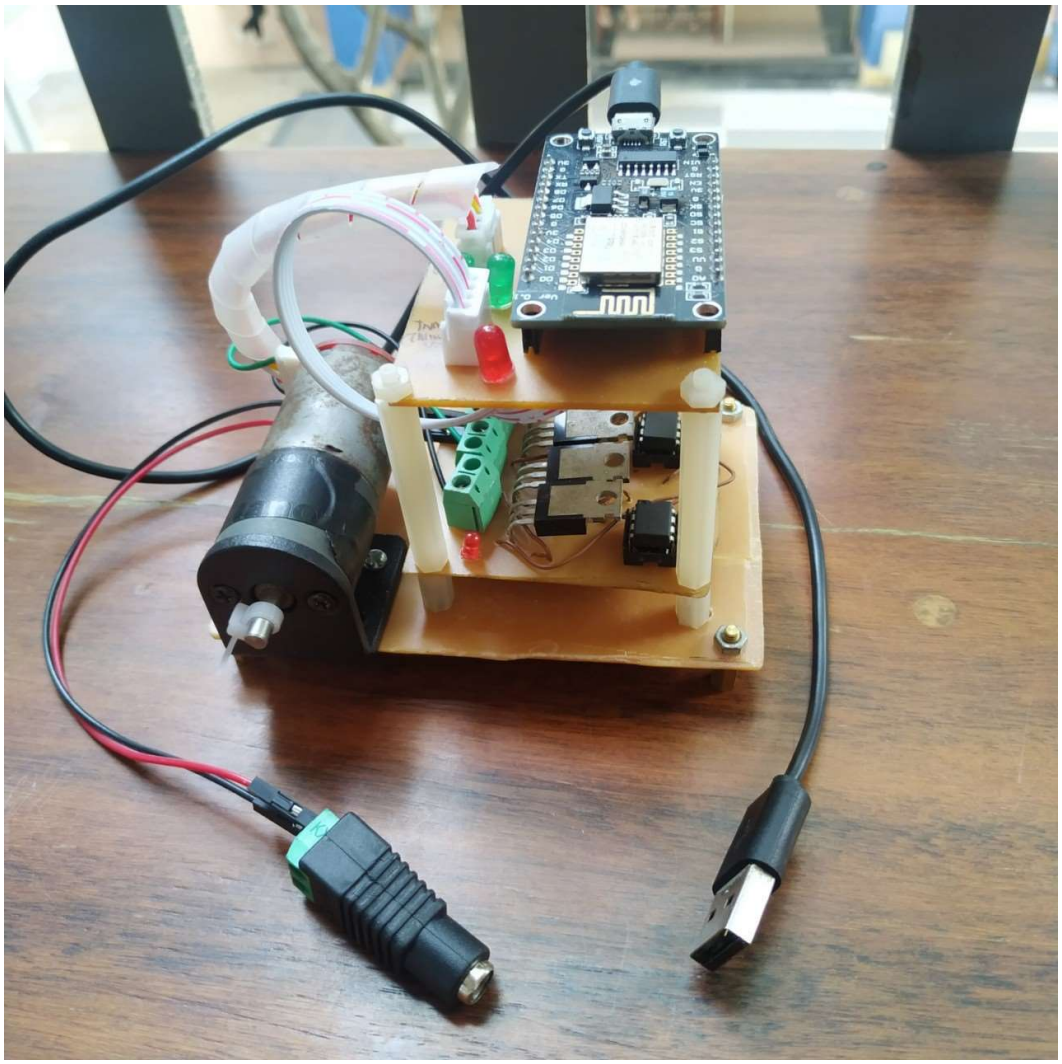
DC GA25 180rpm

Sơ đồ kết nối phần cứng:

