Infrared Remote Control Experiment

Introduce to Infrared receiver tube

Infrared receiver tube, it is a kind of sensor that can identify infrared.Integrated reception and modulation of 38kHZ infrared sensor.In order to avoid the interference of other infrared signals in the wireless transmission process, the infrared remote control usually modulates the signal on a specific carrier frequency, and then it is sent out by the infrared emitting diode.When the infrared receiver needs to filter out other clutter, it receives a specific frequency signal and returns it to binary pulse code, that is, demodulation.

Working Principle

The built-in receiving tube converts the optical signal sent by the infrared transmitter tube into weak current signal, which is amplified by internal IC, and then restored to the original code sent by the infrared remote control through automatic gain control, bandpass filtering, demodulation, waveform shaping, and the decoding circuit input to the electrical appliance through the output pin of the receiving head

Experiment Purpose

The buttons of the remote control are coded through Ardunio

Arduino UNO main control board communicates with infrared receiver. If the "<" button of the remote control is pressed, the fan turn left; if the ">" button is pressed, the fan turn right "ok" button is pressed,

fan start run, when "ok" pressed again, fan stop.

Introduction of NEC

Characteristics

8 address bits, 8 command bits

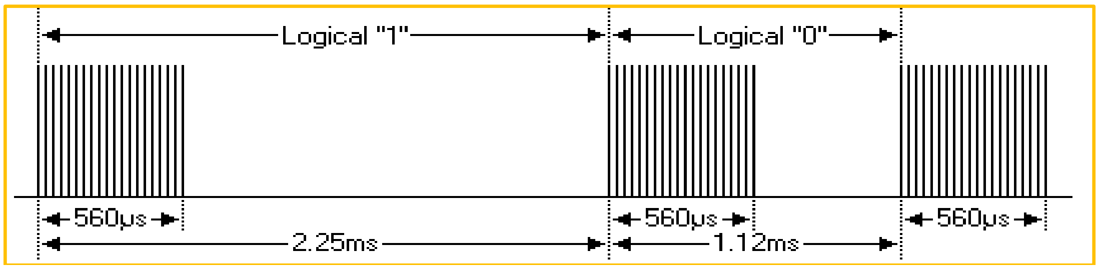
Address bits and command bits are transmitted twice in order to ensure reliability

