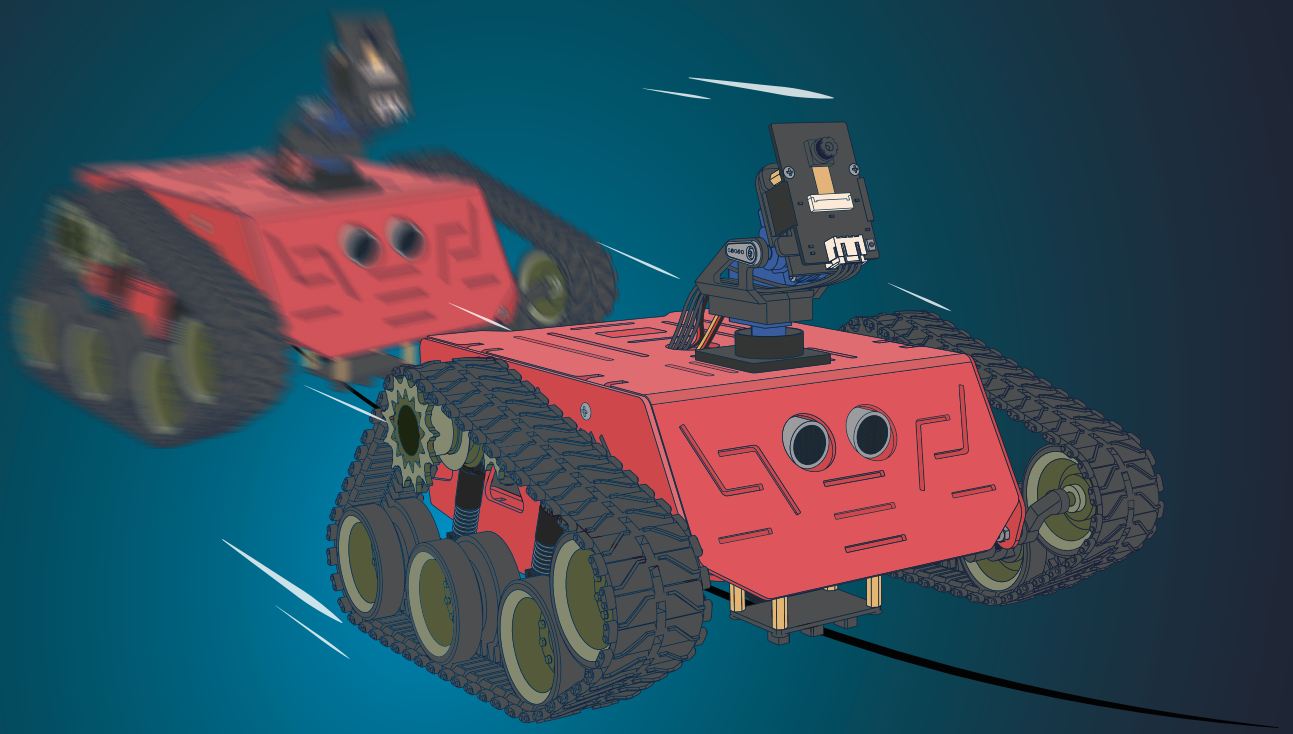
ELEGOO

2 Lesson

Move



ELEGOO.COM

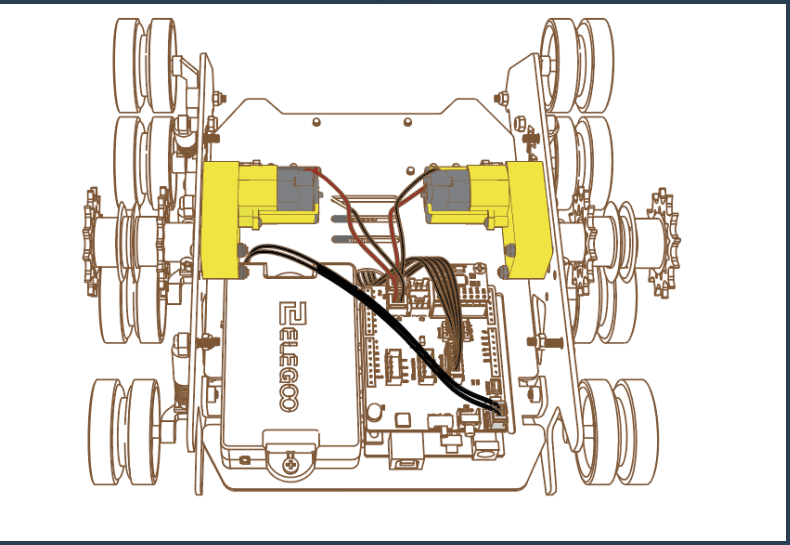
Introduction:

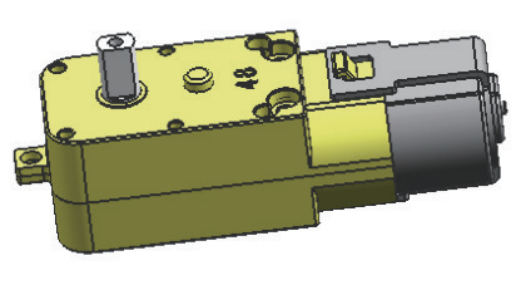
In this lesson, we will teach you how to control the car as you wish through the motor.