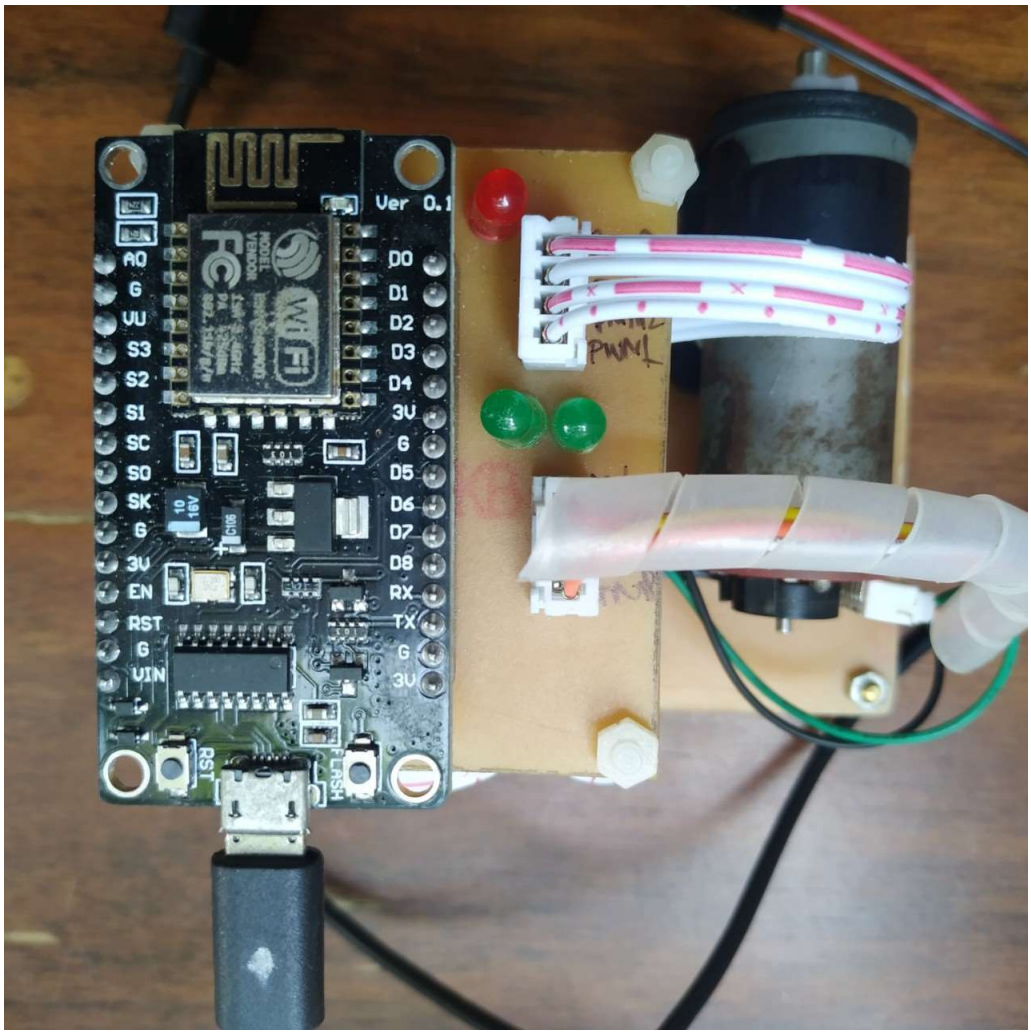


Hình ảnh mạch hoàn chỉnh thực tế

-



-


Ver 0.1
A0
G
VU
S3
S2
S1
SC
SO
SK
G
3V
EN
RST
G
VIN
D0
D1
D2
D3
D4
3V
G
D5
D6
D7
D8
RX
TX
G
3V
RST
FLASH
PWM1

-

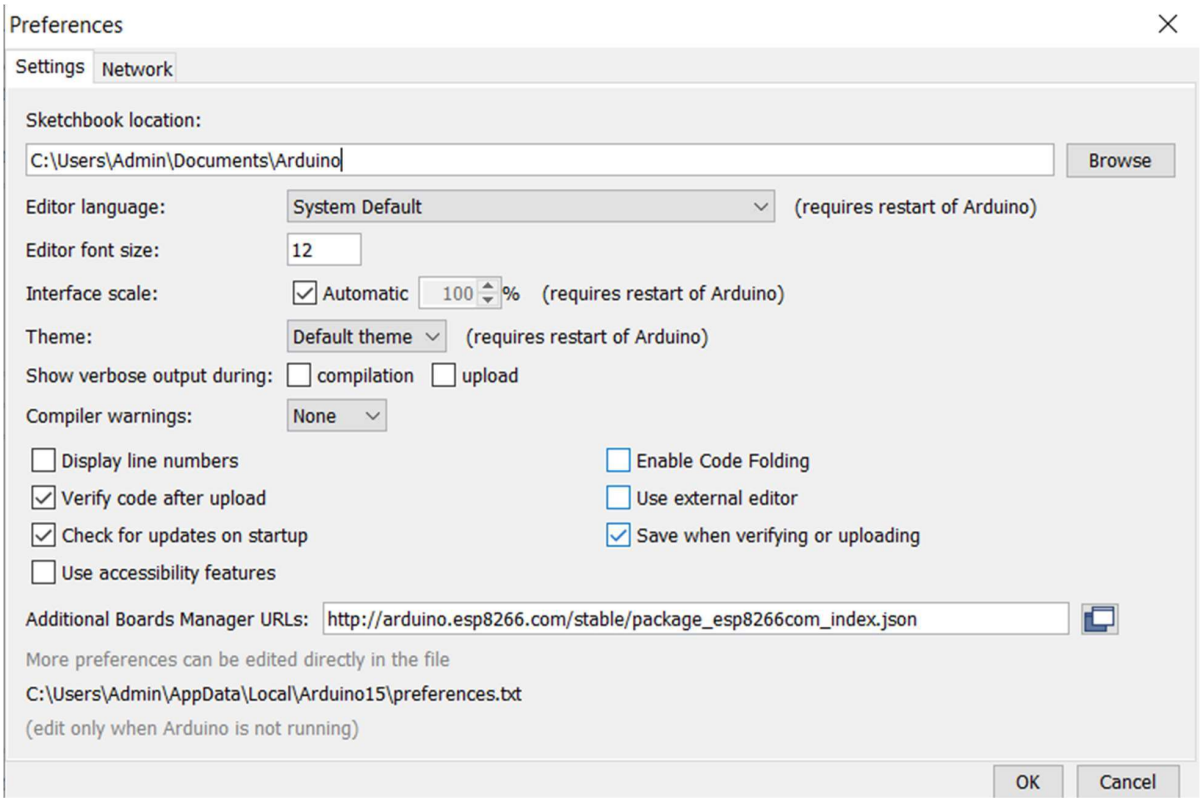
Chương 3: Lập trình cho ESP8266 và App Blynk

3.1. Lập trình cho ESP8266

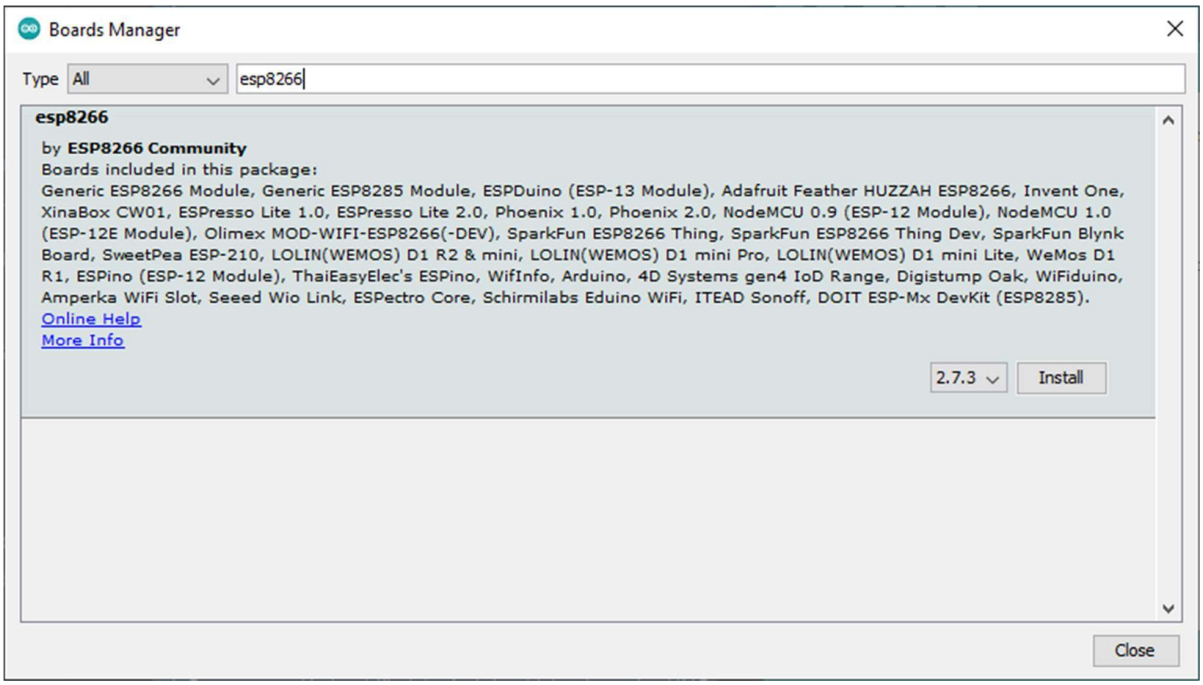
3.1.1. Cài đặt package ESP8266 vào Arduino IDE

Khởi động Arduino IDE, từ màn hình chính chọn File → Preferences, thêm đường dẫn bên dưới vào mục **Addition Boards Manager URLs**.