Pulse-position modulation

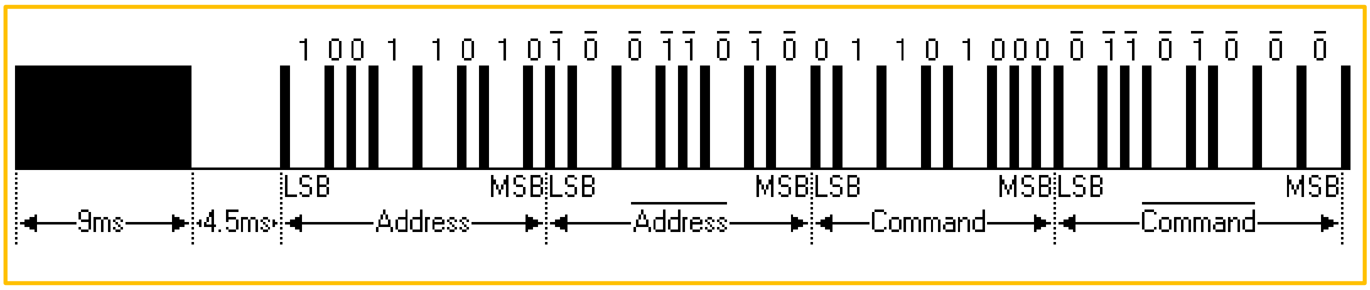
Carrier frequency 38kHz

Every bit lasts 1.125ms or 2.25ms

The definitions of logic 0 and 1 are as below

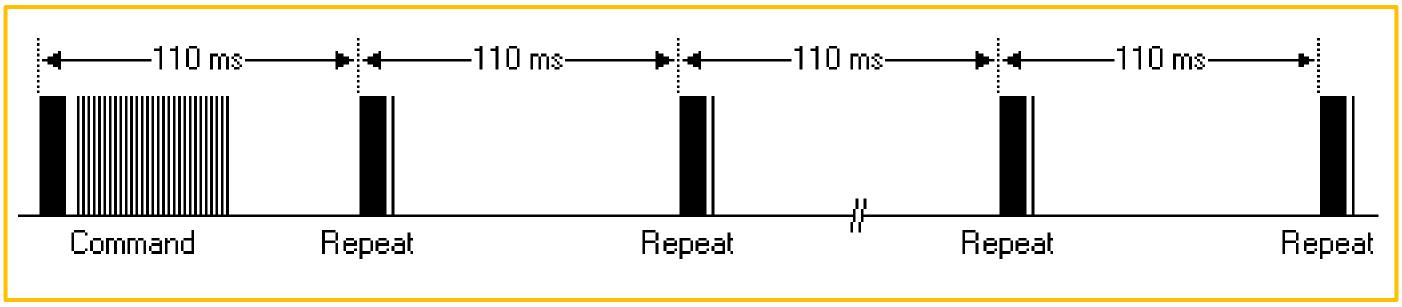


Transmitted pulse which the pressed button released immediately



The picture above shows a typical pulse sequence of the NEC protocol. Notice: The protocol of LSB (least significant) is firstly transmitted. In the above, pulse transmission address is 0x16 and the command is 0x59. A message starts from a 9ms high level, then followed by a 4.5ms low level, and by the address code and command code. The address and command are transmitted twice. All bits flip in the second transmission, this can be used in the confirmation of the received message. The total transmission time is constant, because every bit repeats the flip length. If you are not interested, you can ignore this reliable inversion and expand the address and command every 16 bit as well !

Transmission pulse for a period of time when the button is pressed



Even if you press the button on the remote control again, the command is only sent once. When the button is pressed, the first 110ms pulse is the same as above, and then the same code is sent every 110ms. The next repetition code consists of 9ms high-level pulse, 2.25ms low-level pulse and 560μs high-level pulse.

Note: When the integrated head receives pulses, the head needs to decode, amplify and shape the signal. So we should notice that the output is high when the infrared signal is not present, otherwise the output is low, so the output signal level is reversed in the transmitter. We can see the pulse of the receiver through the oscilloscope and understand the program through the waveform.