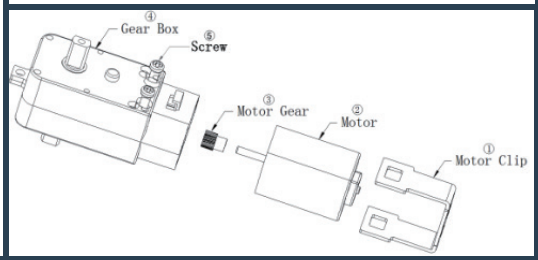
Material Preparation:

A Conqueror Robot Tank Car (with battery)
A USB Cable

Let’s take a look at the two motors used in our kit.

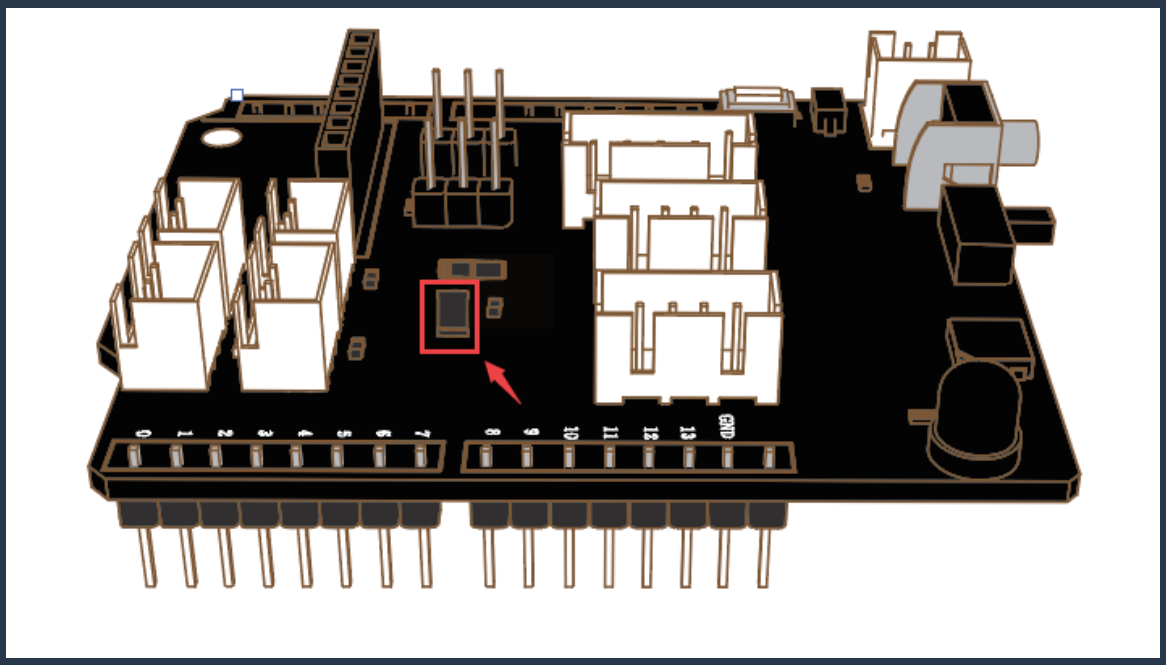


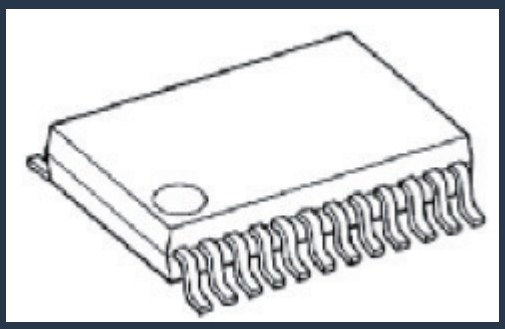




DC motor is the inductive load of large current (load with inductance parameters), while the IO port of Arduino UNO single-chip microcomputer we used has weak load capacity (output current capacity), and the driving capacity is not enough to supply enough current for the motor to rotate.