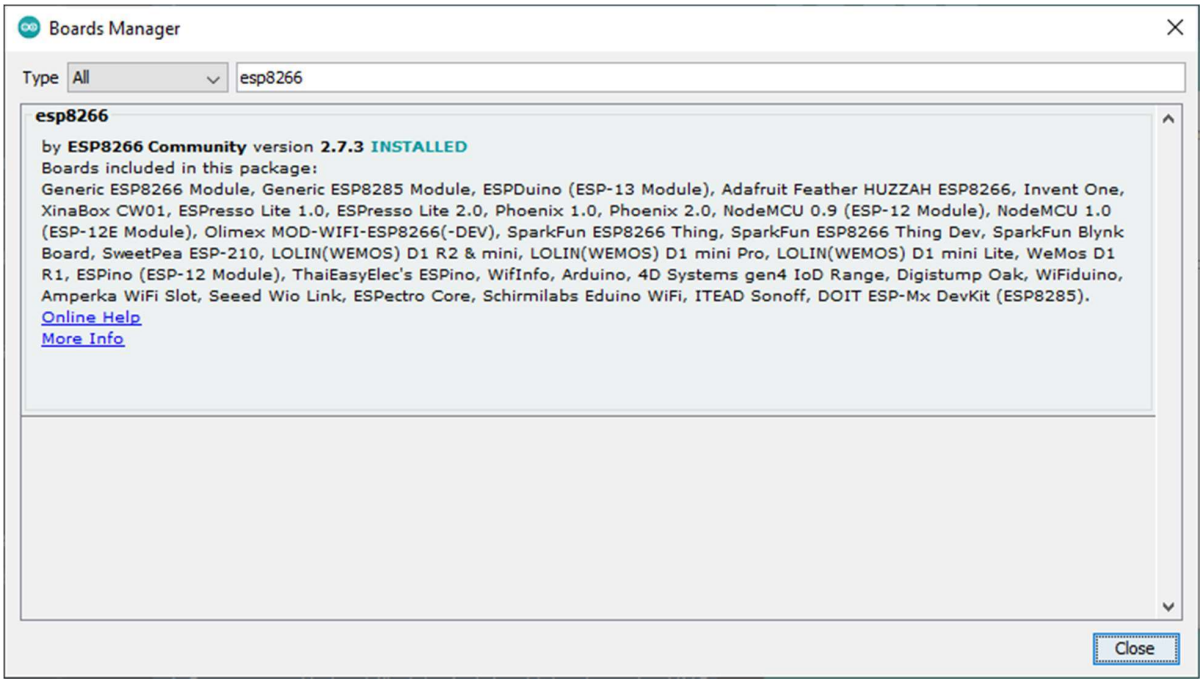
```
http://arduino.esp8266.com/stable/package_esp8266com_index.json
```



Từ giao diện chính của Arduino IDE, chọn Tools → Board → Board Managers, ... Tại thanh tìm kiếm của hộp thoại Board Managers nhập vào esp8266, chọn Install để tiến hành tải và cài đặt thư viện.



Cài đặt thành công, giao diện của Board Managers sẽ trở nên như hình dưới - hoàn tất cài đặt



Như vậy, trong chương trình chính chỉ cần thêm thư viện bằng cách:

```
#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>
```

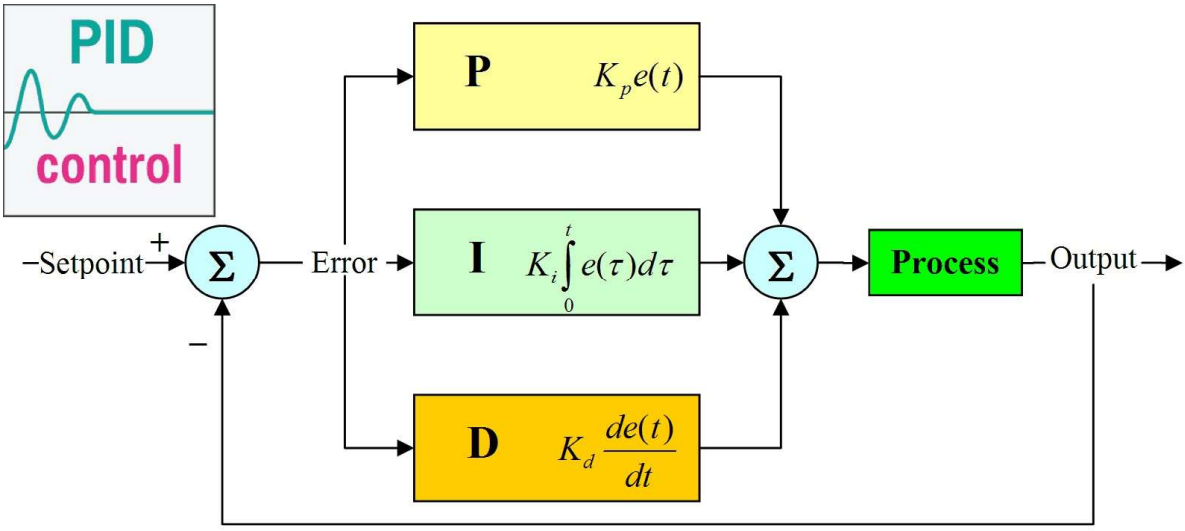
Và khai báo AuthToken lấy từ Blynk App và tên, mật khẩu wifi cần truy cập để vào Blynk server:

```
char auth[] = "70O4dWPgxSZLQmmA-RnrQ8jZMHDWFzVY";
char ssid[] = "Thuyendeptrai"; // Nhập tên WiFi
char pass[] = "sasageyo";   // Nhập password WiFi
```

3.1.2. Lập trình điều khiển động cơ:

Rời rạc hoá thuật toán PID:

Thuật toán sử dụng trong để tài chính là PID, dùng PID để điều khiển vị trí và tốc độ động cơ.



Bộ điều khiển PID bao gồm 3 thông số riêng biệt, do đó đôi khi nó còn được gọi là điều khiển ba khâu: các giá trị tỉ lệ, tích phân và đạo hàm là P, I, và D. Giá trị tỉ lệ xác định tác động của sai số hiện tại, giá trị tích phân xác định tác động của tổng các sai số quá khứ, và giá trị vi phân xác định tác động của tốc độ biến đổi sai số. Tổng chập của ba tác động này dùng để điều chỉnh quá trình thông qua một phần tử điều khiển như vị trí của van điều khiển hay bộ nguồn của phần tử gia nhiệt. Nhờ vậy, những giá trị này có thể làm sáng tỏ về quan hệ thời gian: P phụ thuộc vào sai số hiện tại, I phụ thuộc vào tích lũy các sai số quá khứ, và D dự đoán các sai số tương lai, dựa vào tốc độ thay đổi hiện tại.

Trong Arduino IDE, ta dùng ngôn ngữ C để biểu diễn bộ điều khiển PID như sau:

```
float duty_cycle =  myPID(float KP,float KI,float KD,float
current,int setpoint)
{
  error = setpoint - current;
  error_sum += error;
  d_error = (error - pre_error);
  duty_cycle = KP*error + KI*Ts*error_sum + KD*d_error/Ts ;
  pre_error = error ;
  if (duty_cycle > 100)
    duty_cycle =  99;
  else if(duty_cycle<-100)
    duty_cycle = -99;
  return(duty_cycle);
}
```

Ở đây PID liên tục đã được rời rạc hoá theo chu kì lấy mẫu Ts, Ts càng nhỏ ta càng dễ điều khiển, tuy nhiên, do giới hạn phần cứng của ESP8266 nên chỉ phù hợp ở mức Ts = 0.1 s. Ngõ ra của bộ biểu khiển PID chính là độ rộng xung(tính bằng %): số dương đại điện cho động cơ quay chiều thuận và ngược lại.

