

Lesson

3

Following Mode



Introduction:

There are three parts in this lesson. You will learn the principle of following function and program the Tumbler to follow the target to go forward (step back), turn left and turn right.

Tumbler Follow Mode

1.1 Hardware Design

1.2 Software Design

1.3 Upload Validation

Preparations:

one car(with a battery)

one USB cable

1.1 Hardware Design

In this course, we will use the ultrasonic module. The principle of ultrasonic distance measurement is that the transmitter sends ultrasonic wave to a certain direction, and starts timing of the launching time at the same time. The ultrasonic wave travels in the air, and returns immediately when it encounters an obstacle on the way, and stops timing immediately when the ultrasonic receiver receives the reflected wave. According to the recorded time and the known propagation speed of ultrasound in the air is 340m / s, the distance between the ultrasonic module and the obstacle can be obtained. (As shown in figure 1.1.1~1.1.2).

Finally, let's see which pin of Nano is connected to the ultrasonic module. (As shown in figure 1.1.3)



Figure 1.1.1 The real object of the ultrasonic module

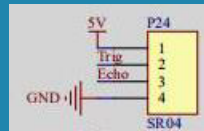


Figure 1.1.2 The schematic of the ultrasonic module

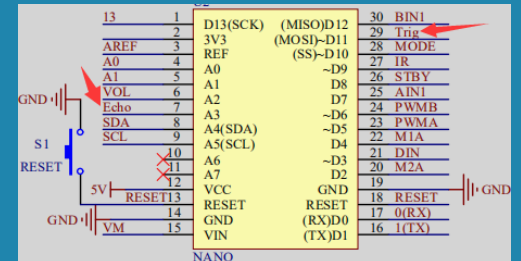


Figure 1.1.3 Pins connected to ultrasonic module on Nano

Since the ultrasonic module can only detect the obstacles in front, there is no way to detect the obstacles when they move from the front to the sides, so we also add photoelectric sensors on both sides, so that the car can judge the obstacles on both sides and turn.



Figure 1.1.4 The real object of the photoelectric sensors

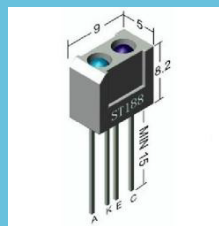


Figure 1.1.5 3D structure diagram

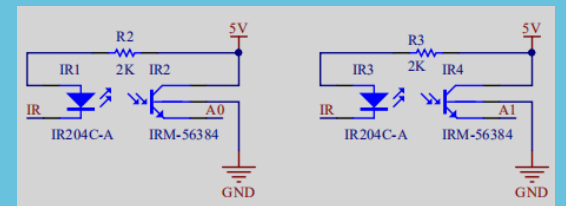


Figure 1.1.6 The schematic diagram of tracking module

The ITR20001/T consist of an infrared emitting diode and an NPN silicon phototransistor. That is to say It is equipped with a light source and a light receiving device. The light emitted by the light source is received by the photosensitive element through the reflection of the object to be tested, and then the required information is obtained through the processing of the relevant circuit. It can be used to detect the change of light, shade and color of the ground, and also to detect whether there are close objects.

Finally, let's see which pin of Nano is connected to the two light blockers and then we could start to write programs. (As shown in figure 1.1.7)

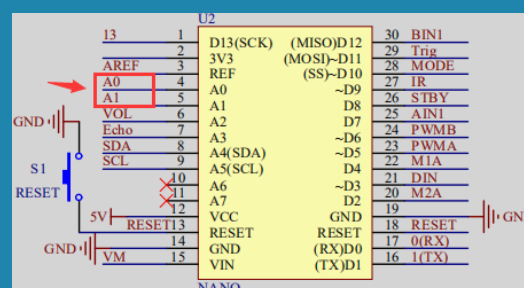


Figure 1.1.7 The schematic of Nano

1.2 Software Design

Please open [ELEGOO Tumbler Self-Balancing Car Tutorial -> Lesson 3 Follow Mode -> Follow Mode 1-> Balanced_Car->Balanced_Car.ino](#) in the current folder and [ELEGOO Tumbler Self-Balancing Car Tutorial -> Lesson 3 Follow Mode -> Follow Mode 2->Balanced_Car->Balanced_Car.ino](#) in the current folder.