Therefore, we chose to use the motor driver chip DRV8835. (Although there may be slight differences in the performance of different driver chips, they are generally the same in usage.):





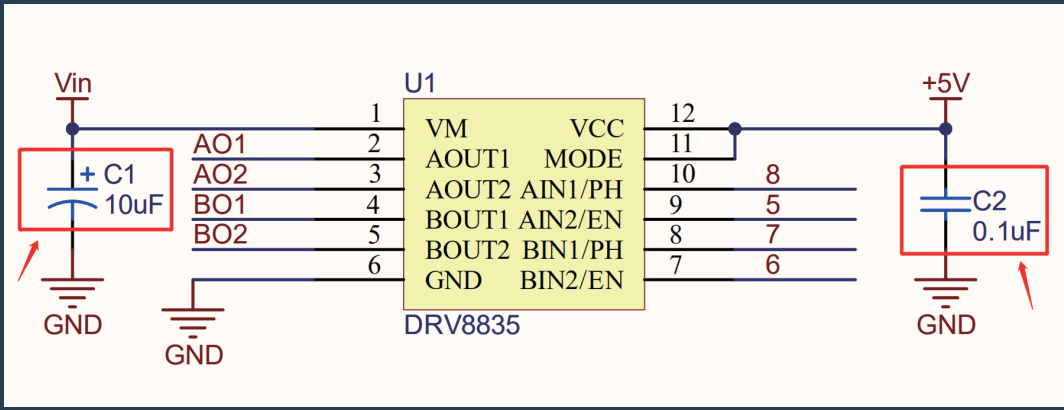
The DRV8835 is a DC motor driver device with a high-current MOSFET-H bridge structure and dual-channel output to drive two motors simultaneously.

It has two advantages:

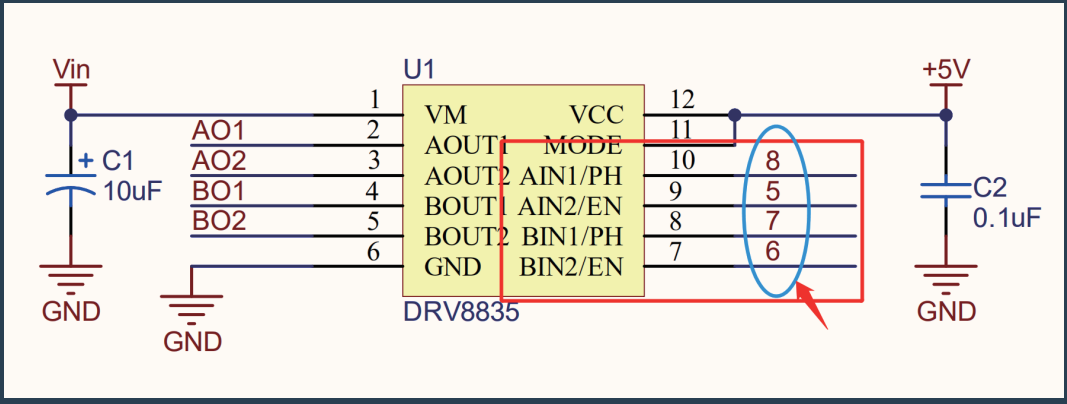
1.DRV8835 Supplies enough current to the motor. Each H bridge of the DRV8835 can provide up to 1.5A of output current. It operates within the motor power voltage range of 0V to 11V and the device power voltage range of 2V to 7V.

2.DRV8835 acts as isolation to avoid damage of control devices by the impulse current generated by the motor.
Perhaps everyone is more familiar with L298N. In fact, the use of the two is basically the same. Compared with L298N's heat dissipation and peripheral diode freewheeling circuit, it does not require an external heat sink, and the peripheral circuit is simple. It only needs an external power filter capacitor. Directly driving the motor is beneficial to reduce the size of the system, and the frequency of up to 100KHZ is sufficient to meet most of our needs.

Please open the previous folder for details: Related chip information ->DRV8835 and ConquerorCar-Shield-V1.0



In addition, the pins that connect the DRV8835 to UNO can also be seen in this diagram.



EN stands for enable pin, which changes the motor speed by changing PWM (0~255).

PH stands for phase pin, and the rotation direction (positive/reverse) of the motor is changed by changing the pin level (0/1).

In motor A, PWMA is connected to the UNO~D5 pins and AIN1 is connected to the UNO D8 pins.

In motor B, PWMB is connected to the UNO~D6 pins and BIN1 is connected to the UNO D7 pins.