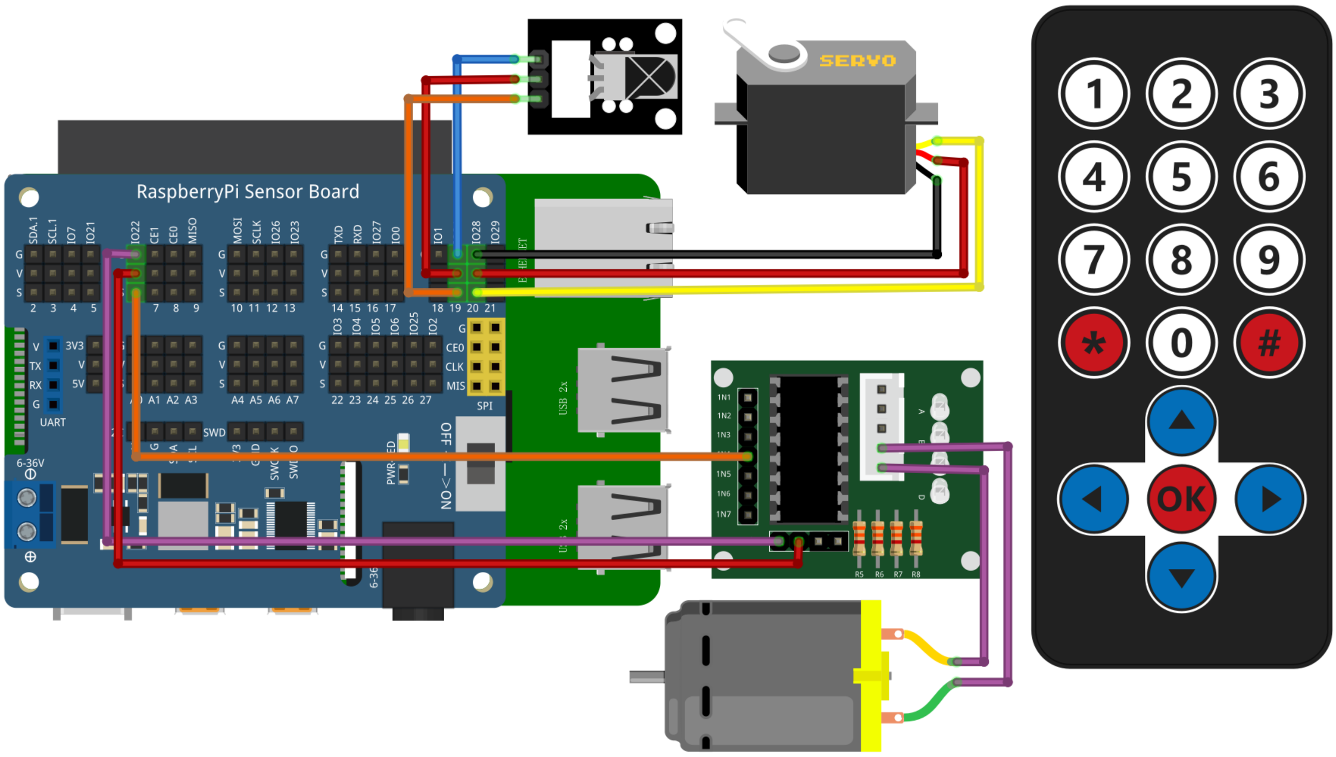
组件清单

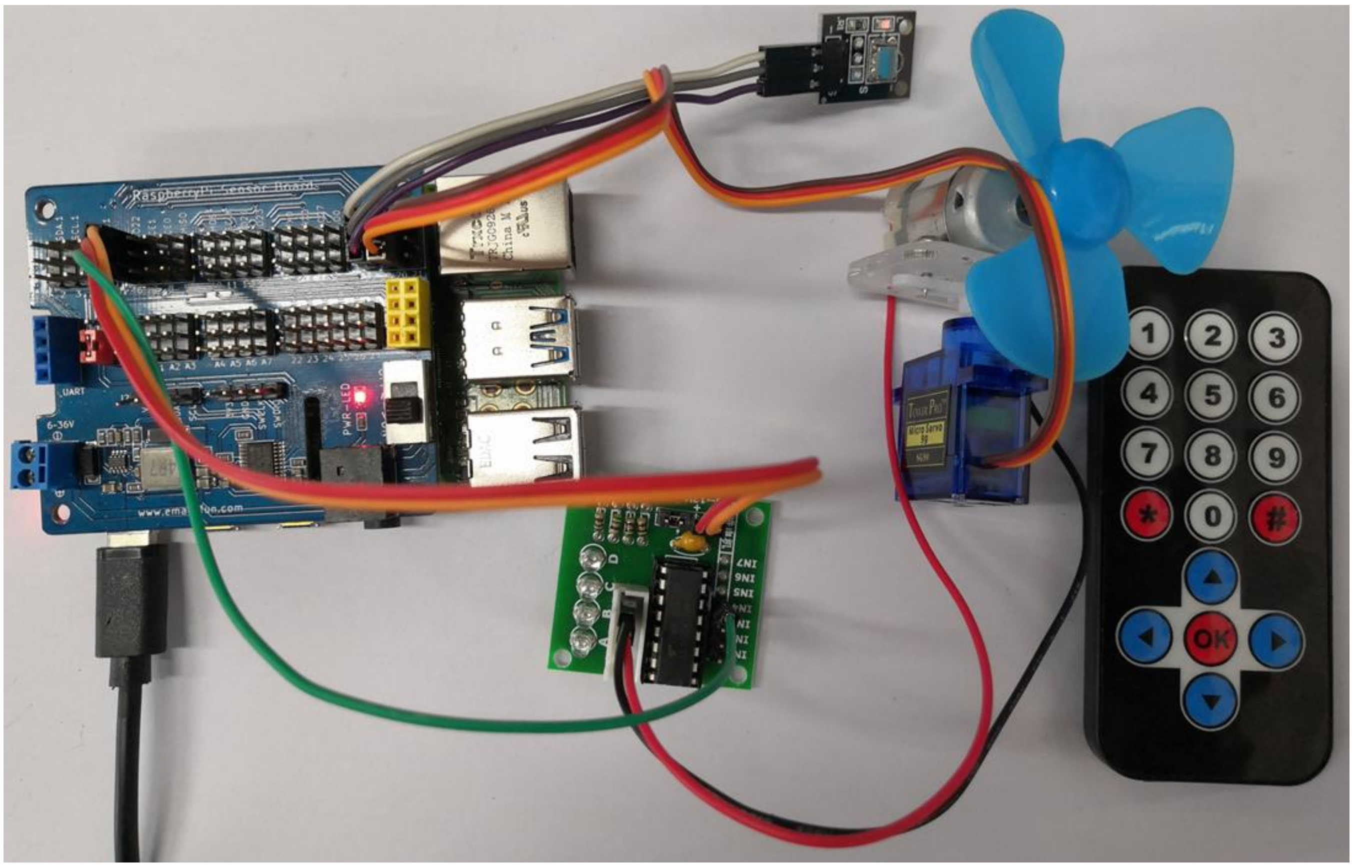
- RaspberryPi mainboard
- Breadboard
- USB data cable
- Infrared remote control * 1
- Integrated infrared receiver module * 1
- DC motor*1
- Fan*1
- Servo*1
- Motor drive board*1
- Motor bracket kit*1
- Battery*1
- Several jumpers

