**TRƯỜNG ĐẠI HỌC BÁCH KHOA – ĐẠI HỌC ĐÀ NẴNG  
KHOA CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN**

**LẬP TRÌNH HỆ THỐNG**

**VI ĐIỀU KHIỂN**

***Xe hai bánh tự cân bằng***



**Giáo viên hướng dẫn:** Trần Thế Vũ

**Sinh viên thực hiện:** Trương Văn Quốc Hoàng 13T2

Văn Đức Dũng 13T2

Hồ Thị Yến 13T2

**Đà Nẵng, 12/2016**

**MỤC LỤC**

**I. Giới thiệu bài toán 3**

**II. Phân tích yêu cầu** 5

1. Linh kiện sử dụng 5

2. Giới thiệu linh kiện 5

**III. Thiết kế 12**

1. Sơ đồ tổng quát 12

2. Cách ghép nối 12

**IV. Triển khai và kết quả 12**

1. Lập trình cho từng linh kiện 12

a. Lập trình điều khiển động cơ qua L298 12

b. Lập trình ứng dụng Android 15

2. Source code cho MPU6050 18

3. Source code cho arduino 18

**Kết luận 27**

**Tài liệu tham khảo 28**

**I. Giới thiệu bài toán:**

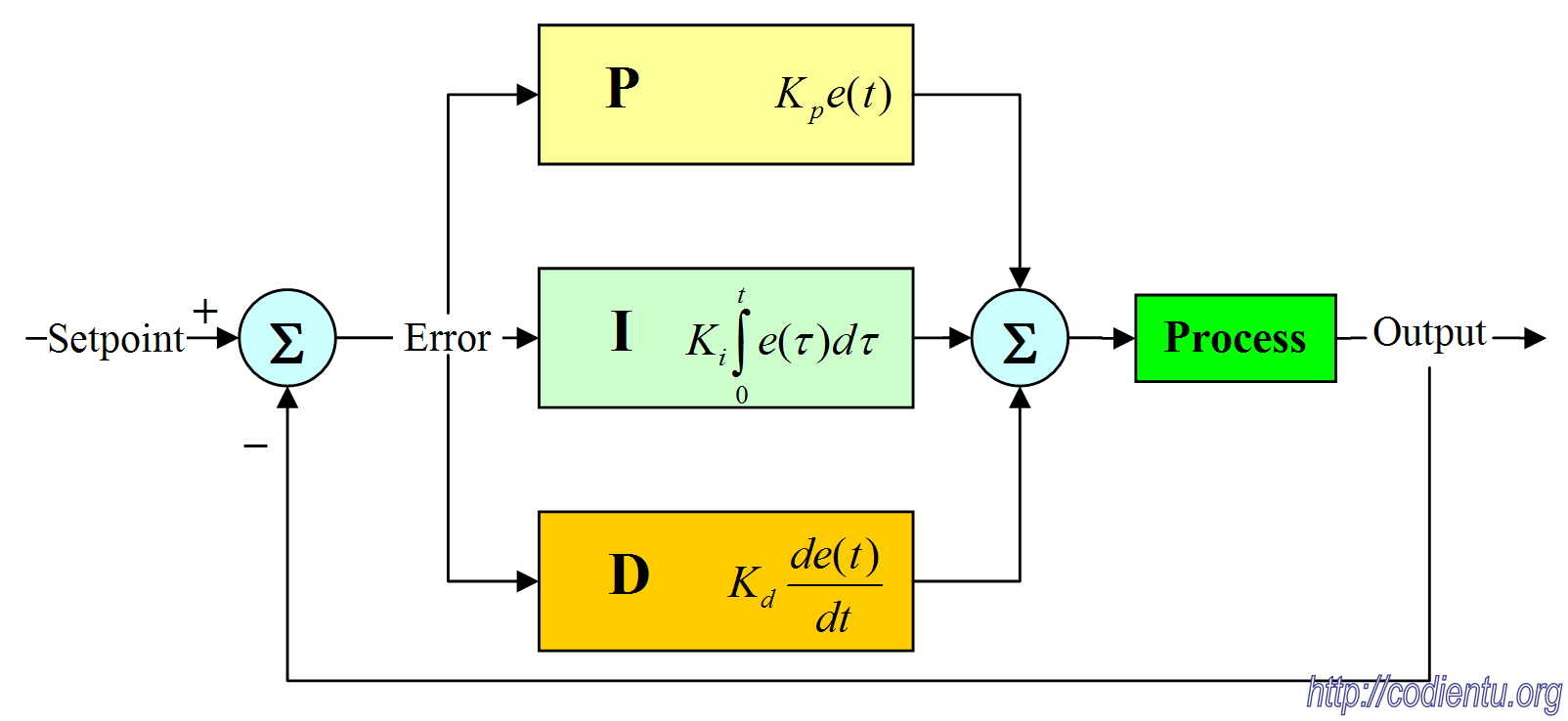
**Bài toán:** Xe hai bánh tự cân bằng điều khiển thông qua bluetooth

Ngày nay khoa học công nghệ ngày càng phát triển, vi điều khiển AVR và vi điều khiển PIC ngày càng thông dụng và hoàn thiện hơn, nhưng có thể nói sự xuất hiện của Arduino vào năm 2005 tại Italy đã mở ra một hướng đi mới cho vi điều khiển. Sự xuất hiện của Arduino đã hỗ trợ cho con người rất nhiều trong lập trình và thiết kế, nhất là đối với những người bắt đầu tìm tòi về vi điều khiển mà không có quá nhiều kiến thức, hiểu biết sâu sắc về vật lý và điện tử. Phần cứng của thiết bị đã được tích hợp nhiều chức năng cơ bản và là mã nguồn mở. Ngôn ngữ lập trình vô cùng dễ sử dụng và hệ thư viện rất phong phú lại được chia sẻ miễn phí. Chính vì những lý do như vậy nên Arduino hiện đang dần phổ biến và được phát triển ngày càng mạnh mẽ trên toàn thế giới.

