

# Lesson

# 1

## Moving Control



## Introduction:

In this lesson, there are three parts. First, we will teach you how to drive the motor and make our car run.

## Tumbller Direction Control

- 1.1 Hardware Design
- 1.2 Software Design
- 1.3 Upload Validation

## Preparations:

- one car(with a battery)
- one USB cable

## Part 1: Tumbler Moving Control:

### 1.1 Hardware Design

In our kit, the motion of the car is controlled by two DC motors.

(As shown in figure 1.1.1~1.1.3)

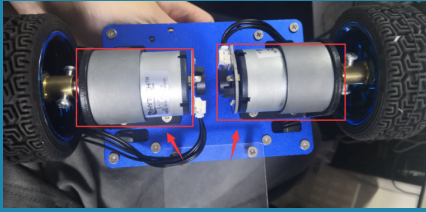


Figure 1.1.1 The real object of motor



Figure 1.1.2 The real object of motor

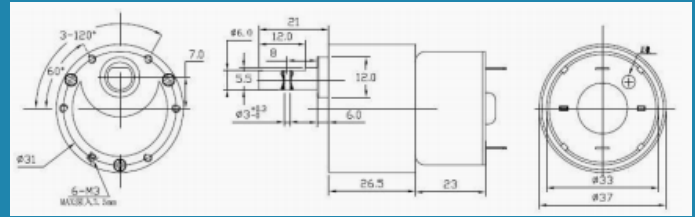


Figure 1.1.3 The structure of motor

However, DC motor is inductive load with large current (load with inductive parameters), while the IO of Arduino Nano, the single chip microcomputer we use, has weak load capacity (i.e. output current capacity), and its driving capacity is not enough to supply enough electric current for motor to rotate.

Therefore, we choose to use TB6612FENG motor driver chip.(As shown in figure 1.1.4~1.1.5):

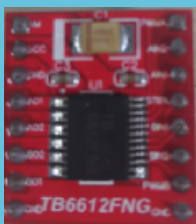


Figure 1.1.4 The real object of motor



Figure 1.1.5 Model picture of TB6612FENG

**TB6612FENG** is a DC motor driver, which has a high current MOSFET-H bridge structure. Two output channels drive two motors at the same time.

It has two advantages:

1. Supply enough current to the motor.
2. Act as isolation to prevent the impact current generated by the motor from damaging the control device.

Maybe you are more familiar with L298N, in fact, the use of the two is basically the same. Compared with L298N's heat consumption and peripheral diode continuous current circuit, it does not need additional heat sink, the peripheral circuit is simple, only external power filter capacitor is needed. It can directly drive the motor, which is beneficial to reduce the system size and the frequency up to 100kHz is enough to meet most of our needs.

## Features

- Power supply voltage:  $V_M = 15\text{ V}(\text{Max})$
- Output current:  $I_{OUT}=1.2\text{ A}(\text{ave}) / 3.2\text{ A}(\text{peak})$
- Output low ON resistor:  $0.5\Omega$  (upper+lower Typ. @ $V_M \geq 5\text{ V}$ )
- Standby (energy-saving) system
- CW / CCW / short brake / stop function modes
- Built-in thermal shutdown circuit and low voltage detecting circuit
- Small panel package (SSOP24: 0.65 mm Lead pitch)
- Response to PB-free packaging

According to the TB6612 IC data sheet: Open the [Related chip information](#) -> TB6612FNG

MOTOR A :

Truth table (Motor Steering)			
AIN1	0	1	0
AIN2	0	0	1
	STOP	forward- rotating	reversal reverse

The motor B is similar to motor A.

Red part controls motor A, blue part controls motor B, AO1, AO2 are respectively connected to + and -.