接线

RaspberryPi 主板	Steering gear module
5V	VCC
GND	GNG
IO28(wiringPi)/20(BCM)	S
RaspberryPi 主板	Infrared remote control
5V	+
GND	-
IO24(wiringPi)/19(BCM)	
RaspberryPi 主板	Motor drive board
5V	5V(+)
GND	GND(-)
IO22(wiringPi)/6(BCM)	IN4
Motor drive board	DC motor
VCC	+
OUT4	-

实物接线





C++ program

```
#include "IR_REC.h"

int main()
{
    int flag;
    if(wiringPiSetup() == -1){
       printf("setup wiringPi failed !");
       return 1;
    }
    while(1){
        key = Change_Map(GetKey());
        if (key != ERROR) {
            printf("Change_Map %s \n",keymap[key].keyname.c_str());
            switch (key) {// Determine which button is pressed and execute the corresponding
program
                case IR_KEYCODE_OK:
                        printf("IR_KEYCODE_OK key\n");
                        flag = !flag;
                        digitalWrite(MotorPin, flag);// Control the motor
                        break;
                case IR_KEYCODE_LEFT:
                        pwm_fun(0);
                        delay(500);// Control the steering gear to turn to 0 degree
                        printf("IR_KEYCODE_OK left\n");
                        break;
                case IR_KEYCODE_RIGHT:
                        pwm_fun(180);
                        delay(500);//Control the steering gear to turn 180 degrees
                        printf("IR_KEYCODE_OK right\n");
                        break;
            }
        }
    }
}
```

Python program

```
#!/usr/bin/python
# -*- coding:utf-8 -*-
import RPi.GPIO as GPIO
import time
```

```
ERROR = 0xFE
PIN = 19
flag = 0
motor = 6
servopin = 20
    #Define infrared receiver pin
keymap = {
  0x45:"1",
  0x46:"2",
  0x47:"3",
  0x44:"4",
  0x40:"5",
  0x43:"6",
  0x07:"7",
  0x15:"8",
  0x09:"9",
  0x19:"0",
  0x16:"*",
  0x0D:"#",
  0x18:"up",
  0x52:"down",
  0x1C:"ok",
  0x08:"left",
  0x5A:"right",
  0xfe:0xfe
}

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(PIN, GPIO.IN, GPIO.PUD_UP)   # Set the infrared receiving pin to pull-up mode
GPIO.setup(motor,GPIO.OUT)
GPIO.setup(servopin,GPIO.OUT)

def getKey():
    byte = [0, 0, 0, 0]
    if IRStart() == False:      # Judge the infrared guidance pulse
        time.sleep(0.11)       # One message frame lasts 108 ms.
        return ERROR
    else:
        for i in range(0, 4):
                byte[i] = getByte()  # Receive 32-bit infrared data (address, address
inversion, data, data inversion)
```