Trở lại vấn đề giới thiệu bài toán, hiện nay Việt Nam là một nước có nhiều tài nguyên và địa hình hiểm trở, chúng ta có nhu cầu khá nhiều trong việc khám phá và tìm hiểu ở các địa hình. Nhu cầu về một thiết bị robot, xe nhỏ gọn và thông minh, nhanh nhạy rất cần thiết. Điều này thực sự gây ra một số khó khăn cho cho việc tìm kiếm và khám phá mở rộng. Chính vì lý do này, chúng em đã nghiên cứu thực hiện đề tài **“Xe hai bánh tự cân bằng”**, với tiêu chí có thể tạo ra được một chiếc xe nhỏ gọn, nhưng không kém phần mạnh mẽ, linh hoạt có thể phục vụ hiệu quả cho nghiên cứu. Do kiến thức còn hạn hẹp nên chắc chắn không tránh khỏi những thiếu sót, hạn chế, vì thế chúng em rất mong nhận được sự nhắc nhở và góp ý của thầy giáo để hoàn thiện sản phẩm của mình.

**Giải Pháp**: Giải thuật PID

[PID](http://codientu.org/tags/pid/) là cách viết tắc của các từ Propotional (tỉ lệ), Integral (tích phân), Derivative (đạo hàm) và là giải thuật [điều khiển](http://codientu.org/tags/%C4%91i%E1%BB%81u+khi%E1%BB%83n/) được dùng nhiều nhất trong các ứng dụng điều khiển [tự động](http://codientu.org/tags/t%E1%BB%B1+%C4%91%E1%BB%99ng/) với yêu cầu chính xác (accurate), nhanh (fast response), ổn định (small overshot).

 ​

Một điều rất tự nhiên, với yêu cầu nhanh thì một cách đơn giản để công thức hóa ý tưởng này là dùng quan hệ tuyến tính:

F=Kp\*e ​

Trong đó Kp là một hằng số dương nào đó mà chúng ta gọi là hệ số P (Propotional gain), e là sai số cần điều khiển. Mục tiêu điều khiển là đưa e (sai số) tiến về 0 càng nhanh càng tốt. Rõ ràng nếu Kp lớn thì tác động nhanh của điều khiển cũng càng lớn. Tuy nhiên, do quán tính mà việc điều khiển càng nhanh càng gây ra tính mất ổn định (do lực quán tính và lực điều khiển tạo ra cặp đối lực xuất hiện ở hai khoảng thời gian liên tiếp nhau => chúng tạo ra dao động không kiểm soát được).   
Như vậy, ta sẽ sử dụng đạo hàm của sai số e để làm tăng giá trị nhưng ngược chiều của lực F (vì e đang giảm nhanh dần). Nếu sử dụng đạo hàm làm thành phần “thắng” thì có thể giảm được overshot của xe. Thành phần “thắng” này chính là thành phần D (Derivative) trong bộ điều khiển PID mà chúng ta đang khảo sát. Thêm thành phần D này vào bộ điều khiển P hiện tại, chúng ta thu được bộ điều khiển PD như sau:

F=Kp\*e + Kd\*(de/dt) ​

Trong đó (de/dt) là vận tốc thay đổi của sai số e và Kd là một hằng số không âm gọi là hệ số D (Derivative gain).   
Sự hiện diện của thành phần D làm giảm overshot vật thể điều khiển khi nó tiến gần về vị trí cân bằng (vị trí ổn định), lực F gồm 2 thành phần Kp\*e > =0 (P) và Kd\*(de/dt) <=0 (D). Trong một số trường hợp thành phần D có giá trị lớn hơn thành phần P và lực F đổi chiều, “thắng” (hãm tốc) lại, yếu tố cần điều khiển (ví dụ vận tốc, vị trí...) của vật thể giảm mạnh ở gần vị trí cân bằng. Một vấn đề nảy sinh là nếu thành phần D quá lớn so với thành phần P hoặc bản thân thành phần P nhỏ thì khi tiến gần điểm cân bằng (chưa thật sự đến vị trí này), vật thể có thể dừng hẳn, thành phần D bằng 0 (vì sai số e không thay đổi nữa), lực F = Kp\*e. Trong khi Kp và e lúc này đều nhỏ nên lực F cũng nhỏ và có thể không thắng được lực ma sát tĩnh. Sai số e trong tình huống này gọi là steady state error (tạm dịch là sai số trạng thái tĩnh). Để tránh steady state error, người ta thêm vào bộ điều khiển một thành phần có chức năng “cộng dồn” sai số. Khi steady state error xảy ra, 2 thành phần P và D mất tác dụng, thành phần điều khiển mới sẽ “cộng dồn” sai số theo thời gian và làm tăng lực F theo thời gian. Đến một lúc nào đó, lực F đủ lớn để thắng ma sát tĩnh và đẩy vật tiến tiếp về điểm cân bằng. Thành phần “cộng dồn” này chính là thành phần I (Integral - tích phân) trong bộ điều khiển PID. Vì chúng ta điều biết, tích phân một đại lượng theo thời gian chính là tổng của đại lượng đó theo thời gian. Bộ điều khiển đến thời điểm này đã đầy đủ là PID:

F=Kp\*e + Kd\*(de/dt)+Ki\*∫edt ​

Như vậy, chức năng của từng thành phần trong bộ điều khiển PID giờ đã rõ. Tùy vào mục đích và đối tượng điều khiển mà bộ điều khiển PID có thể được lượt bớt để trở thành bộ điều khiển P, PI hoặc PD. Công việc chính của người thiết kế bộ điều khiển PID là chọn các hệ số Kp, Kd và Ki sao cho bộ điều khiển hoạt động tốt và ổn định (quá trình này gọi là PID gain tuning). Đây không phải là việc dễ dàng vì nó phụ thuộc vào nhiều yếu tố. Tôi tóm tắt một kinh nghiệm cơ bản khi chọn các hệ số cho PID như sau:  
- Chọn Kp trước: thử bộ điều khiển P với đối tượng thật (hoặc mô phỏng), điều chỉnh Kp sao cho thời gian đáp ứng đủ nhanh, chấp nhận overshot nhỏ.  
- Thêm thành phần D để loại overshot, tăng Kd từ từ, thử nghiệm và chọn giá trị thích hợp. Steady state error có thể sẽ xuất hiện.  
- Thêm thành phần I để giảm steady state error. Nên tăng Ki từ bé đến lớn để giảm steady state error đồng thời không để cho overshot xuất hiện trở lại.

**II. Phân tích yêu cầu:**

**1. Linh kiện sử dụng**

- Board mạch Arduino uno

- Module điều khiển động cơ L298

- Module Bluetooth HC-06

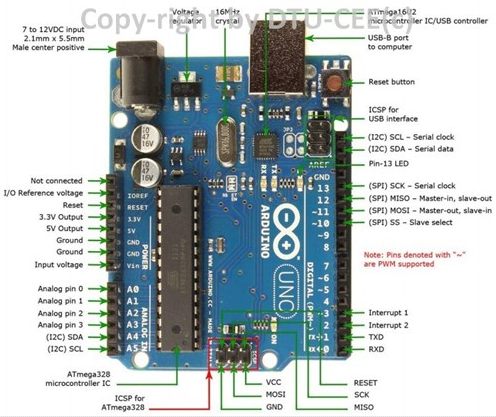
- Cặp động cơ GA25

- Dây nối

- Cảm biến gyro MPU6050

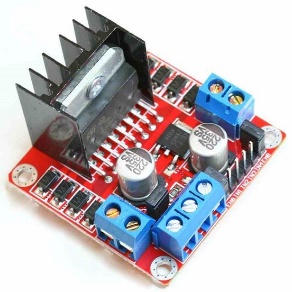
**2. Giới thiệu linh kiện**

**a. Mạch Arduino**



Arduino thật ra là một bo mạch vi xử lý được dùng để lập trình tương tác với các thiết bị phần cứng như cảm biến, động cơ, đèn hoặc các thiết bị khác. Đặc điểm nổi bật của Arduino là môi trường phát triển ứng dụng cực kỳ dễ sử dụng, với một ngôn ngữ lập trình có thể học một cách nhanh chóng ngay cả với người ít am hiểu về điện tử và lập trình. Và điều làm nên hiện tượng Arduino chính là mức giá rất thấp và tính chất nguồn mở từ phần cứng tới phần mềm.

**b. Module điều khiển động cơ L298:**

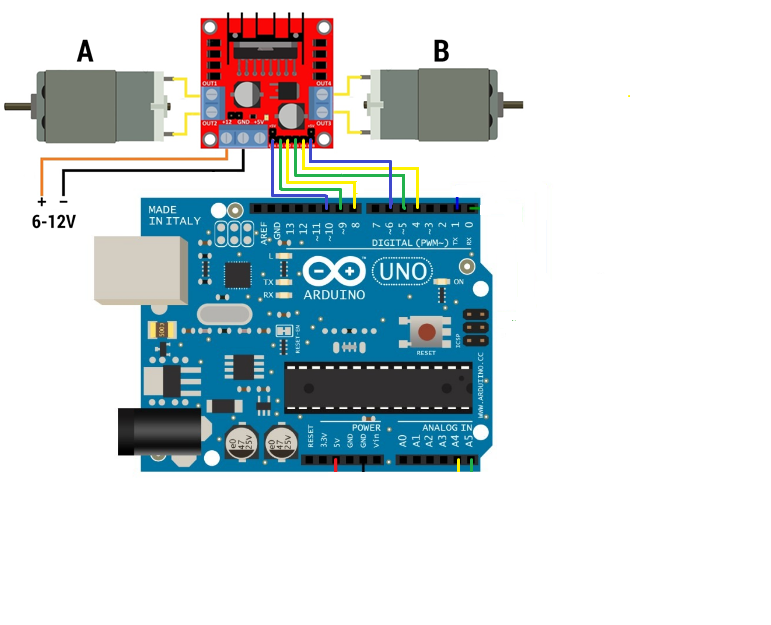


Module [L298](http://codientu.org/tags/74hc595/) sử dụng IC L298 đã tích hợp sẵn mạch cầu H, ứng dụng trong việc điều khiển cùng lúc 2 động cơ theo chiều quay bất kì, kết hợp với điều xung PWM có thể điều chỉnh tốc độ xoay của động cơ.

Điện áp điều khiển: +5 V ~ +12 V.

L298 gồm các chân:

* 12V power, 5V power: bên trong mạch có sử dụng IC điều hướng dòng điện, hoạt động với điện áp 5V, khi nối nguồn 12V mạch sẽ hoạt động với nguồn 12V, tuy nhiên mạch sẽ điều hướng dòng điện.
  + Dòng 12V đẩy vào động cơ để hoạt động.
  + Hạ dòng điện xuống thành 5V và cấp nguồn cho IC.
* Như vậy khi cấp nguồn 9V: động cơ hoạt động với dòng 9V và IC của mạch sẽ hoạt động với dòng điện 5V. Việc thiết kế có nhằm điều hướng thành 2 dòng điện khác nhau giúp IC hoạt động ổn định và tách nguồn riêng biệt với động cơ.
* Power GND chân này là GND của nguồn cấp cho Động cơ.
* 2 Jump A enable và B enable, nếu bạn dùng Board để điều khiển động cơ bước thì giữa nguyên. Nếu điều khiển động cơ bình thường thì nối với chân PWM để điều khiển tốc độ.
* Gồm có 4 chân Input. IN1, IN2, IN3, IN4.
* Output A: nối với động cơ giảm tốc V1. Có hai chân +, -. Nếu nối ngược thì động cơ sẽ chạy ngược.