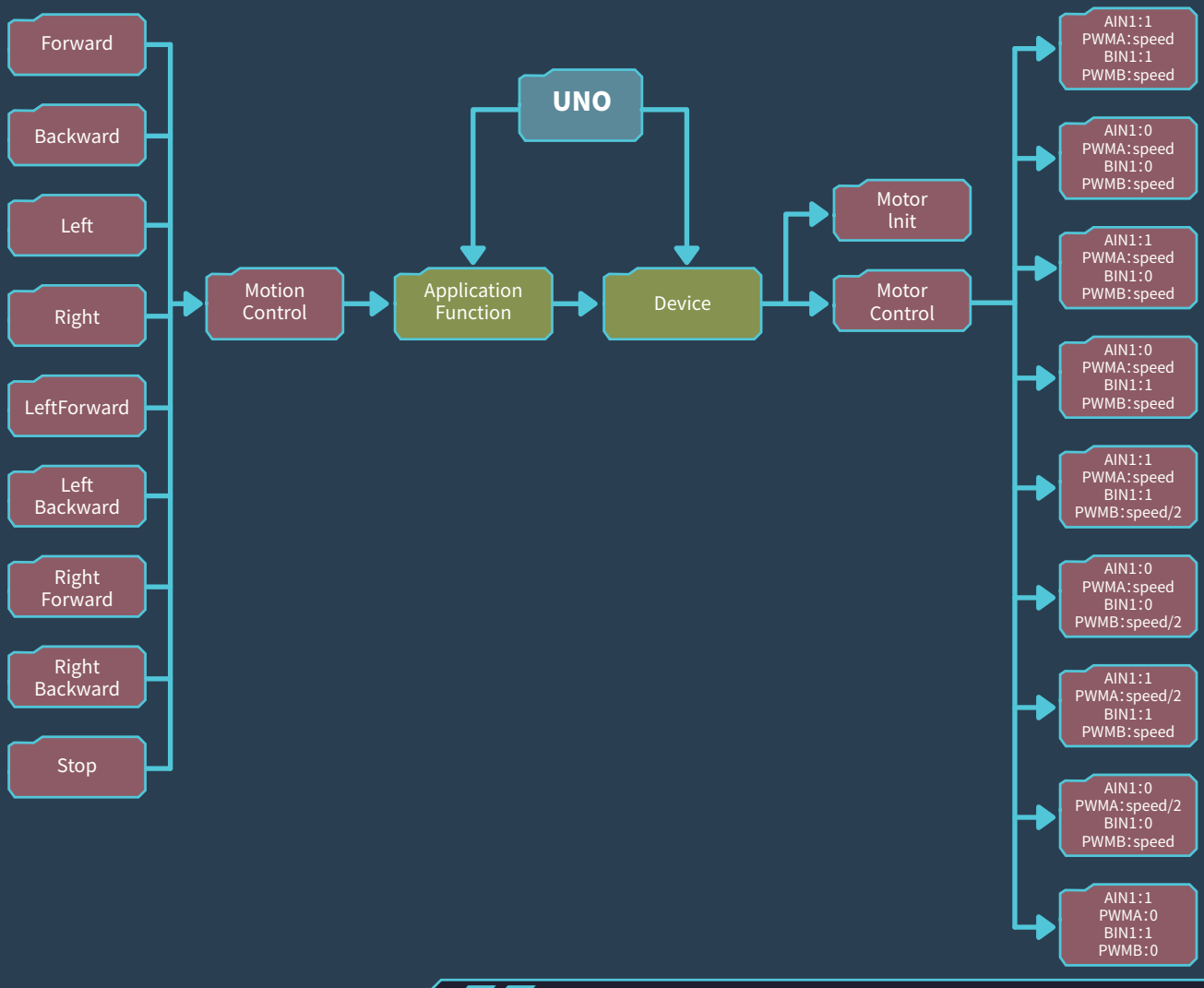
Finally, the driving method can be obtained from Related chip information ->DRV8835.

Table 2. IN/IN MODE

MODE	xIN1	xIN2	xOUT1	xOUT2	FUNCTION (DC MOTOR)
0	0	0	Z	Z	Coast
0	0	1	L	H	Reverse
0	1	0	H	L	Forward
0	1	1	L	L	Brake

Then, we can start writing program according to the flow chart below.
Please open Demo1 in the current folder.

Implementation principle of Move



First, we can see that there are four files.

Next, let's take a look at the definition of relevant pins and variables:

```
// in DeviceDriverSet_xxx0.h

/*Motor*/
class DeviceDriverSet_Motor
{
public:
 void DeviceDriverSet_Motor_Init(void);
#if _Test_DeviceDriverSet
 void DeviceDriverSet_Motor_Test(void);
#endif
 void DeviceDriverSet_Motor_control(boolean direction_A, uint8_t speed_A,
                                    boolean direction_B, uint8_t speed_B,
                                    boolean controlED
 );
private:

#define PIN_Motor_PWMA 5
#define PIN_Motor_PWMB 6
#define PIN_Motor_BIN_1 7
#define PIN_Motor_AIN_1 8

public:
#define speed_Max 255
#define direction_just true
#define direction_back false
#define direction_void 3

#define Duration_enable true
#define Duration_disable false
#define control_enable true
#define control_disable false
};
```

Define the maximum speed of motor rotation：

```
// in DeviceDriverSet_xxx0.h

#define speed_Max 255
```

Define flag bit

Motor rotation direction flag

```
// in DeviceDriverSet_xxx0.h

#define direction_just true
#define direction_back false
#define direction_void 3
```

Other functional flags

```
// in DeviceDriverSet_xxx0.h

#define Duration_enable true
#define Duration_disable false
#define control_enable true
#define control_disable false
```

These pins need to be initialized before they can be operated.

```
//in DeviceDriverSet_xxx0.cpp

extern DeviceDriverSet_Motor AppMotor;
void DeviceDriverSet_Motor::DeviceDriverSet_Motor_Init(void)
{
  pinMode(PIN_Motor_PWMA, OUTPUT);
  pinMode(PIN_Motor_PWMB, OUTPUT);
  pinMode(PIN_Motor_AIN_1, OUTPUT);
  pinMode(PIN_Motor_BIN_1, OUTPUT);
}
```

And call it in 'setup'.

```
DeviceDriverSet_Motor AppMotor;
void setup()
{
  AppMotor.DeviceDriverSet_Motor_Init();
}
```

After initialize the pins, we can change the status of the pins to control the motor and make the motor rotate as we want.