Xuất xung PWM ra cầu H

Sau khi có được độ rộng xung, dùng ESP8266 xuất xung PWM ra cầu H đã thiết kế:

```
if (my_duty_cycle >= 0)
{
  analogWrite(PWM1,(int)(my_duty_cycle/100*1023));
  analogWrite(PWM2,0);
}
else
{
  analogWrite(PWM2,-my_duty_cycle/100*1023);
  analogWrite(PWM1,0);
}
```

Xung PWM mặc định của ESP8266 là 1000Hz với độ phân giải là 10 bit tức 1024 mức (0..1023), do đó ta có được hàm mapping là: `my_duty_cycle/100*1023`

Đọc giá trị Encoder:

Động cơ mà nhóm sử dụng là GA25 – 180 rpm – 11PPR – 45 là động cơ gồm hộp số 45:1 và encoder có độ phân giải là 11 xung / vòng, do đó, ta mong muốn số vòng quay của trục giảm tốc nên ta quy đổi sang là 495PPR.

Tiến hành khởi tạo hàm ngắt cho các ENCODER PIN:

```
pinMode(ENCA, INPUT_PULLUP);
pinMode(ENCB, INPUT_PULLUP);
attachInterrupt(ENCA, ISR_encoder, FALLING);
myTic.attach(Ts, handle_interrupt);
```

và trình phục vụ ngắt:

```
ICACHE_RAM_ATTR void ISR_encoder()
{
 if (digitalRead(ENCB)==HIGH)   encoderPos++;
 else                           encoderPos--;
    }

    void handle_interrupt()
 {
   cnt = cnt +1;
   if (cnt ==5)
   cnt =0;
     if((flag_speed == 1)&&(flag_position==0))
     {
     calc_speed();
     }
     else if((flag_position == 1)&&(flag_speed == 0))
     {
     calc_position();
     }
 }
```

Tính toán vận tốc:

```
_speed =(float)(encoderPos/(float)Ts/(float)res*60);
```

Tính toán vị trí:

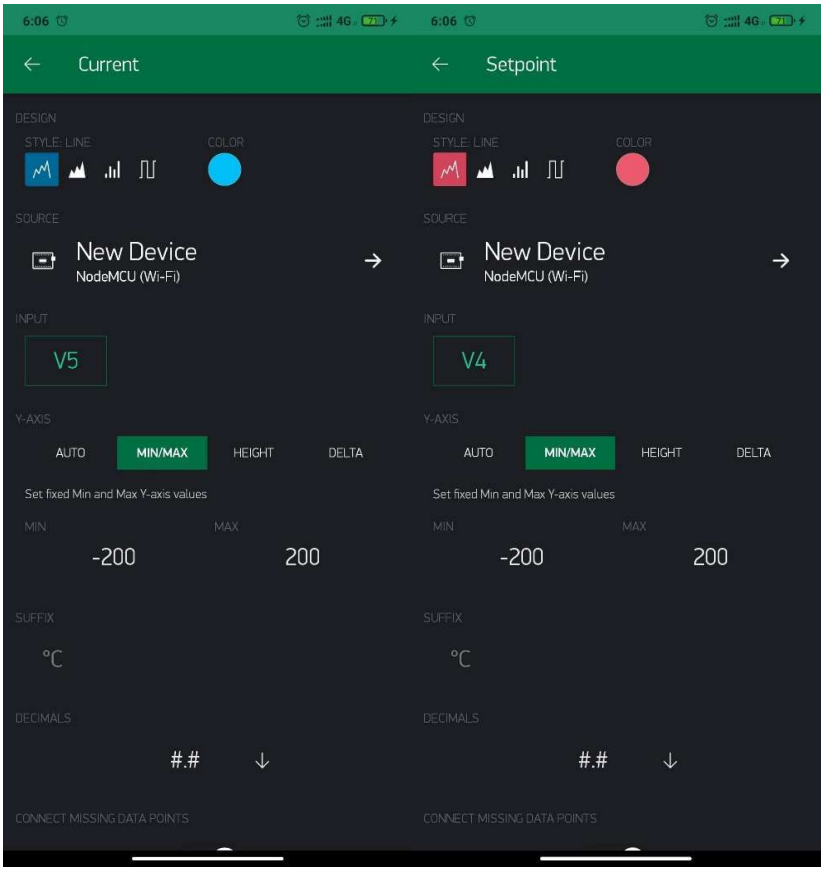
```
_position = (float)(encoderPos)*360/(float)res;
```