Now, let's take a look at the class declaration of ultrasonic and photoelectric sensors.

```
//in Follow1.h

class Ultrasonic
{
public:
    void Pin_init();
    void Get_Distance();
    void Check();
    static void Distance_Measure();

public:
    static char measure_flag;
    static unsigned long measure_prev_time;
    static unsigned long get_distance_prev_time;
    static double distance_value;

    unsigned long ir_send_time;
private:
    #define ECHO_PIN 17
    #define TRIG_PIN 11
    #define Distance_MIN 3
    #define Distance_MAX 35
    #define DISTANCE_JUDGEMENT (distance_value > Distance_MIN && distance_value < Distance_MAX)
};

.....
class IR
{
public:
    void Pin_init();
    void Check();
    void Send();
    void Filter();
    static void Left_Receive();
    static void Right_Receive();
public:
    static unsigned char left_receive_flag;
    static unsigned int left_count;
    static unsigned char right_receive_flag;
    static unsigned int right_count;
    unsigned long left_count_time = 0;
    static int left_is_obstacle;
    static int right_is_obstacle;
private:
    #define RECV_PIN 9
    #define IR_SEND_PIN 9
    #define LEFT_RECEIVE_PIN A0
    #define RIGHT_RECEIVE_PIN A1
    #define If_IR_TRIGGERED (IR.left_is_obstacle || IR.right_is_obstacle)
};
```

First, we need to initialize the ultrasonic module and photoelectric sensor.

```
//in Follow1.cpp

void Ultrasonic::Pin_init()
{
    pinMode(ECHO_PIN, INPUT);
    pinMode(TRIG_PIN, OUTPUT);
}
```

Because the photoelectric module needs external interrupt, the corresponding pin interrupt should also be initialized.

```
//in PinChangeInterrupt.cpp

void IR::Pin_init()
{
    pinMode(LEFT_RECEIVE_PIN, INPUT_PULLUP);
    pinMode(RIGHT_RECEIVE_PIN, INPUT_PULLUP);
    pinMode(IR_SEND_PIN, OUTPUT);
    attachPinChangeInterrupt(LEFT_RECEIVE_PIN, Left_Receive, FALLING);
    attachPinChangeInterrupt(RIGHT_RECEIVE_PIN, Right_Receive, FALLING);
}
```

The infrared light emitted by the light source is received by the photosensitive element through the reflection of the object to be tested, so the interruption will be triggered when encountering obstacles. Therefore, we need to drive the emission pin of the photoelectric sensor to send 39 square waves continuously to emit infrared light.

```
//in Follow1.cpp

void IR::Send()
{
    static unsigned long ir_send_time;

    if (millis() - ir_send_time > 15)
    {
        for (int i = 0; i < 39; i++)
        {
            digitalWrite(IR_SEND_PIN, LOW);
            delayMicroseconds(9);
            digitalWrite(IR_SEND_PIN, HIGH);
            delayMicroseconds(9);
        }
        ir_send_time=millis();
    }
}
```

Every time an obstacle is encountered, the interruption function will be triggered to change the moving sign of the car, thus changing the moving state of the car. Let's take a look at the interrupt functions of the left and right photoelectric sensors.

```
//in Follow1.cpp

int IR::left_is_obstacle;
int IR::right_is_obstacle;
unsigned int IR::left_count;
unsigned int IR::right_count;
void IR::Left_Receive()
{
    left_is_obstacle=1;
}
void IR::Right_Receive()
{
    right_is_obstacle=2;
}
```

Then we need to drive the ultrasonic module to get the distance of the front obstacles.

Let's take a look at the driving sequence diagram of the ultrasonic module. (As shown in figure 1.1.7)