1. Car Motion

```
//in ApplicationFunctionSet_xxx0.cpp
static void ApplicationFunctionSet_ConquerorCarMotionControl
(ConquerorCarMotionControl direction, uint8_t is_speed)
```

parameter:
direction: Passing motion direction control sequence.
is_speed：0~255

```
//in ApplicationFunctionSet_xxx0.cpp

/*Motion direction control sequence*/
enum ConquerorCarMotionControl
{
  Forward,      //(1)
  Backward,     //(2)
  Left,         //(3)
  Right,        //(4)
  LeftForward,  //(5)
  LeftBackward, //(6)
  RightForward, //(7)
  RightBackward,//(8)
  stop_it       //(9)
};
```

2. Motor control

```
//in DeviceDriverSet_xxx0.cpp
void DeviceDriverSet_Motor::DeviceDriverSet_Motor_control
(boolean direction_A, uint8_t speed_A, //motor A
boolean direction_B, uint8_t speed_B, //motor B
boolean controlED              //AB enable true
                                 )
```

parameter:

direction_A,direction_B: Passing motor rotation direction flag.

speed_A,speed_B: range 0~255

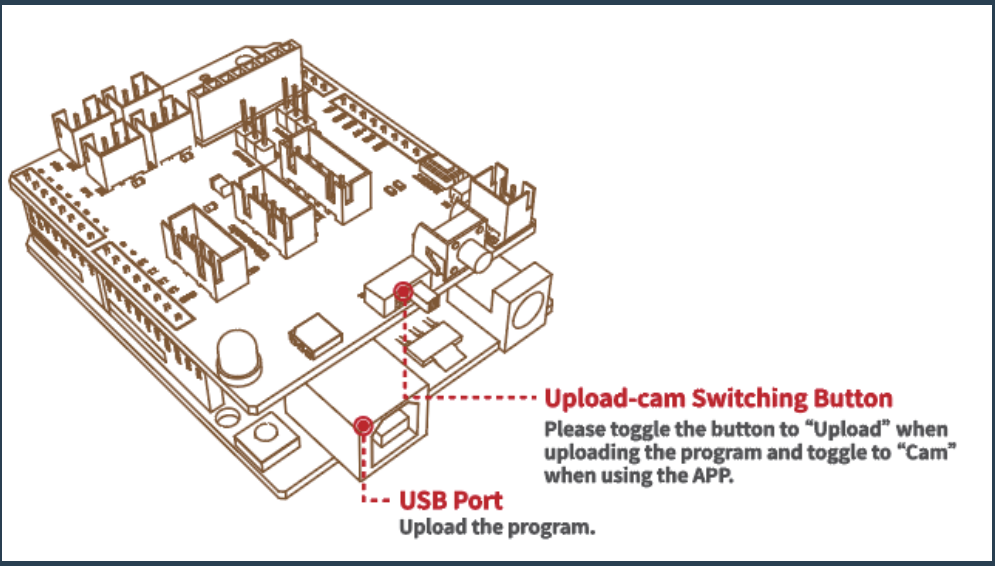
```
// in DeviceDriverSet_xxx0.h

#define direction_just true    //motor forward
#define direction_back false   //motor backward
#define direction_void 3
```

In the end, make the car move in each direction for one second after 2s waiting.

```
void setup() {
  AppMotor.DeviceDriverSet_Motor_Init();
  delay(2000);
  for (Application_ConquerorCarxxx0.Motion_Control = 0;
Application_ConquerorCarxxx0.Motion_Control < 9;
Application_ConquerorCarxxx0.
Motion_Control = Application_ConquerorCarxxx0.
Motion_Control + 1)
  {
    ApplicationFunctionSet_ConquerorCarMotionControl
(Application_ConquerorCarxxx0.
Motion_Control /*direction*/, 255 /*speed*/);
    delay(1000);
  }
}
```