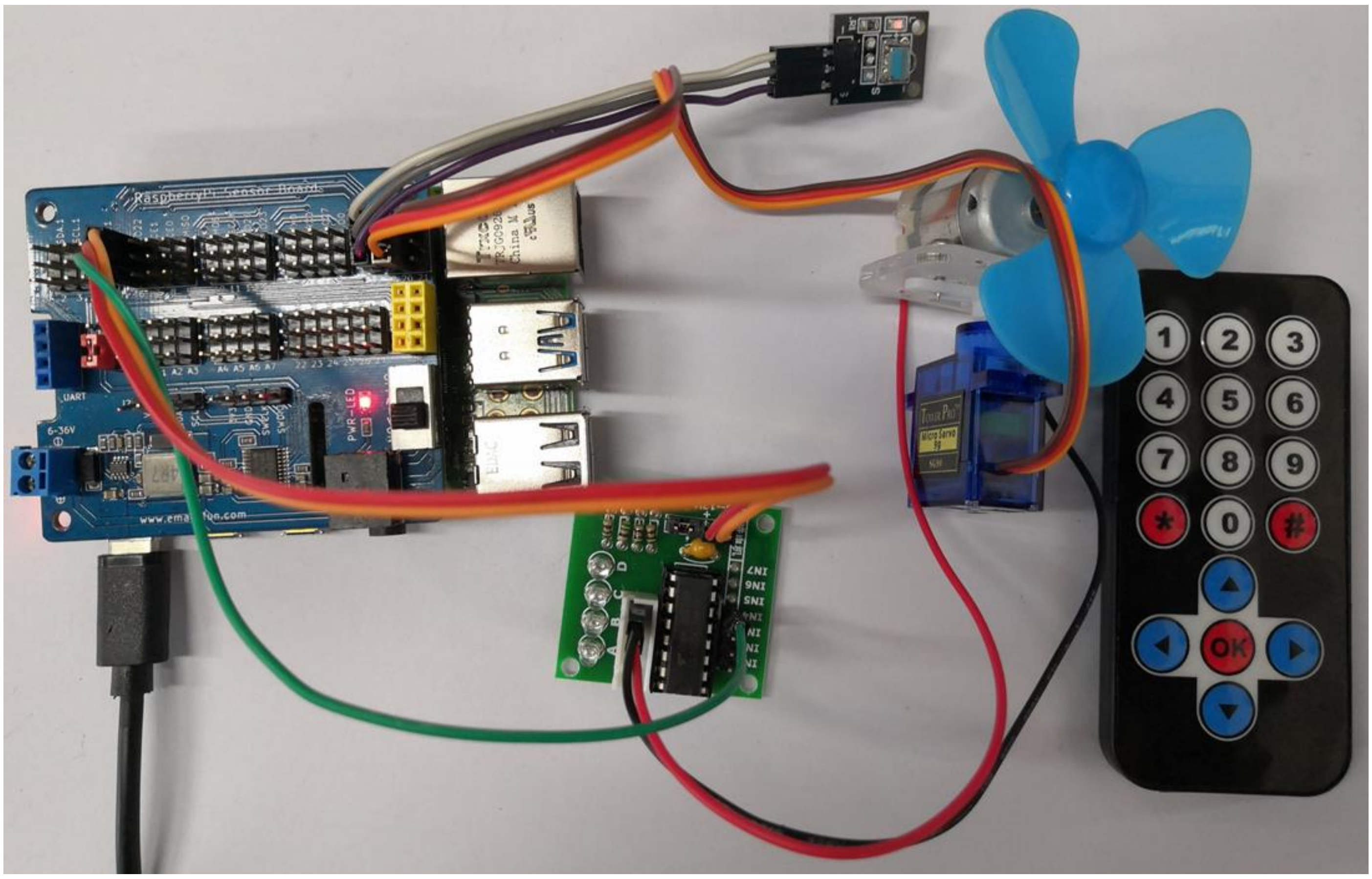
```
        if byte[0] + byte[1] == 0xff and byte[2] + byte[3] == 0xff:  # Verify that the received 
data is correct
#print("right")
            return byte[2]
        else:
        #print("error")
            return ERROR
    #return byte[2]
def IRStart():     # Judge the infrared guidance pulse
    timeFallingEdge = [0, 0]
    timeRisingEdge = 0
    timeSpan = [0, 0]
    # Read pulse time by pulse rising and falling edge
    GPIO.wait_for_edge(PIN, GPIO.FALLING)
    timeFallingEdge[0] = time.time()
    GPIO.wait_for_edge(PIN, GPIO.RISING)
    timeRisingEdge = time.time()
    GPIO.wait_for_edge(PIN, GPIO.FALLING)
    timeFallingEdge[1] = time.time()

    timeSpan[0] = timeRisingEdge - timeFallingEdge[0]  # First pulse time
    timeSpan[1] = timeFallingEdge[1] - timeRisingEdge  # Second pulse time
    #print(timeSpan[0],timeSpan[1])
    if timeSpan[0] > 0.0085 and timeSpan[0] < 0.0095 and timeSpan[1] > 0.004 and timeSpan[1] 
< 0.005:
    #print("1")
        return True
    else:
    #print("0")
        return False
def getByte():   # Receive 32-bit infrared data (address, address inversion, data, data 
inversion)
    byte = 0
    timeRisingEdge = 0
    timeFallingEdge = 0
    timeSpan = 0
    for i in range(0, 8):
        # Read pulse time by pulse rising and falling edge
        GPIO.wait_for_edge(PIN, GPIO.RISING)
        timeRisingEdge = time.time()
        GPIO.wait_for_edge(PIN, GPIO.FALLING)
        timeFallingEdge = time.time()
        timeSpan = timeFallingEdge - timeRisingEdge   # Read pulse time
        if timeSpan > 0.0016 and timeSpan < 0.0018:   # Determine whether the pulse is 
representative 1
            byte |= 1 << i
    return byte

def change_map(key_val):
    for index in keymap.keys():
        if index == key_val :
            return keymap[index]

def pwm_change(temp):
    GPIO.output(servopin ,True)
    time.sleep(0.0005+float(temp)*0.0005/45)
    GPIO.output(servopin , False)
    time.sleep(0.02-(0.0005+float(temp)*0.0005/45))

print('IRM Test Start ...')
try:
    while True:
        # Read infrared pulse
        key = change_map(getKey())
        if(key != ERROR):
            print("Get the key:" + key)
            if(key == "ok"):
                print("IR_KEYCODE_OK key")
                flag = 1 - flag
                GPIO.output(motor,flag)

            if(key == "left"):
                print("IR_KEYCODE_left key")
                pwm_change(0)
                time.sleep(0.5)

            if(key == "right"):
                print("IR_KEYCODE_right key")
                pwm_change(180)
                time.sleep(0.5)

except KeyboardInterrupt:
    GPIO.cleanup()
```