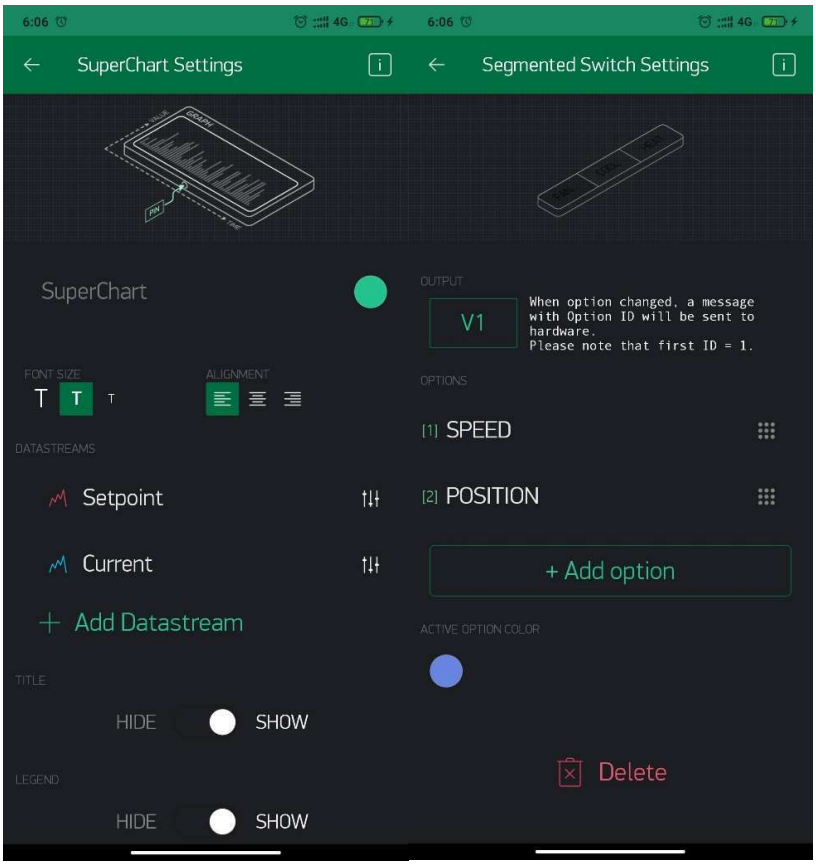
3.2. Lập trình App Blynk

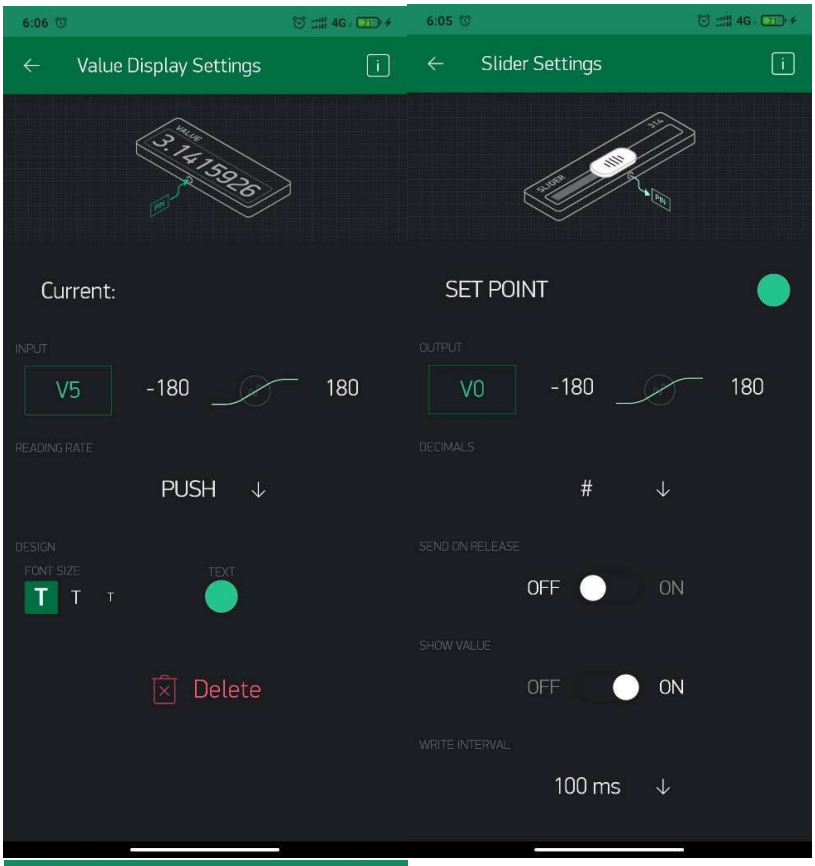
Các đối tượng cần lập trình trên Blynk là:

- 1 Button ON/OFF để cho phép, không cho phép cấp xung
- 1 Slider để trượt thay đổi giá trị đặt
- 1 Supper Chart để vẽ đồ thị đáp ứng
- 1 Value Display để hiện thị giá trị phản hồi.

Cách cấu hình trên từng phần tử:




6:06
SuperChart Settings
SuperChart
FONT SIZE
ALIGNMENT
DATASTREAMS
Setpoint
Current
Add Datastream
TITLE
HIDE
SHOW
LEGEND
HIDE
SHOW
6:06
Segmented Switch Settings
OUTPUT
V1
When option changed, a message with Option ID will be sent to hardware.
Please note that first ID = 1.
OPTIONS
[1] SPEED
[2] POSITION
+ Add option
ACTIVE OPTION COLOR
Delete

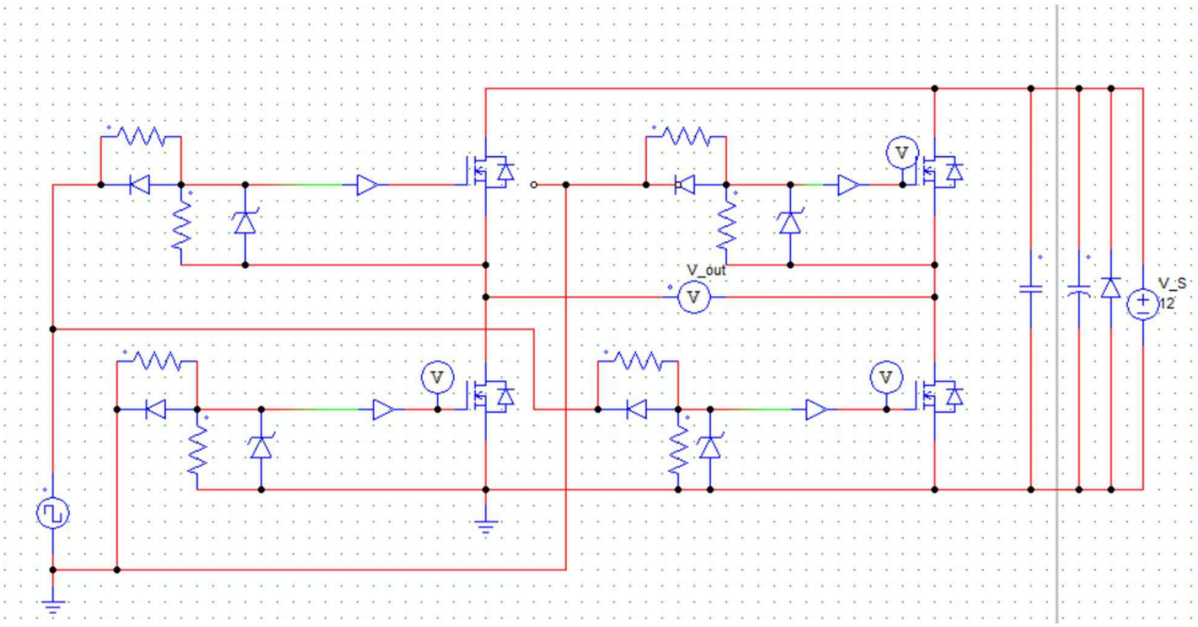

6:06
Value Display Settings
3.14159926
Current:
INPUT
V5
-180
180
READING RATE
PUSH
DESIGN
FONT SIZE
TEXT
Delete
6:05
Slider Settings
SET POINT
OUTPUT
V0
-180
180
DECIMALS
#
SEND ON RELEASE
OFF
ON
SHOW VALUE
OFF
ON
WRITE INTERVAL
100 ms


Styled Button Settings
Title (optional)
OUTPUT
D0
MODE
SWITCH
PUSH
ON
OFF
FONT SIZE
ROUNDED
SHARP

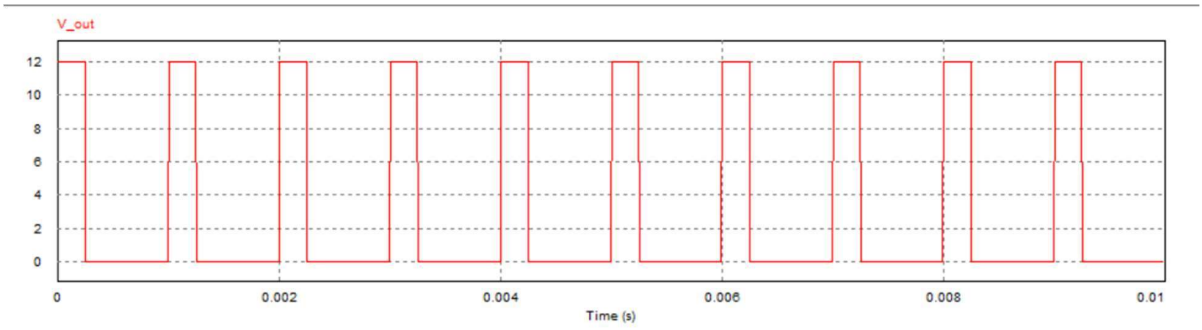
Chương 4: Kết quả

4.1. Mô phỏng

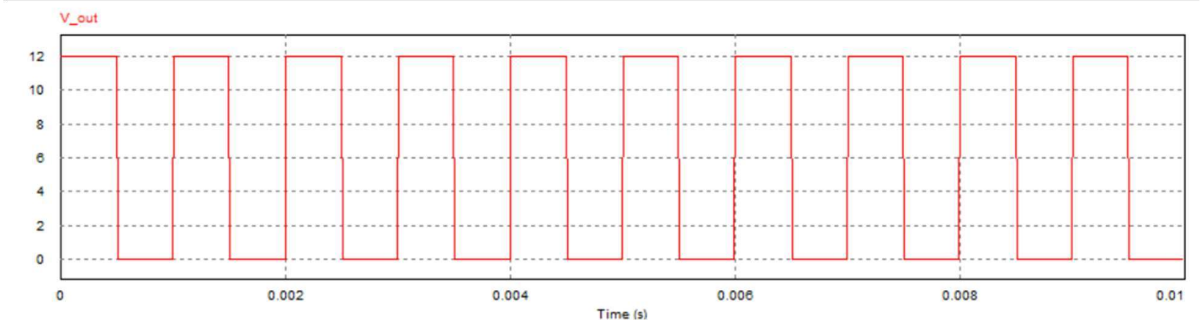
Tiến hành mô phỏng hoạt động của cầu H đã thiết kế trên PSIM