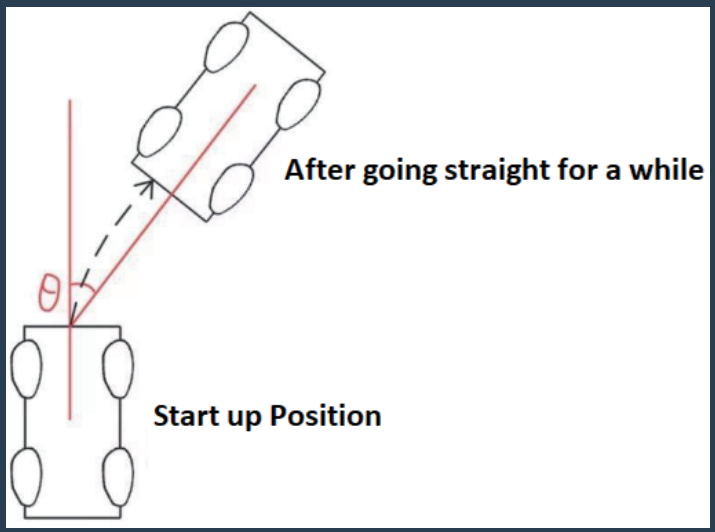
Upload program. (Please toggle the “Upload-Cam” button to “Upload” when uploading the program.) Place the car on the ground and turn on the switch after the program has been uploaded successfully. Then, you will see the car move forward, move backward, turn left, turn right, turn front-left, left-rear, turn front-right, turn right-rear at 1s intervals and finally stop.



When Demo1 was executed, if you are careful enough, you may notice that there are some deviations in the direction of the Conqueror Car as it moves forward and backward. On the one hand, different terrain has different friction, on the other hand, even the same type of motor may have slight differences in hardware characteristics and reduction structure, as well as disturbances during operation, such as momentary wheel slippage, tiny obstacles and other factors that may cause the left and right wheels to rotate at different speeds when going "straight". That’ s why it deviates from the path slightly.

You can upload the program Demo2 to keep observing if you don't see it clearly in the Demo1.

Upload program. (Please toggle the “Upload-Cam” button to “Upload” when uploading the program.) You will see the car move forward and backward at 1s of intervals. In the process of moving forward and backward, the moving route will have a more obvious deviation.



From the above, we know that similar to driving a car, if you close your eyes, it is difficult to keep walking straight for a long distance even if you straighten the steering wheel. Therefore, we have to get the deviation by eye and then correct it by turning the steering wheel manually to walk a straight line. In this case, we must introduce closed-loop control if we want the car to walk a straight line.

Closed-loop control is a basic concept of cybernetics. Refers to a control relationship in which the controlled output returns to the controlled input in a certain way and exerts a control influence on the input.

The advantage of closed-loop control over open-loop control is that it allows the results of the control to be fed back and compared to the desired values and adjusts the control effect based on their errors.

Therefore, by applying closed loop control to our Conqueror Car, we can perform speed corrections via the PID algorithm after integrating the yaw axis data by obtaining the angular velocity filter of the deviation via the MPU6050. Next, please open Demo3 in the current folder.

In this routine, we first ported the "MPU6050.h" library.

Let's first define a related class and class variable.

```
//in MPU6050_getdata.h
class MPU6050_getdata
{
public:
  bool MPU6050_dveInit(void);
  bool MPU6050_calibration(void);
  bool MPU6050_dveGetEulerAngles(float *Yaw);
public:
  //int16_t ax, ay, az, gx, gy, gz;
  int16_t gz;
  //float pith, roll, yaw;
  unsigned long now, lastTime = 0;
  float dt;
  float agz = 0;
  long gzo = 0;
};

extern MPU6050_getdata MPU6050Getdata;
```

Initialize the MPU6050 and ensure that the driver is successful.

```
//in MPU6050_getdata.cpp

bool MPU6050_getdata::MPU6050_dveInit(void)
{
 Wire.begin();
 uint8_t chip_id = 0x00;
 uint8_t cout;
 do
 {
  chip_id = accelgyro.getDeviceID();
  Serial.print("MPU6050_chip_id: ");
  Serial.println(chip_id);
  delay(10);
  cout += 1;
  if (cout > 10)
  {
   return true;
  }
 } while (chip_id == 0X00 || chip_id == 0XFF);
 accelgyro.initialize();
 return false;
}
```

Because of the data drift problem of MPU6050, we had better use mean filtering when sampling, and then obtain the z-axis angular velocity value at the beginning of the car placement.

```
//in MPU6050_getdata.cpp

bool MPU6050_getdata::MPU6050_calibration(void)
{
  unsigned short times = 100;
  for (int i = 0; i < times; i++)
  {
    gz = accelgyro.getRotationZ();
    gzo += gz;
  }
  gzo /= times;
  // gzo = accelgyro.getRotationZ();
  return false;
}
```

Then, put it into the setup() to call.

```
void setup() {
  Serial.begin(9600);
  AppMPU6050getdata.MPU6050_dveInit();
  delay(2000);
  AppMPU6050getdata.MPU6050_calibration();
}
```

From MPU6050, the sensitivity factor is known to be 131 LSB (count) /°/s. After obtaining the original sensor data, we can calculate the angular velocity by dividing the original sensor data by their sensitivity scaling factor:

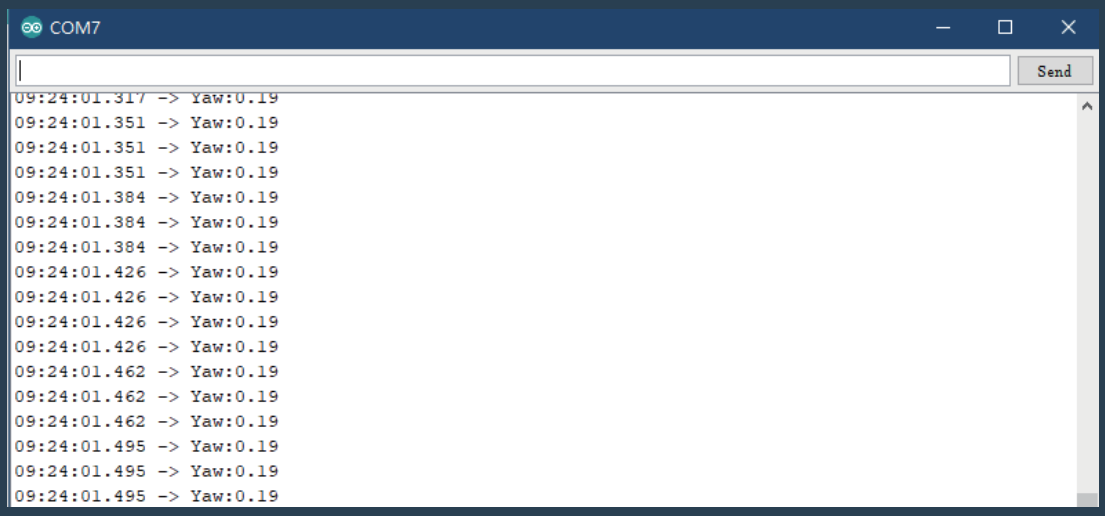
angular velocity along the Z-axis = (gyro z-axis raw data /131) °/s.

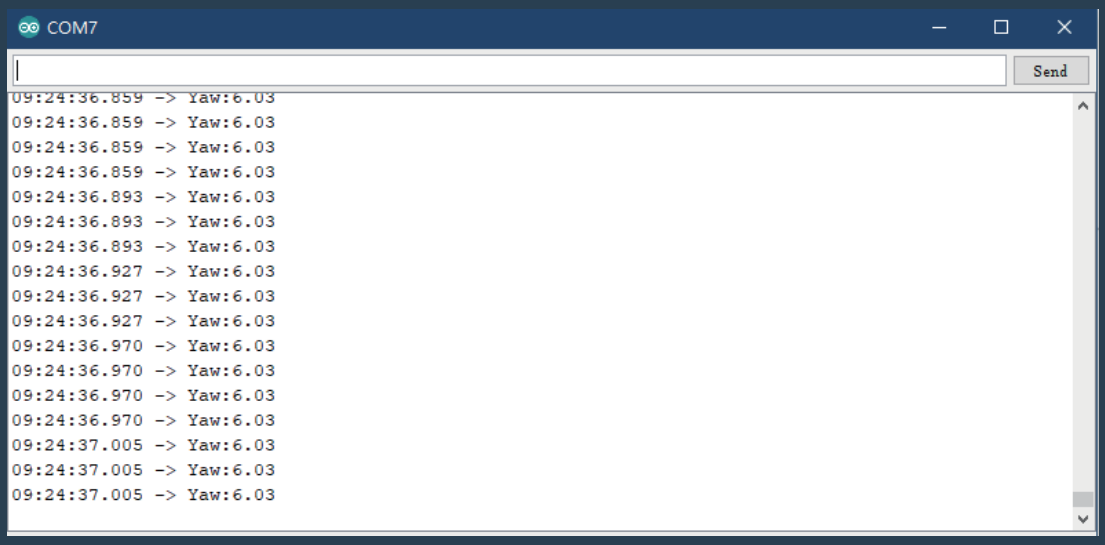
Because the execution time of the program is less than 1S, we need to multiply the final result by time.