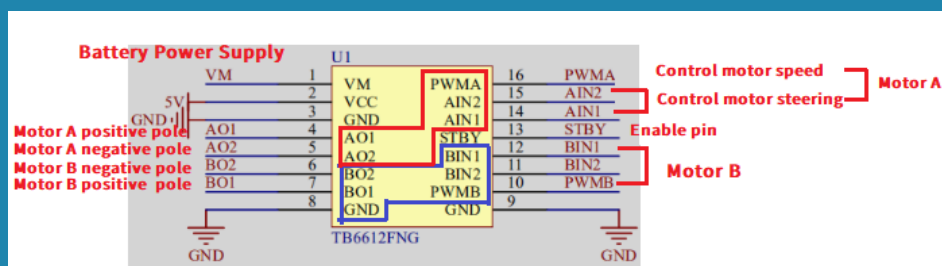


Figure 1.1.6 TB6612 module wiring diagram

In this kit,

MOTOR A:

PWMA is connected to D5

AIN1 is connected to D7

MOTOR B:

PWMB is connected to D6.

Bin1 is connected to D12. (As shown in figure 1.1.6)

According to the truth table, when the motor is rotating, AIN1 is high level, AIN2 is always low level, and when AIN1 is low level, AIN2 is always high level.

When the motor rotates, the input signals of AIN1 and AIN2 are always opposite. So in order to save IO resources, we add SN74LVC2G14 Two-Way Schmitt Trigger Inverter (As shown in figure 1.1.7), the input terminal is connected to AIN1, and the output terminal is connected to AIN2, so that the level state of the input terminal and the output terminal is always inverted.

Same for Bin1 and BIN2.

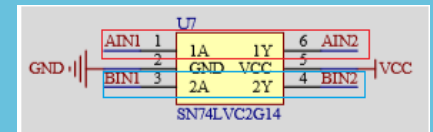


Figure 1.1.7 The schematic of SN74LVC2G14

In order to achieve the balance function in the later courses, we need to obtain the motor speed and rotation direction, so we also need to drive two hall sensors on the motor to achieve the speed detection and rotation direction detection function.(As shown in figure 1.1.8~1.1.9)

The reduction ratio is 1:30, which means that the input speed of the motor is 30 and the output shaft is 1 revolution.

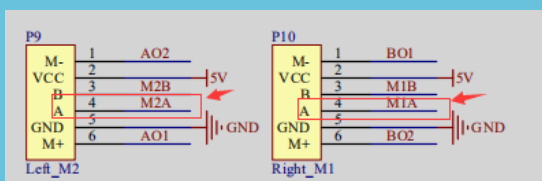


Figure 1.1.8 Encoder drive pin on motor

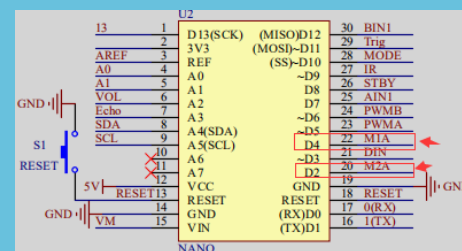


Figure 1.1.9 Corresponding pin connected to Nano

## 1.2 Software Design:

Please open [ELEGOO Tumbler Self-Balancing Car Tutorial -> Lesson 1 Moving Control -> Move->Balanced\\_Car->Balanced\\_Car.ino](#) in the current folder

We can see four files in this program: (As shown in figure 1.2.1)



(As shown in figure 1.2.1)

Please open the [Program FAQ](#) in the current folder and operate according to the contents.

First, let's have a look at the definition of related pins:

```
// in Motor.h

/*Motor pin*/
#define AIN1 7
#define PWMA_LEFT 5
#define BIN1 12
#define PWMB_RIGHT 6
#define STBY_PIN 8

/*Encoder measuring speed pin*/
#define ENCODER_LEFT_A_PIN 2
#define ENCODER_RIGHT_A_PIN 4
```

And initialize these pins. Only after initialization can these pins be operated.

```
//in Motor.cpp

void Motor::Pin_init()
{
    pinMode(AIN1, OUTPUT);
    pinMode(BIN1, OUTPUT);
    pinMode(PWMA_LEFT, OUTPUT);
    pinMode(PWMB_RIGHT, OUTPUT);
    pinMode(STBY_PIN, OUTPUT);
    digitalWrite(STBY_PIN, HIGH);
}
```

And call it in 'setup'.

```
void setup()
{
    Motor.Pin_init();
    .....
}
```

When the pin initialization is completed, we can control the motor by changing the state of the pin, and let the motor rotate in our ideal way.