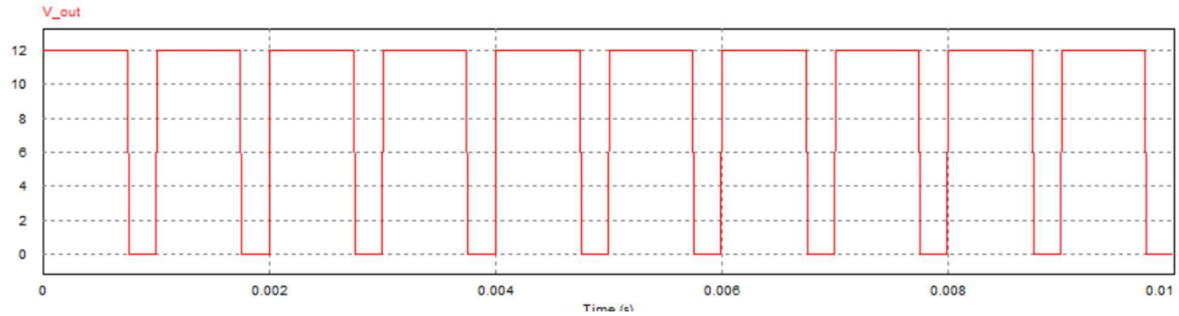
Khi cấp xung điều khiển có tần số 1000Hz, độ rộng xung là 25%:



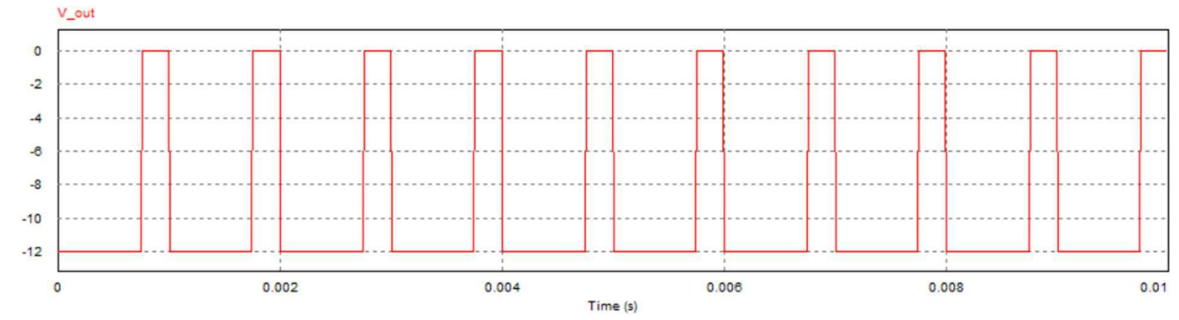
Khi cấp xung điều khiển có tần số 1000Hz, độ rộng xung là 50%:

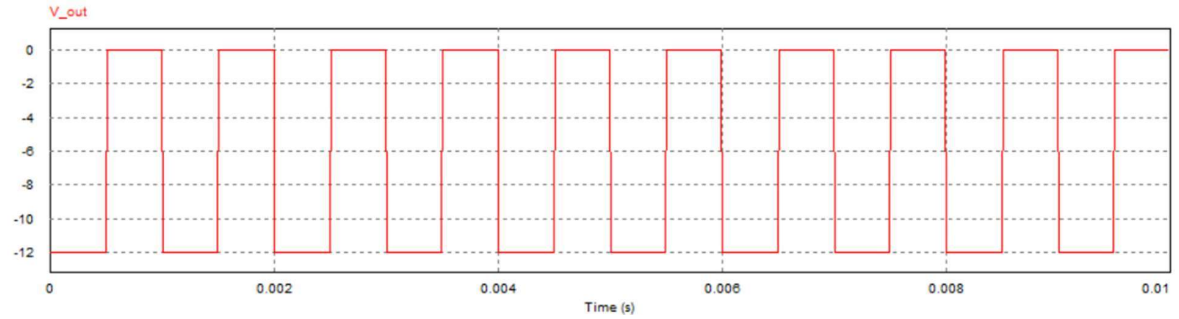


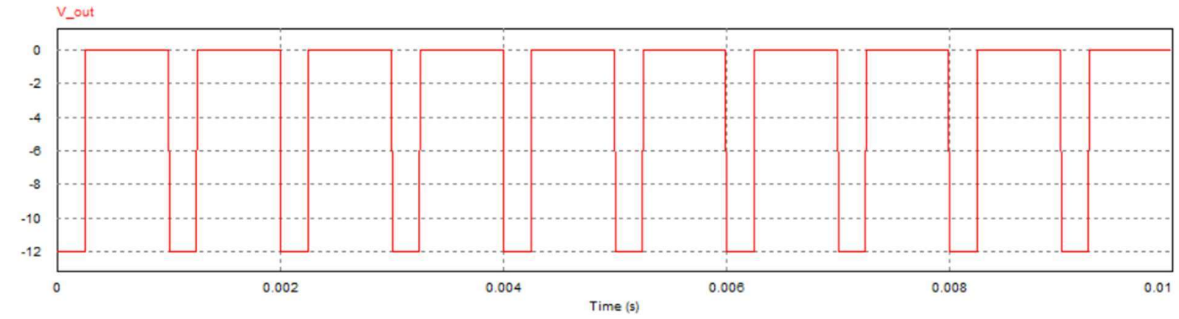
Khi cấp xung điều khiển có tần số 1000Hz, độ rộng xung là 75%:



Thực hiện với chiều ngược lại với độ rộng xung tương ứng 25% 50% 75%





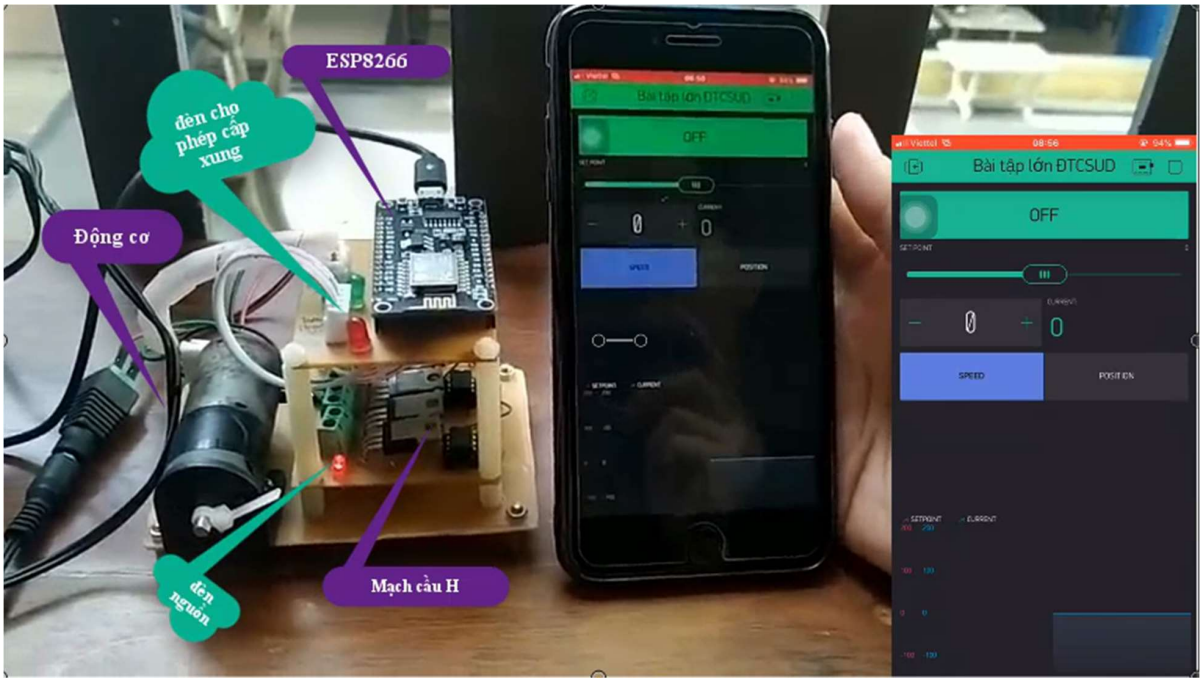


Nhận xét:

- Kết quả mô phỏng cho thấy cầu H đã thiết kế hoạt động đúng với độ rộng xung cấp vào
- Cầu H đó cũng tạo được ngõ ra cả vùng dương và âm của điện áp

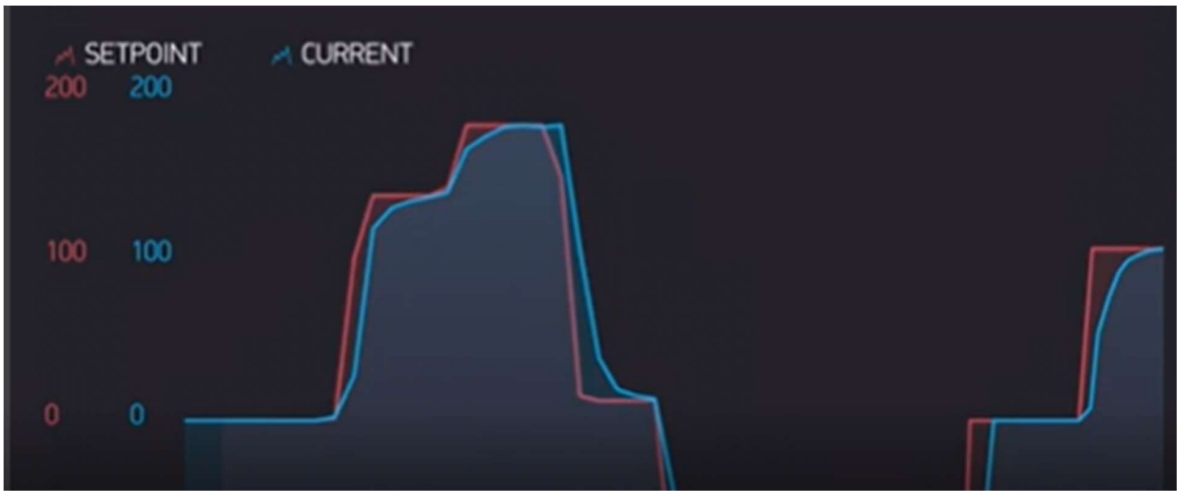
4.2. Thực nghiệm

Toàn bộ hệ thống khi chạy:



Khảo sát đáp ứng:

- Điều khiển vận tốc:



Đáp ứng vận tốc đảm bảo chất lượng điều khiển, thời gian xác lập nhanh, sai số điều khiển bằng 0 và không bị vọt lố. (Do SERVER BLYNK không thể ghi và hiện dữ liệu với tần số quá nhanh, dù tần số lấy mẫu của hệ thống là 100ms nhưng 500ms mới gửi giá trị lên server để vẽ nên đồ thị sẽ không hoàn toàn là đường cong).

Đáp ứng cụ thể có thể xem video đính kèm.

- Điều khiển vị trí

Đáp ứng vị trí là nhanh, sai số bằng 0 nhưng có vọt lố. Điều này có thể lý giải do động cơ gắn hộp số, làm tính phi tuyến của đối tượng tăng lên nên 1 thông số PID không thể đáp ứng được ở mọi vùng hoạt động.

Chương 5: Kết Luận

Kết quả trên phần nào đã đạt được yêu cầu đề ra trong bài tập lớn môn điện tử công suất ứng dụng.

Các công việc đã làm được:

- Thiết kế và thi công mạch cầu H điều khiển động cơ một cách chuẩn theo tiêu chuẩn thiết kế mạch điện tử công suất nói riêng và mạch điện tử nói chung.
- Thiết kế và thi công mạch ra chân cho ESP 8266 cũng như kết nối phần cứng một cách hoàn chỉnh, tạo thành một khối hoàn chỉnh
- Viết chương trình cho ESP8266 và xây dựng chương trình BLYNK trên smartphone để điều khiển vị trí và tốc độ động cơ DC bằng thuật toán PID. Người dùng hoàn toàn có quyền chọn điều khiển vị trí hoặc tốc độ động cơ bằng phần mềm mà không cần can thiệp phần cứng
- Đáp ứng ngõ ra của thuật toán PID cho vận tốc và vị trí động cơ là đảm bảo sai số bằng 0, thời gian xác lập nhanh. Điều đó chứng tỏ mạch cầu H đã thiết kế đáp ứng được nhiệm vụ đề ra.

Các công việc chưa làm được và hướng phát triển:

- Như đã trình bày ở chương 4 thì đáp ứng của PID vị trí chưa hoàn toàn tốt do tính phi tuyến của đối tượng động cơ DC có gắn hộp số lớn (45:1) nên bộ điều khiển PID tuyến tính không thể đáp ứng được toàn bộ vùng làm việc nên xảy ra hiện tượng vọt lố.
- Nhận dạng mô hình động cơ và dùng các bộ điều khiển phi tuyến như hồi tiếp tuyến hoá, bộ điều khiển trượt hoặc sử dụng các bộ điều khiển thích nghi hoặc thông minh như Logic mờ (Fuzzy) để cải thiện chất lượng.