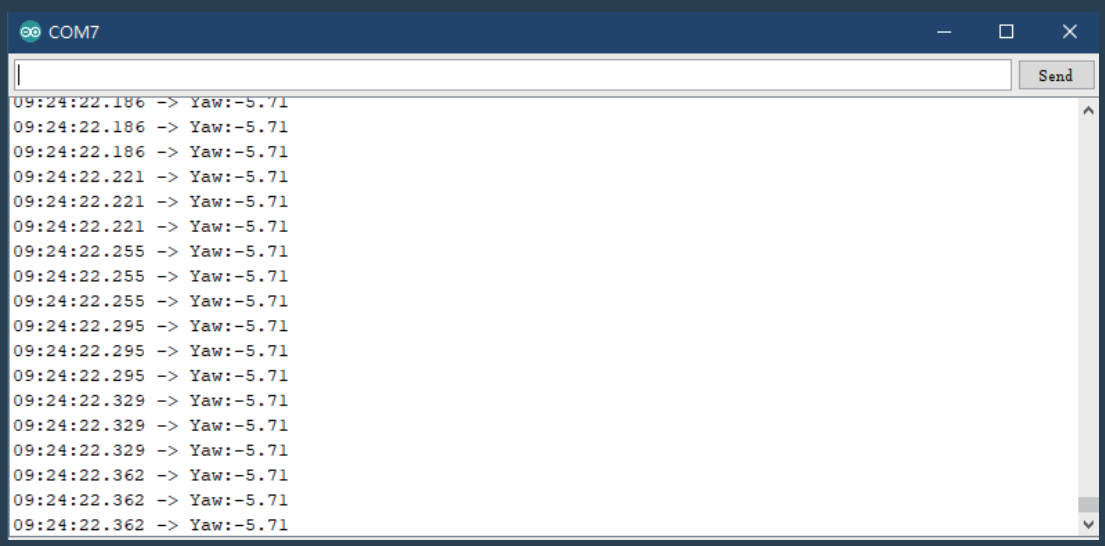
```
//in MPU6050_getdata.cpp

bool MPU6050_getdata::MPU6050_dveGetEulerAngles(float *Yaw)
{
  unsigned long now = millis();
  dt = (now - lastTime) / 1000.0;
  lastTime = now;
  gz = accelgyro.getRotationZ();
  float gyroz = -(gz - gzo) / 131.0 * dt;
  if (fabs(gyroz) < 0.05)
  {
    gyroz = 0.00;
  }
  agz += gyroz;
  *Yaw = agz;
  return false;
}
```

Please place the car steadily before uploading the program. (Please toggle the “Upload-Cam” button to “Upload” when uploading the program) After uploading the program, if you do not move the car, wait two seconds and open the serial port, the yaw angle will fluctuate around 0. If you move the car slightly, the yaw angle will also change accordingly. The straight-line walking function of the car is programmed based on this principle.







After the yaw data is obtained, the closed-loop feedback control can be achieved through PID algorithm to adjust the wheel speed so as to achieve linear walking. Next, open Demo4 in the current folder.

KP scale parameter is the best data we can get through repeated debugging.

UpperLimit is the maximum speed of the car.

```
//in ApplicationFunctionSet_xxx0.cpp

static void ApplicationFunctionSet_ConquerorCarMotionControl
(ConquerorCarMotionControl direction, uint8_t is_speed)
{
Kp = 10;
  UpperLimit = 255;
}
```

The yaw data is acquired at the first startup, and the data is updated every ten millimeters.

```
//in ApplicationFunctionSet_xxx0.cpp

static void ApplicationFunctionSet_ConquerorCarLinear
MotionControl(ConquerorCarMotionControl direction,
uint8_t directionRecord, uint8_t speed, uint8_t Kp,
uint8_t UpperLimit)
{
  .....
  if (en != directionRecord || millis() - is_time > 10)
  {
    AppMotor.DeviceDriverSet_Motor_control
(/*direction_A*/ direction_void, /*speed_A*/ 0,
                                /*direction_B*/ direction_void,
/*speed_B*/ 0, /*controlED*/ control_enable); //Motor control
    AppMPU6050getdata.MPU6050_dveGetEulerAngles(&Yaw);

    is_time = millis();
  }
.....
   }
```

Save the yaw data of the initial position for comparison with the yaw data of subsequent position changes.

```
//in ApplicationFunctionSet_xxx0.cpp

static void ApplicationFunctionSet_ConquerorCarLinearMotion
Control(ConquerorCarMotionControl direction,
uint8_t directionRecord, uint8_t speed, uint8_t Kp,
uint8_t UpperLimit)
{
  .....
  if(en != directionRecord )
  {
    en = directionRecord;
    yaw_So = Yaw;
  }
.....
   }
```

Then, we need to perform P proportional control in the PID algorithm on the acquired input data.

The principle of P control is roughly as follows: Suppose you are running to an end point, when you are far away from the end point, you will run at full speed; when you are close to the end point, you will slow down; and when you are about to reach the end point, you will also about to stop.

And then, remember to limit the corrected speed so that it does not exceed the maximum speed.

```
//in ApplicationFunctionSet_xxx0.cpp

static void ApplicationFunctionSet_ConquerorCarLinearMotionControl
(ConquerorCarMotionControl direction, uint8_t directionRecord,
uint8_t speed, uint8_t Kp, uint8_t UpperLimit)
{
  .....
   int R = (Yaw - yaw_So) * Kp + speed;
  if (R > UpperLimit)
  {
   R = UpperLimit;
  }
  else if (R < 10)
  {
   R = 10;
  }
  int L = (yaw_So - Yaw) * Kp + speed;
  if (L > UpperLimit)
  {
   L = UpperLimit;
  }
  else if (L < 10)
  {
   L = 10;
  }
.....
   }
```

Finally, upload the program. (Pleasetoggle the “Upload-Cam” button to “Upload”when uploading the program.) After you have uploaded the program, please place the car on the ground steadily first and then open the switch. Wait for two seconds, you will find that the car will repeatedly move forward for 3 seconds and then move backward for 3 seconds in a relatively straight line.

ELEGOO

http://www.elegoo.com