Java program

```
import com.pi4j.wiringpi.Gpio;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;


public class IR_NEC {
       static int PIN = 24, motor = 22, PWMPIN = 28;
       static int ERROR = 0xfe, key, flag = 0;
       static long timeRisingEdge, timeFallingEdge, timeRising, timeFalling_0,
timeFalling_1;
       static long timeSpan_val = 0;
       static long[] time_span = new long[2];
       static Map<Integer, String> map = new HashMap<>();


    static {
       if (Gpio.wiringPiSetup() == -1) {
          System.out.println(" ==>> GPIO SETUP FAILED");
       }
       Gpio.pinMode(motor, Gpio.OUTPUT);
       Gpio.pinMode(PWMPIN, Gpio.OUTPUT);

       // store data in map collection
       map.put(0x45, "1" );
       map.put(0x46, "2" );
       map.put(0x47, "3" );
       map.put(0x44, "4" );
       map.put(0x40, "5" );
       map.put(0x43, "6" );
       map.put(0x07, "7" );
       map.put(0x15, "8" );
       map.put(0x09, "9" );
       map.put(0x19, "0" );
       map.put(0x16, "*" );
       map.put(0x0D, "#" );
       map.put(0x18, "up");
       map.put(0x52, "down");
```

```
        map.put(0x1C, "ok");
        map.put(0x08, "left");
        map.put(0x5A, "right");
        map.put(0xfe, "error");

        Gpio.pinMode(PIN, Gpio.INPUT);
    }

    public static boolean  IRStart() {
        while(!(Gpio.digitalRead(PIN) == 0));
        timeFalling_0 = gettimeofday();
        while(!(Gpio.digitalRead(PIN) == 1));
        timeRising = gettimeofday();
        while(!(Gpio.digitalRead(PIN) == 0));
        timeFalling_1 = gettimeofday();
        time_span[0] = timeRising - timeFalling_0;
        time_span[1] = timeFalling_1 - timeRising;

//      System.out.println("start_time " + time_span[0] + "," +time_span[1]);

        if (time_span[0] > 8500 && time_span[0] < 9500 && time_span[1] >= 4000 && time_span[1]
<= 5000)
        {
//          System.out.println("start singe************");
            return true;
        }
        else   {
            return false;
        }
    }

    public static long gettimeofday() {
//      return System.currentTimeMillis() ;// +System.nanoTime() / 1000;
        return System.nanoTime() / 1000;

    }

    public static int GetByte() {
        int byte_val = 0;
        for (int i = 0; i < 8; i++) {
            while(!(Gpio.digitalRead(PIN) == 1));
            timeRisingEdge = gettimeofday();
            while(!(Gpio.digitalRead(PIN) == 0));
            timeFallingEdge = gettimeofday();
            timeSpan_val = timeFallingEdge - timeRisingEdge;
//          System.out.print("start byte  ");
//          System.out.println(timeSpan_val);

            if (timeSpan_val > 1500 && timeSpan_val < 1800)
                byte_val |= 1 << i;

        }
//  System.out.printf("byte_val: %x  \n", byte_val);

        return byte_val;
    }

    public static int GetKey() {
        int[] byte_val = new int[4];
        if (IRStart() == false) {
            Gpio.delay(108);
            return ERROR;
        } else {
            for (int i = 0; i < 4; i++) {
                byte_val[i] = GetByte();
//              System.out.printf("byte_val[%d]: %x \n",i, byte_val[i]);

            }
            if ((byte_val[0] + byte_val[1] == 0xff) && (byte_val[2] + byte_val[3] == 0xff))
{
                return byte_val[2];
            } else {
                    return ERROR;
              }
          }
     }

    public static String change_map(int data) {
        Set<Integer> keys = map.keySet();
        for(Integer key:keys){
            //System.out.println("key值: "+key+" value值: "+map.get(key));
            if(data == key)
                return map.get(key);
        }
        return "ERROR";
    }

    public static void pwm(int val){
        Gpio.digitalWrite(PWMPIN, Gpio.HIGH);
        Gpio.delayMicroseconds(500 + val*500 / 45);
        Gpio.digitalWrite(PWMPIN,Gpio.LOW);
        Gpio.delayMicroseconds((20000 - (500 + val*500 / 45)));
    }

    public static void main(String args[]) {
        String rec_val;
        int get_data;
        IR_NEC ir_nec = new IR_NEC();
        for ( ; ;) {
            get_data = ir_nec.GetKey();
            rec_val = ir_nec.change_map(get_data);
            if (rec_val != "error") {
                System.out.println("key: " + rec_val);
                switch (get_data) {// Determine which button is pressed and execute the
corresponding program
                    case 0x1C:
                            System.out.println("IR_KEYCODE_OK key");
                            flag = ~flag;
                            Gpio.digitalWrite(motor, flag);// Control the motor
                            break;
                    case 0x08:
                            pwm(0);
                            Gpio.delay(500);// Control the steering gear to turn to 0 degree
                            System.out.println("IR_KEYCODE_OK left");
                            break;
                    case 0x5A:
                            pwm(180);
                            Gpio.delay(500);//Control the steering gear to turn 180 degrees
                            System.out.println("IR_KEYCODE_OK right");
                            break;
                }
            }
        }
    }
}
```

Experimental results



You can control the rotation of the steering gear and the rotation of the motor through the RaspberryPi board to control the infrared remote control