PHỤ LỤC

A. Chương trình ESP8266

```
#define BLYNK_PRINT Serial
#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>
#include <Ticker.h>
char auth[] = "70O4dWPgxSZLQmmA-RnrQ8jZMHDWFzVY";
char ssid[] = "Thuyendeptrai"; // Nhập tên WiFi
char pass[] = "sasageyo";   // Nhập password WiFi
//************************************************************
//************************WIRING*****************************
//************************************************************
// PWM2        <->    D1
// PWM1        <->    D2
// ENCB(C2)  <->    D6
// ENCA(C1)  <->    D5
// ENBALE    <->    D0
//************************************************************
//******************** DEFINE*******************************
//************************************************************
#define Ts 0.1
#define res 495
#define PWM1 D2
#define PWM2 D1
#define ENCA D5
#define ENCB D6
#define ENABLE_PIN D0


//************************************************************
//******************DECLARATION*****************************
//************************************************************
float SPEED_KP = 0.025;
float SPEED_KI = 0.5;
float SPEED_KD = 0;

float POSITION_KP = 0.35;
float POSITION_KI = 0;
float POSITION_KD = 0.0001;
```

```
int encoderPos = 0;
int _speed, _position;
int enable;
int cur_mode, pre_mode;
float error, pre_error=0, error_sum, d_error;
float control_signal;
int flag_speed =1, flag_position=0;
int setpoint;
int reset_position_signal ,reset_speed_signal;
int cnt = 0;
//*********************************************************
//*******************CODING*********************************
//*********************************************************
Ticker myTic;   // Ticker to calc speed


BLYNK_WRITE(V0) // get speed_d from blynk
{
 if (digitalRead(ENABLE_PIN) == HIGH)
 {
  setpoint = param.asInt();
 }
}

BLYNK_WRITE(V1) // get speed_d from blynk
{
 cur_mode = param.asInt();
 // mode = 1: Speed control
 // mode = 2: Position control
 if ((cur_mode == 1)&&(pre_mode == 2))
 {
      reset_position();
    Serial.println("Change to Speed control ");
 }
 if ((cur_mode == 2)&&(pre_mode == 1))
 {
  reset_speed();

    Serial.println("Change to Position control ");
 }
 pre_mode = cur_mode;
```

```
}

void handle_interrupt()
  {
    cnt = cnt +1;
    if (cnt ==5)
    cnt =0;


    //Serial.print(" Encoder Pulse: ");
   // Serial.println(encoderPos)          ;
    //Serial.print("RESET SIGNAL SPEED - POSITION: ");
   // Serial.print(reset_speed_signal) ;
   // Serial.print(reset_position_signal) ;
   // Serial.print(" FLAG SPEED - POSITION: ");
   // Serial.print(flag_speed);
  //  Serial.print(flag_position);

      if((flag_speed == 1)&&(flag_position==0))
      {
      calc_speed();
      }
      else if((flag_position == 1)&&(flag_speed == 0))
      {
      calc_position();
      }
  }

void calc_speed()
{
if (digitalRead(ENABLE_PIN) == HIGH)
{
//***** calc PID
float my_duty_cycle =
myPID(SPEED_KP,SPEED_KI,SPEED_KD,_speed,setpoint);
control_signal = my_duty_cycle*12/100;
if (my_duty_cycle >= 0)
{
  analogWrite(PWM1,(int)(my_duty_cycle/100*1023));
  analogWrite(PWM2,0);
}
```

```
else
{
  analogWrite(PWM2,-my_duty_cycle/100*1023);
  analogWrite(PWM1,0);
}
//******
}
else{
  error_sum = 0;
  setpoint = 0;
  analogWrite(PWM1,0);
  analogWrite(PWM2,0);
  if (cnt == 4)
  {
  Blynk.virtualWrite(V0,0);
  Blynk.virtualWrite(V4,0);
  }
}


Serial.print(" Speed_d: ");
Serial.print(setpoint);
Serial.print(" (rpm) ");
Serial.print(" Speed: ");
_speed = (float)(encoderPos/(float)Ts/(float)res*60);
encoderPos = 0;// reset counter
Serial.print(_speed);
Serial.println(" (rpm) ");
if (cnt == 4)
{
Blynk.virtualWrite(V5,_speed);
Blynk.virtualWrite(V4,setpoint);
}


if ((reset_speed_signal == 1)&&(abs(_speed) <= 1))
  {
    flag_speed   =   0 ;
    flag_position =  1 ;
    reset_speed_signal = 0;
  }
```

```
}

void calc_position()
{
 if (digitalRead(ENABLE_PIN) == HIGH)
{
//***** calc PID
float my_duty_cycle =
myPID(POSITION_KP,POSITION_KI,POSITION_KD,_position,setpoint);
control_signal = my_duty_cycle*12/100;
if (my_duty_cycle >= 0)
{
 analogWrite(PWM1,(int)(my_duty_cycle/100*1023));
 analogWrite(PWM2,0);
}
else
{
 analogWrite(PWM2,-my_duty_cycle/100*1023);
 analogWrite(PWM1,0);
}
//******
}
else{
 error_sum =0;
 setpoint = 0;
 analogWrite(PWM1,0);
 analogWrite(PWM2,0);
 if (cnt == 4)
 {
 Blynk.virtualWrite(V0,0);
 Blynk.virtualWrite(V4,0);
 }
}


Serial.print(" Set point: ");
Serial.print(setpoint);
Serial.print(" (degree) ");
Serial.print(" Position: ");
_position = (float)(encoderPos)*360/(float)res;
```

```
Serial.print(_position);
Serial.println(" (degree) ");
if (cnt == 4)
{
Blynk.virtualWrite(V5,_position);
Blynk.virtualWrite(V4,setpoint);
}

if ((reset_position_signal == 1)&&(abs(_position) <= 5))
  {
    flag_speed   =  1 ;
    flag_position = 0 ;
    reset_position_signal=0;
  }

}
void reset_speed()
{
  setpoint  = 0;
  //Blynk.virtualWrite(V0,0);
 // Blynk.virtualWrite(V4,0);
  reset_speed_signal = 1;
}

void reset_position()
{
  setpoint = 0;
  if (cnt == 4)
  {
  Blynk.virtualWrite(V0,0);
  Blynk.virtualWrite(V4,0);
  }
  reset_position_signal = 1;
}

float myPID(float KP,float KI,float KD,float current,int setpoint)
{
  error = setpoint - current;
  error_sum += error;
  d_error = (error - pre_error);
  float duty_cycle;
```

```
  duty_cycle = KP*error + KI*Ts*error_sum + KD*d_error/Ts ;
  pre_error = error ;
  if (duty_cycle > 100)
    duty_cycle =  99;
  else if(duty_cycle<-100)
    duty_cycle = -99;
  return(duty_cycle);
}


void setup()
{
  Serial.begin(9600);
  Blynk.begin(auth, ssid, pass);
  pinMode(ENCA, INPUT_PULLUP);                    // quadrature encoder
input A
  pinMode(ENCB, INPUT_PULLUP);                    // quadrature encoder
input B
  pinMode(ENABLE_PIN, OUTPUT);
  attachInterrupt(14, ISR_encoder, FALLING);      // update encoder position
  myTic.attach(Ts, handle_interrupt);
}
void loop()
{
  Blynk.run();
}

ICACHE_RAM_ATTR void ISR_encoder()  {
  if (digitalRead(ENCB)==HIGH)   encoderPos++;
  else                    encoderPos--;
}
```