*Nối dây cơ bản cho module:*

**c. Module Bluetooth HC-06**

Bluetooth module SLAVE cho phép vi điều khiển kết nối với các thiết bị ngoại vi: smartphone, laptop, usb bluetooth... thông qua giao tiếp Serial gửi và nhận tín hiệu 2 chiều.

Module bluetooth được tích hợp trên board cho phép bạn sử dụng nguồn  từ 3.5V đến 6V cung cấp cho board mà không cần lo lắng về chênh lệch điện áp 3V - 5V gây hỏng board.

Bluetooth module gồm 6 chân theo thứ tự: KEY, VCC, GND, TX, RX, STATE.

Đây là module bluetooth SLAVE nghĩa là bạn không thể chủ động kết nối bằng vi điều khiển, mà cần sử dụng smartphone, laptop, bluetooth usb... để dò tín hiệu và kết nối (pair) từ smartphone, laptop, bluetooth usb... Sau khi pair thành công, bạn có thể gửi và nhận tín hiệu từ vi điều khiển đến các thiết bị này.

Hướng dẫn nối dây

|  |  |
| --- | --- |
| **Bluetooth module** | [**Arduino**](http://machtudong.vn/arduino) |
| VCC | 5V |
| GND | GND |
| TX | RX |
| RX | TX |

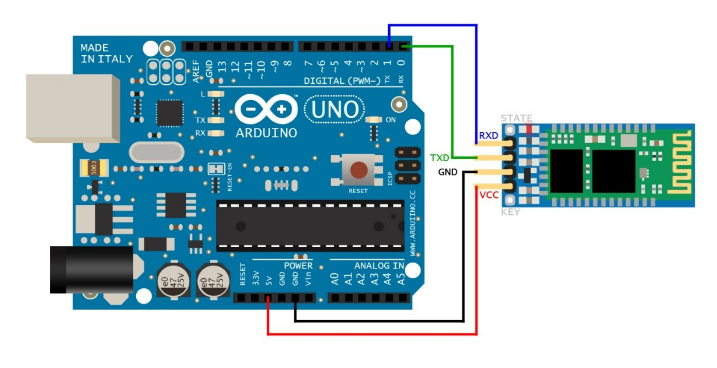
Mặc định, module bluetooth SLAVE sử dụng***baud rate là 9600, PIN là 1234***. Bạn có thể tùy chỉnh baud rate bằng tập lệnh AT

## Bluetooth module SLAVE có thể dễ dàng kết nối với các dòng smartphone Android

thông qua thư viện [AMARINO](http://www.amarino-toolkit.net/index.php/download.html).

Trước hết, cần tải về thư viện MeetAndroid dành cho Arduino .

Sau đó, cài đặt APP Amarino cho Android,  [file apk](http://code.google.com/p/amarino/downloads/detail?name=Amarino_2_v0_55.apk&can=2&q=).



**d. Cảm biến gyro MPU6050**

MPU-6050 là một chip tích hợp thiết bị dò chuyển động (Motion Tracking) 6-trục kết hợp một con quay hồi chuyển 3 trục(3-axis gyroscope), một bộ đo gia tốc 3 trục(3-axis accelerometer), và một bộ tính toán chuyển động tín hiệu số (Digital Motion Processor – DMP). MPU6050 đọc dữ liệu từ các cảm biến gia tốc và con quay hồi chuyển, sau đó lưu vào thanh ghi dữ liệu. Các vi xử lý/vi điều khiển có thể đọc thông tin này qua [bus I2C](http://iotbreaks.vn/tag/i2c/).

### Ứng dụng của MPU6050

MPU6050 thường được sử dụng ở máy bay đồ chơi, fly-cam, xe tự cân bằng… Ngoài ra, có thể bạn chưa biết, MPU6050 còn được dùng nhiều trong thiết bị chơi game cầm tay, bộ điều khiển dựa trên sự chuyển động cơ thể, điều khiển TV 3D, thiết bị đeo tay dùng cho sức khoẻ, thể dục thể thao… MPU6050 là một thành viên trong gia đình MPU60x0 được sử dụng rộng rải nhờ giá thành rẻ và dễ sử dụng.

### Các chức năng của MPU6050

#### Tính năng đo góc quay (Gyroscope)

* Cảm biến tốc độ gốc từ con quay hồi chuyển có thể output về X-, Y-, và Z-Axis với các scale ± 250 ± 500, ± 1000, và ± 2.000 ° / giây.
* Tích hợp 16-bit ADC cho phép lấy mẫu đồng thời các trục gyroscope

#### Tính năng đo gia tốc (Accelerometer)

* Cảm biến gia tốc 3 trục output về gia tốc theo 3 trục với các scale ± 2g, ± 4g, ± 8g và ± 16g.
* Tích hợp 16-bit ADC cho phép lấy mẫu đồng thời của gia tốc trong khi không cần đa bên ngoài.
* Orientation detection
* Tap detection
* Hỗ trợ interrupts (có thể lập trình được) khi có data hoặc FIFO bị tràn.