Let's take a look at the overall implementation framework of the car motion. (As shown in figure 1.2.2)

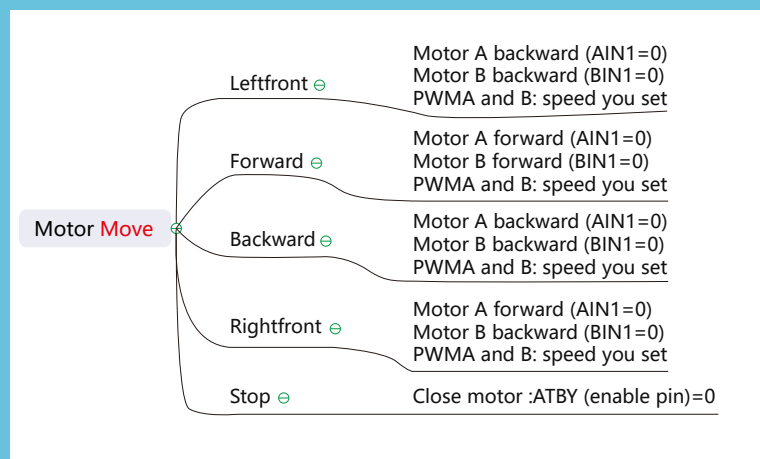


Figure 1.2.2 The Tumbler motion frame diagram

For specific implementation functions: (As shown in figure 1.2.3)

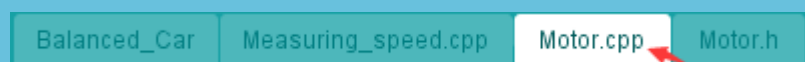


Figure 1.2.3 Specific implementation functions

For details of speed measurement, please refer to [Measuring\\_speed.cpp](#)

Speed calculation method: use the capture value (the number of pulses output in one second) / encoder lines (the number of pulses output in one revolution) / motor subtraction ratio (the ratio between the number of turns of internal motor and the number of turns of motor output shaft, i.e. reduction gear ratio)

It can be seen from the formula that the rotation speed is directly proportional to the number of pulses output by the encoder in a certain period of time, so we only need to obtain the number of pulses.

For specific implementation functions, please see:



First, we add the library file "PinChangeInt.h" of external interrupt. Then we initialize the external interrupt: set the external interrupt pin, set the external interrupt function name, and set the external interrupt as the level change interrupt.

```
//in Measuring_speed.cpp  
  
#include "PinChangeInt.h"
```

```
//in Measuring_speed.cpp  
  
void Motor::Encoder_init()  
{  
    attachInterrupt(digitalPinToInterrupt(ENCODER_LEFT_A_PIN), EncoderCountLeftA, CHANGE);  
    attachPinChangeInterrupt(ENCODER_RIGHT_A_PIN, EncoderCountRightA, CHANGE);  
}
```

And call it in 'setup'.

```
void setup()  
{  
    .....  
    Motor.Encoder_init();  
    .....  
}
```

Finally, let's have a look at the loop () function

We will analyze the content of the loop () function by disassembling it.

```
for(int i = 0,j=0; i < 4,j<8; i++)  
{  
    (Motor.*(Motor.MOVE[i]))(SPEED);  
    delay(2000);  
}
```

The meaning of this paragraph is actually the same as the one above.

```
Motor.Forward(SPEED);
delay(2000);
Motor.Back(SPEED);
delay(2000);
Motor.Left(SPEED);
delay(2000);
Motor.Right(SPEED);
delay(2000);
```