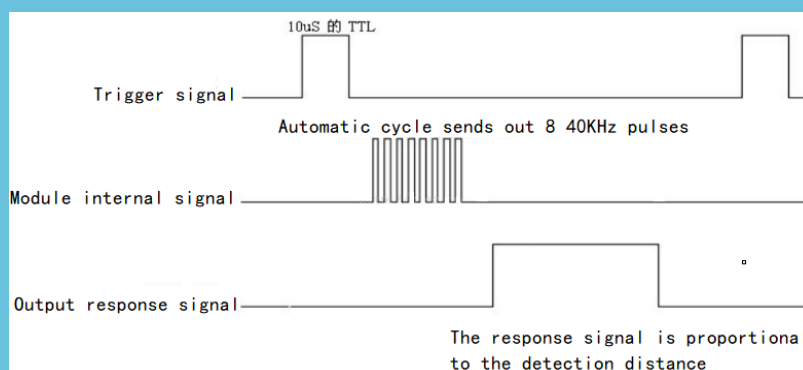


Figure 1.1.7 The driving sequence diagram of the ultrasonic module

- (1) Adopt IO trigger ranging to give high-level signal of at least 10us;
- (2) The module automatically sends 8 square waves of 40KHz to detect whether there is signal return;
- (3) When a signal is returned, a high level is output through IO, the duration time of high level is the time from transmission to return of ultrasonic.
- (4) Calculate the test distance: $\text{test distance} = (\text{high level time} * \text{sound speed (340m / s)}) / 2$;

Let's have a look at the program analysis.

```
//in PinChangeInterrupt.cpp
void Ultrasonic::Get_Distance()
{
  if (millis() - get_distance_prev_time > 50)
  {
    delayMicroseconds(1);
    get_distance_prev_time = millis();
    measure_flag = 0;
    attachPinChangeInterrupt(ECHO_PIN, Distance_Measure, RISING);
    digitalWrite(TRIG_PIN, LOW);
    delayMicroseconds(2);
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG_PIN, LOW);
  }
}
```

First of all, we need to send a high-level square wave with a time of 10us as the trigger signal
(As shown in figure 1.1.8)

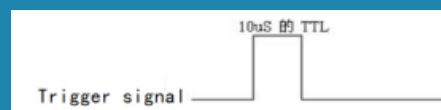


Figure 1.1.8 a high-level square wave with a time of 10us as the trigger signal

```
digitalWrite(TRIG_PIN, LOW);
delayMicroseconds(2);
digitalWrite(TRIG_PIN, HIGH);
delayMicroseconds(10);
digitalWrite(TRIG_PIN, LOW);
```

Then read the status of echo pin. If the status of echo pin changes from low level to high level
(as shown in figure 1.1.9), it means that the ultrasonic starts to emit. When there is a rising edge, the interrupt function is triggered.



Figure 1.1.9 The rising edge

```
//in PinChangeInterrupt.cpp
void Ultrasonic::Get_Distance()
{
  if (millis() - get_distance_prev_time > 50)
  {
    .....
    attachPinChangeInterrupt(ECHO_PIN, Distance_Measure, RISING);
    .....
  }
}
```

In the interrupt function, we start to record the time of entering the interrupt (time of ultrasonic emission), and modify the interrupt condition to trigger the falling edge.

```
//in PinChangeInterrupt.cpp
void Ultrasonic::Distance_Measure()
{
  if (measure_flag == 0)
  {
    measure_prev_time = micros();
    attachPinChangeInterrupt(ECHO_PIN, Distance_Measure, FALLING);
    measure_flag = 1;
  }
  .....}
}
```

When the ultrasonic encounters an obstacle and returns, the echo pin status changes from low level to high level, that is to say, when there is a falling edge, the interrupt function will be triggered again. (As shown in figure 1.1.10)

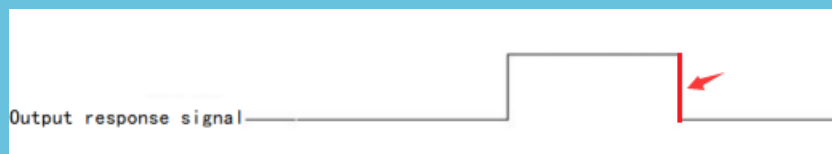


Figure 1.1.10 the falling edge

When the interrupt function is triggered again, we can record the current time and get the distance from the obstacle according to the formula. (As shown in figure 1.1.11)

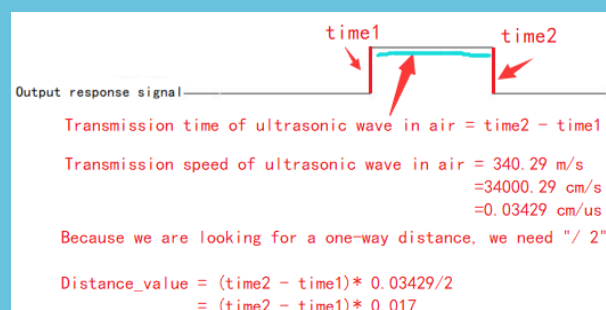


Figure 1.1.11 Formula principle

```
//in PinChangeInterrupt.cpp
void Ultrasonic::Distance_Measure()
{
  .....
  else if (measure_flag == 1)
  {
    distance_value = (micros() - measure_prev_time) * 0.017; //340.29 m/s / 2 -> (340.29*100 cm)
    // (1000*1000 us) / 2 = 0.0170145
    measure_flag = 2; //Serial.println(distance_value);
  }
}
```