### Đặc tính điện của MPU6050

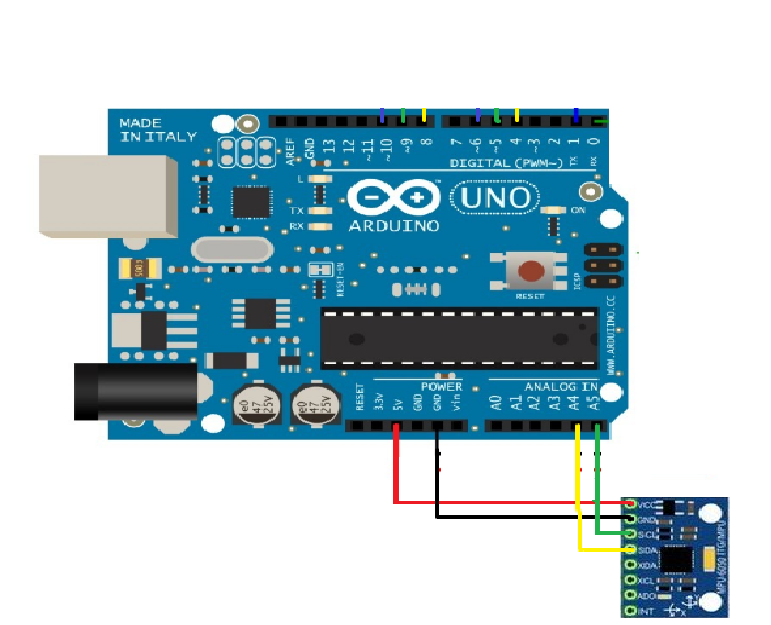
-Với tính năng Gyroscope:

Dòng điện tiêu thụ khi hoạt động : 3.6mA  
Dòng điện tiêu thụ khi standby: 5μA

-Với tính năng accelerometer

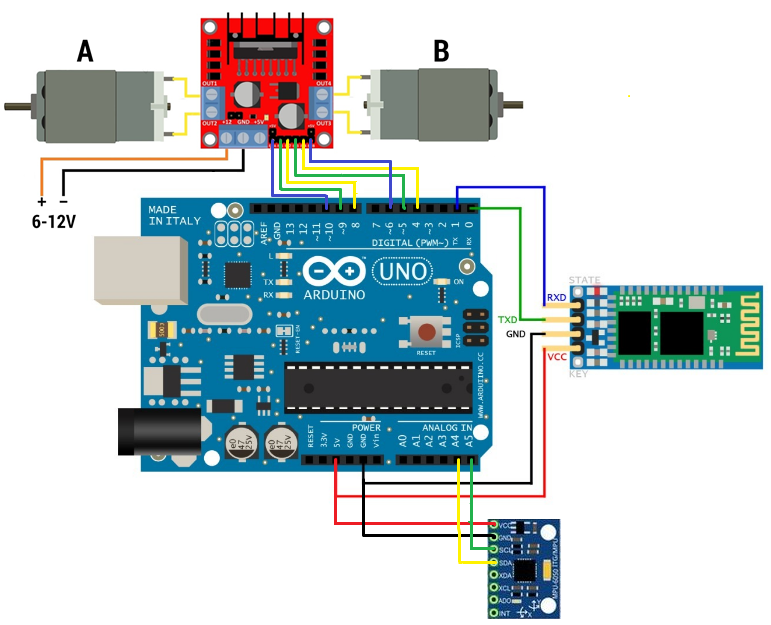
Dòng điện tiêu thụ khi hoạt động: 500μA  
Dòng điện tiêu thụ ở các tần số lấy mẫu khác nhau(để tiết kiệm pin): 10μA với 1.25Hz, 20μA vớ 5Hz, 60μA với 20Hz, 110μA với 40 Hz.

**Cách nối dây:**

****

**III. Thiết kế:**

**1. Sơ đồ tổng quát:**

****

**2. Cách ghép nối:**

Bluetooth nối với Arduino để kết nối với Android.

MPU 6050 nối với Arduino để xác định góc nghiêng.

2 động cơ nối với module L298N để giao tiếp với Arduino.

**IV. Triển khai và kết quả:**

**1. Lập trình cho từng linh kiện:**

**a. Lập trình điều khiển 2 động cơ thông qua Module L298N:**

Lập trình file header MotorController.h như sau:

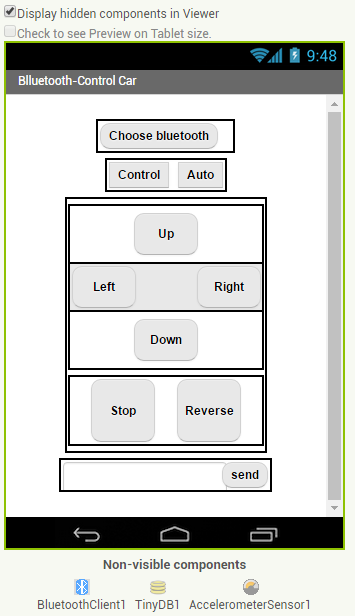
|  |
| --- |
| #ifndef \_MotorController\_H\_  #define \_MotorController\_H\_  #if ARDUINO < 100  #include "WProgram.h"  #else  #include "Arduino.h"  #endif  struct Wheel  {  byte pin1;  byte pin2;  byte speedPin; // must be PWN pin  byte pin1Value;  byte pin2Value;  byte speedValue;  };  /\* one wheel functionality \*/  void registerWheel(Wheel \*a);  void executeWheel(Wheel \*a);  void wheelGoUp(Wheel \*w);  void wheelGoDown(Wheel \*w);  void stopWheel(Wheel \*w);  void reverseWheel(Wheel \*a);  void setSpeedByPercent(Wheel \*w, byte speed);  void setSpeedByPWN(Wheel \*w, byte speed);  #endif |

Và MotorController.cpp cùng thư mục như sau:

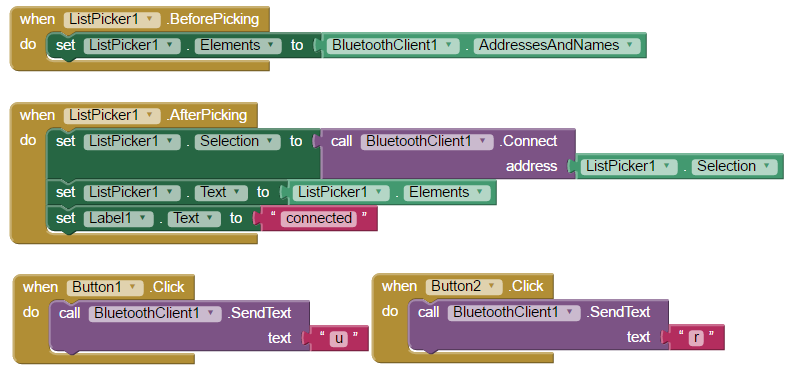
|  |
| --- |
| #include "MotorController.h"  void registerWheel(Wheel \*a)  {  pinMode(a->pin1, OUTPUT);  pinMode(a->pin2, OUTPUT);  pinMode(a->speedPin, OUTPUT);  }  void executeWheel(Wheel \*a)  {  digitalWrite(a->pin1, a->pin1Value);  digitalWrite(a->pin2, a->pin2Value);  analogWrite(a->speedPin, a->speedValue);  }  void wheelGoUp(Wheel \*w)  {  w->pin1Value = HIGH;  w->pin2Value = LOW;  executeWheel(w);  }  void wheelGoDown(Wheel \*w)  {  w->pin1Value = LOW;  w->pin2Value = HIGH;  executeWheel(w);  }  void stopWheel(Wheel \*w)  {  w->pin1Value = LOW;  w->pin2Value = LOW;  executeWheel(w);  }  void reverseWheel(Wheel \*w)  {  w->pin1Value = 1 - w->pin1Value;  w->pin2Value = 1 - w->pin2Value;  executeWheel(w);  }  void setSpeedByPercent(Wheel \*w, byte speed)  {  speed = map(speed, 0, 100, 0, 255);  setSpeedByPWN(w, speed);  }  void setSpeedByPWN(Wheel \*w, byte speed)  {  if (speed < 0) speed = 0;  if (speed > 255) speed = 255;  w->speedValue = speed;  } |

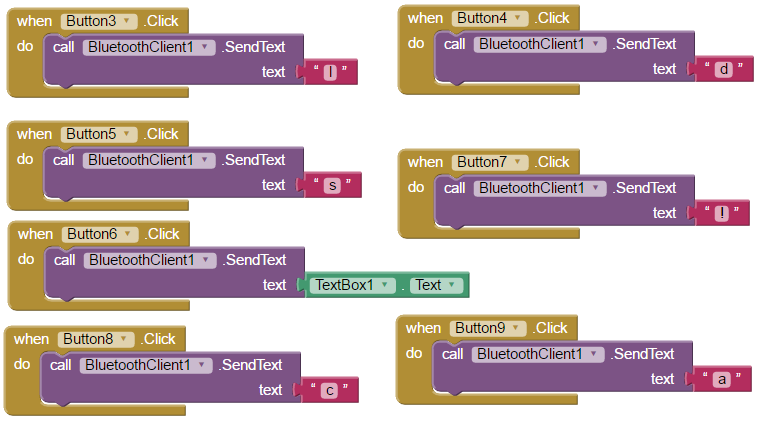
**b. Lập trình ứng dụng Android:**

***Design***



***Codeblocks***





**2.Source code cho MPU6050:**

MPU6050 được khai thác thông qua thư viên I2Cdev và MPU6050 của jrowberg

https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050

**3.Source code cho arduino:**

**a. thư viện PID:**

Thư viện PID được sử dụng của tác giả: Brett Beauregard tại địa chỉ:

https://github.com/br3ttb/Arduino-PID-Library

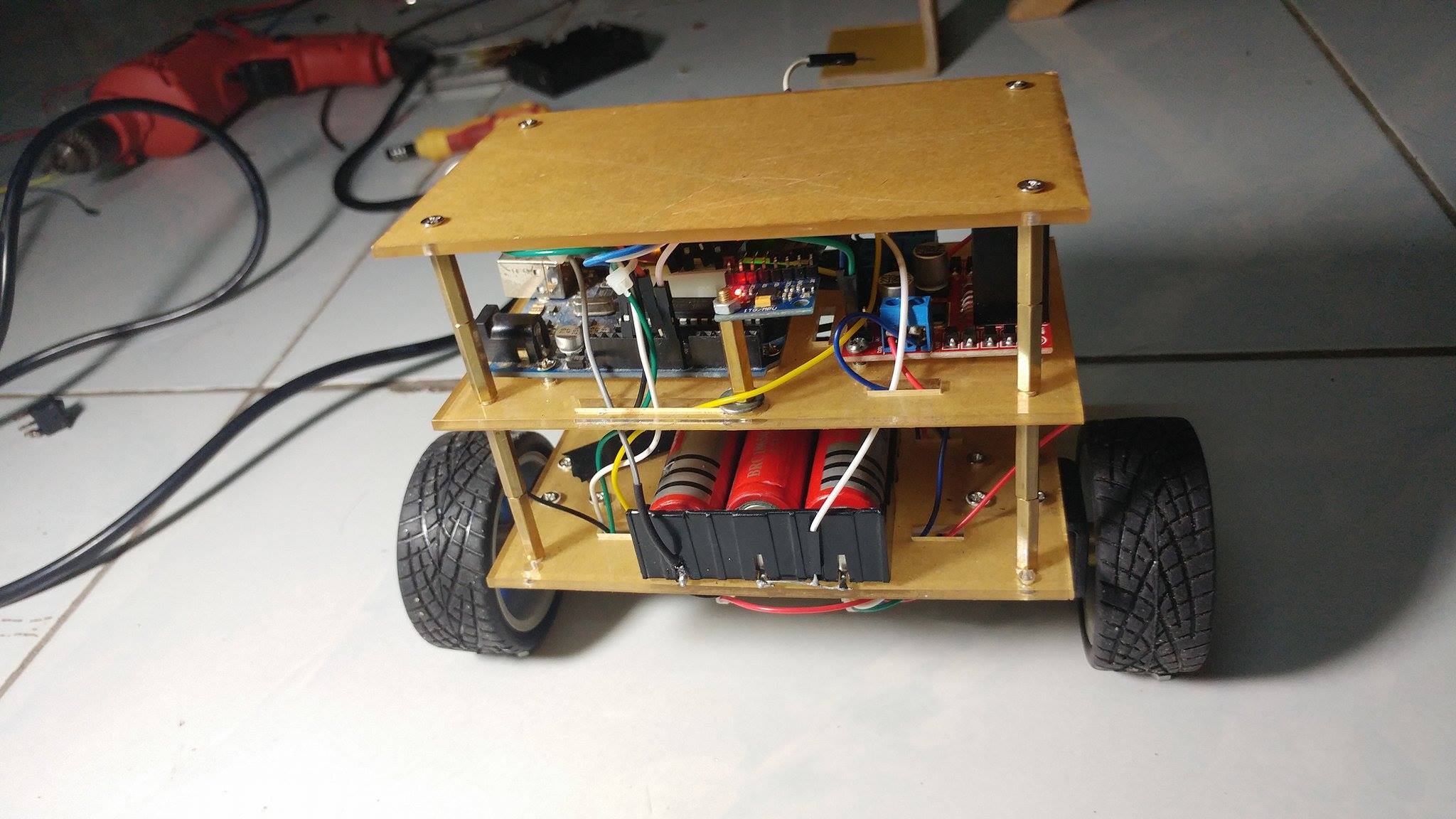
**Arduino Code:**

|  |
| --- |
| #include <MsTimer2.h>  #include <PID\_v1.h>  #include "I2Cdev.h"  #include "MotorController.h"  #include "MPU6050\_6Axis\_MotionApps20.h"  #if I2CDEV\_IMPLEMENTATION == I2CDEV\_ARDUINO\_WIRE  #include "Wire.h"  #endif  #define OUTPUT\_READABLE\_YAWPITCHROLL  #define GYRO\_INTERRUPT\_PIN 2  #define BALANCE\_KP 18  #define BALANCE\_KI 10  #define BALANCE\_KD 0.185  double Ku = 33;  double Pu = 0.22;  // End define and preprocessors  double input, out, setpoint;  PID pid(&input, &out, &setpoint, BALANCE\_KP, BALANCE\_KI, BALANCE\_KD, DIRECT);  MPU6050 mpu;  /\* pin1, pin 2, analog pin, value 1, value 2, speed (255) \*/  Wheel left = {10, 11, 9, 0, 1, 130}, //mask as black  right = {7, 6, 5, 0, 1, 130};  // MPU control/status vars  bool dmpReady = false; // set true if DMP init was successful  uint8\_t mpuIntStatus; // holds actual interrupt status byte from MPU  uint8\_t devStatus; // return status after each device operation (0 = success, !0 = error)  uint16\_t packetSize; // expected DMP packet size (default is 42 bytes)  uint16\_t fifoCount; // count of all bytes currently in FIFO  uint8\_t fifoBuffer[64]; // FIFO storage buffer  // orientation/motion vars  Quaternion q; // [w, x, y, z] quaternion container  VectorInt16 aa; // [x, y, z] accel sensor measurements  VectorInt16 aaReal; // [x, y, z] gravity-free accel sensor measurements  VectorInt16 aaWorld; // [x, y, z] world-frame accel sensor measurements  VectorFloat gravity; // [x, y, z] gravity vector  float euler[3]; // [psi, theta, phi] Euler angle container  float ypr[3]; // [yaw, pitch, roll] yaw/pitch/roll container and gravity vector  bool isGetSetpoint = false;  int count = 0;  int countPrint = 0;  // ================================================================  // === INTERRUPT DETECTION ROUTINE ===  // ================================================================  volatile bool mpuInterrupt = false; // indicates whether MPU interrupt pin has gone high  void dmpDataReady()  {  mpuInterrupt = true;  }  // ================================================================  // === CONTROL CAR TO BALANCE ===  // ================================================================  void control()  {  // call every 5ms  mpuInterrupt = true;  input = ypr[1] \* 180/M\_PI; // use pitch  pid.Compute();  double speed = out;  if (!isGetSetpoint || abs(speed) < 20)  {  speed = 0;  }  // setSpeedByPWN(&left, fabs(speed)\*(190.0/175));  setSpeedByPWN(&left, abs(speed));  setSpeedByPWN(&right, abs(speed));    if (speed < 0)  {  wheelGoUp(&right);  wheelGoUp(&left);  }  else  {  wheelGoDown(&right);  wheelGoDown(&left);  }  Serial.print("Input: ");  Serial.print(input);  Serial.print(" Setpoint: ");  Serial.print(setpoint);  Serial.print(" Speed: ");  Serial.println(speed);    }  // ================================================================  // === INITIAL SETUP ===  // ================================================================  void setup()  {  /\* init motor \*/  registerWheel(&left);  registerWheel(&right);    /\* initial setting for pid\*/  pid.SetMode(AUTOMATIC);  pid.SetOutputLimits(-255, 255);  pid.SetSampleTime(5); // milisec  double kp, ki, kd;  kp=0.6\*Ku;  ki=2\*(kp/Pu);  kd=(kp\*Pu)/8;  pid.SetTunings(kp, ki, kd);  // ToDo: set Kp, Ki, Kd  setpoint = 0;  // join I2C bus (I2Cdev library doesn't do this automatically)  #if I2CDEV\_IMPLEMENTATION == I2CDEV\_ARDUINO\_WIRE  Wire.begin();  Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having compilation difficulties  #elif I2CDEV\_IMPLEMENTATION == I2CDEV\_BUILTIN\_FASTWIRE  Fastwire::setup(400, true);  #endif  // initialize serial communication  // (115200 chosen because it is required for Teapot Demo output, but it's  // really up to you depending on your project)  Serial.begin(115200);  // initialize device  Serial.println(F("Initializing I2C devices..."));  mpu.initialize();  pinMode(GYRO\_INTERRUPT\_PIN, INPUT);  // verify connection  Serial.println(F("Testing device connections..."));  Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));  // load and configure the DMP  Serial.println(F("Initializing