Because we found that the parameters of these four functions are the same, in order to make the `loop()` function more concise, we can write it as an array of function pointers in the class, initialize it in the constructor, and call these four different functions in the `for()` loop.

```
//in Motor.h

public:
    Motor();
    .....
    void (Motor::*MOVE[4])(int speed);
    .....
```

```
//in Motor.cpp

Motor::Motor()
{
    MOVE[0] = &Motor::Forward;
    MOVE[1] = &Motor::Back;
    MOVE[2] = &Motor::Left;
    MOVE[3] = &Motor::Right;
}
```

In the end, we print out the number of left and right encoder pulse signals obtained in the external interrupt function every 2 seconds.

```
//in Motor.cpp

for(int i = 0,j=0; i < 4,j<8; i++)
{
    (Motor.*(Motor.MOVE[i]))(SPEED);
    delay(2000);
    Serial.print(strbuf[j++]);
    Serial.println(Motor.encoder_count_right_a);
    Serial.print(strbuf[j++]);
    Serial.println(Motor.encoder_count_left_a);
    Motor.encoder_count_right_a=0;
    Motor.encoder_count_left_a=0;
}
```



## 1.3 Upload Validation

First, turn on the power, connect the USB cable, upload the program, and don't disconnect the USB in this time, turn on the serial Monitor (as shown in Figure 1.3.1), you will see the number of encoder pulse signals in two seconds (as shown in Figure 1.3.2), then disconnect the USB cable, put the balance car on the ground, although it will not be balanced at this time, but it has been able to move around according to our idea!



Figure 1.3.1 Open the Serial Monitor

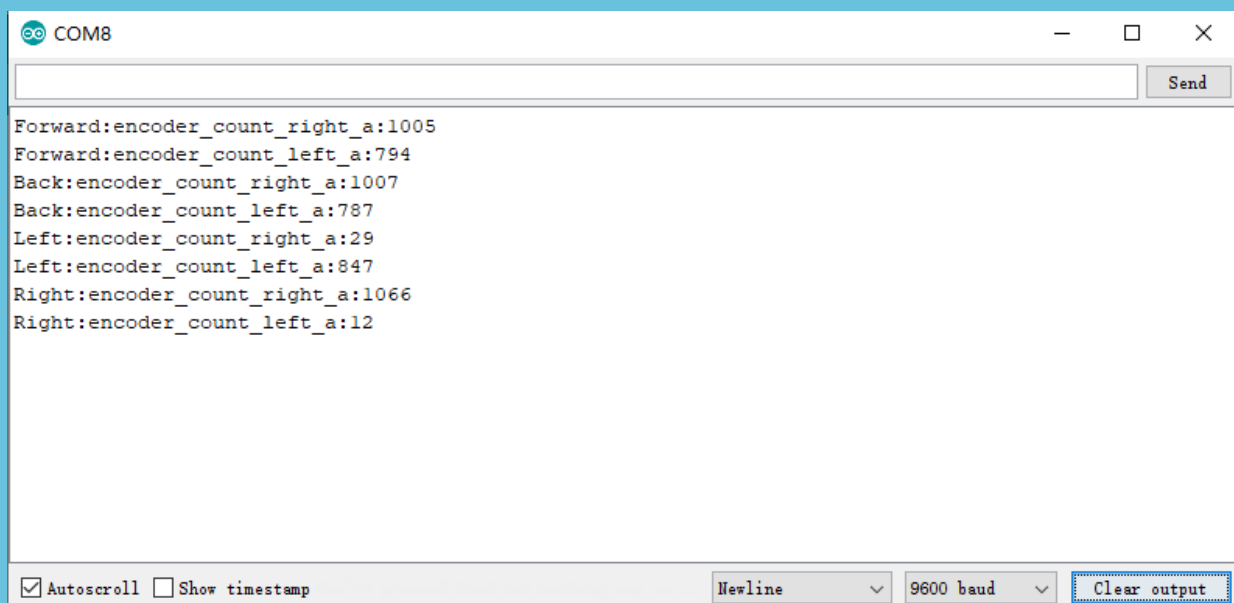


Figure 1.3.2 Open the Serial Monitor interface