After we have driven the ultrasonic module and two photoelectric sensors, we can realize the following mode according to the detected value. First of all, let's write a program to implement the follow-up function. Frist, the idea is as follows: (As shown in figure 1.1.12)

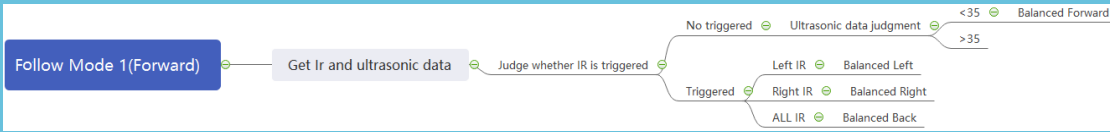


Figure 1.2.12 Programming ideas

```

//in Follow1.cpp
void Function::Follow_Mode1()
{
    IR.Send();
    Ultrasonic.Get_Distance();
    if (millis() - follow_prev_time >= 100)
    {
        If_IR_TRIGGERED ? IR.Check():Ultrasonic.Check();
        follow_prev_time = millis();
    }
}
  
```

Similarly, following mode 2 are shown in figure 1.1.13.

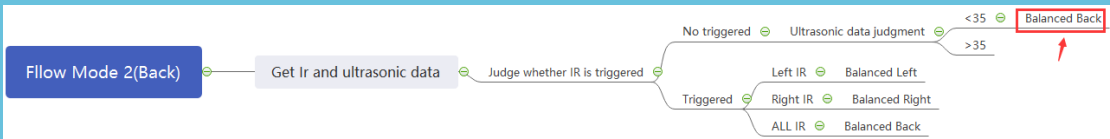


Figure 1.2.13 Programming ideas

1.3 Upload Validation

After uploading the program following mode 1, the Tumbler will follow the target to go forward, turn left and turn right.

After uploading the program following mode 2, the Tumbler will follow the target to go step back, turn left and turn right.

After we have driven the ultrasonic module and two photoelectric sensors, we can realize the following mode according to the detected value. First of all, let's write a program to implement the follow-up function. First, the idea is as follows:(As shown in figure 1.1.12)

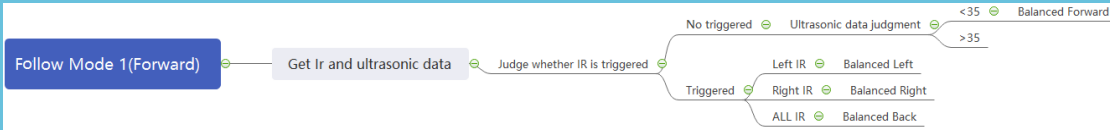


Figure 1.2.12 Programming ideas

```

//in Follow1.cpp
void Function::Follow_Mode1()
{
    IR.Send();
    Ultrasonic.Get_Distance();
    if (millis() - follow_prev_time >= 100)
    {
        If_IR_TRIGGERED ? IR.Check():Ultrasonic.Check();
        follow_prev_time = millis();
    }
}
  
```

Similarly, following mode 2 are shown in figure 1.1.13.

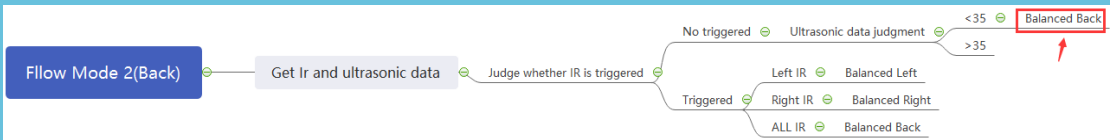


Figure 1.2.13 Programming ideas

1.3 Upload Validation

After uploading the program following mode 1, the Tumbler will follow the target to go forward, turn left and turn right.

After uploading the program following mode 2, the Tumbler will follow the target to go step back, turn left and turn right.