DMP..."));  devStatus = mpu.dmpInitialize();  // supply your own gyro offsets here, scaled for min sensitivity  mpu.setXGyroOffset(220);  mpu.setYGyroOffset(76);  mpu.setZGyroOffset(-85);  mpu.setZAccelOffset(1788); // 1688 factory default for my test chip  // make sure it worked (returns 0 if so)  if (devStatus == 0)  {  // turn on the DMP, now that it's ready  Serial.println(F("Enabling DMP..."));  mpu.setDMPEnabled(true);  // enable Arduino interrupt detection  Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0)..."));  attachInterrupt(digitalPinToInterrupt(GYRO\_INTERRUPT\_PIN), dmpDataReady, RISING);  mpuIntStatus = mpu.getIntStatus();  // set our DMP Ready flag so the main loop() function knows it's okay to use it  Serial.println(F("DMP ready! Waiting for first interrupt..."));  dmpReady = true;  // get expected DMP packet size for later comparison  packetSize = mpu.dmpGetFIFOPacketSize();  } else  {  // ERROR!  // 1 = initial memory load failed  // 2 = DMP configuration updates failed  // (if it's going to break, usually the code will be 1)  Serial.print(F("DMP Initialization failed (code "));  Serial.print(devStatus);  Serial.println(F(")"));  }  // ToDo: Set timer for pid calculation and encoder interrupt    MsTimer2::set(5, control); // 5ms period  MsTimer2::start();  // attachInterrupt(1, encoder, RISING);    }  // ================================================================  // === MAIN PROGRAM LOOP ===  // ================================================================  void loop()  {  // if programming failed, don't try to do anything  if (!dmpReady) return;  // wait for MPU interrupt or extra packet(s) available  while (!mpuInterrupt && fifoCount < packetSize)  {  }  // reset interrupt flag and get INT\_STATUS byte  mpuInterrupt = false;  mpuIntStatus = mpu.getIntStatus();  // get current FIFO count  fifoCount = mpu.getFIFOCount();  // check for overflow (this should never happen unless our code is too inefficient)  if ((mpuIntStatus & 0x10) || fifoCount == 1024)  {  // reset so we can continue cleanly  mpu.resetFIFO();  Serial.println(F("FIFO overflow!"));  // otherwise, check for DMP data ready interrupt (this should happen frequently)  } else if (mpuIntStatus & 0x02)  {  // wait for correct available data length, should be a VERY short wait  while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();  // read a packet from FIFO  mpu.getFIFOBytes(fifoBuffer, packetSize);    // track FIFO count here in case there is > 1 packet available  // (this lets us immediately read more without waiting for an interrupt)  fifoCount -= packetSize;  #ifdef OUTPUT\_READABLE\_YAWPITCHROLL  countPrint++;  // Serial.print(countPrint);  // display Euler angles in degrees  mpu.dmpGetQuaternion(&q, fifoBuffer);  mpu.dmpGetGravity(&gravity, &q);  mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);  /\*  Serial.print(" ypr\t");  Serial.print(ypr[0] \* 180/M\_PI);  Serial.print("\t");  Serial.print(ypr[1] \* 180/M\_PI);  Serial.print("\t");  Serial.println(ypr[2] \* 180/M\_PI);  \*/  count++;  if (count == 700)  {  setpoint = ypr[1] \* 180/M\_PI;  isGetSetpoint = true;  }  #endif  }  } |

**SẢN PHẨM**

****

Hình: Sản phẩm hoàn thiện

****

Hình: Sản phẩm hoàn thiện

**KẾT LUẬN**

Sau một thời gian nghiên cứu và làm việc, nhóm em đã rút ra được những kết luận như sau.

Về ưu điểm:

- Nắm được kỹ thuật điều khiển PID để áp dụng cho các xử lý thời gian thực

- Tìm hiểu về lập trình Android bằng công cụ App Inventor.

- Tìm hiểu về MPU6050 và cách sử dụng để lấy các giá trị góc nghiêng và gia tốc

- Ứng dụng nhiều module và thuật toán để tạo ra một sản phẩm hoàn chỉnh.

Tuy nhiên đồ án của chúng em vẫn còn một số thiếu sót:

- Xe đôi khi thiếu ổn định

- Bề ngoài của xe vẫn chưa được đẹp lắm

Hướng phát triển:

- Hướng đến việc tích hợp thêm camera hoặc các cảm biến giúp cho việc dùng thiết bị này cho các hoạt động thám hiểm, khảo sát những nới nguy hiểm, con người không tới được.

Chúng em xin cảm ơn thầy Trần Thế Vũ đã giúp đỡ chúng em hoàn thành đồ án này.

**TÀI LIỆU THAM KHẢO**

1. Arduino.vn - Cộng đồng Arduino Việt Nam
2. <http://www.instructables.com/> - tài liệu về MPu6050, xe tự cân bằng
3. Và nhiều hướng dẫn khác trên internet về PID