基于 Arduino 的智能风扇设计

徐红月　　17307130262

摘要：本实验实现了温度自动控制风扇挡位，手势识别控制方向，也可通过红外遥控或手机蓝牙控制风扇挡位方向。将所有部件集成到小车上，可以通过蓝牙远程控制小车行动。

一、引言：

Arduino 是一款基于易使用的电子器件和软件的开源平台，具有简单易上手，价格低，功能丰富，应用广泛等特点。

本实验用舵机控制风扇角度，用 DHT11 温湿度传感器实现温度自动控制风扇挡位，用 PAJ7620 手势识别传感器实现手势识别控制方向，也可通过红外遥控或手机蓝牙控制风扇挡位方向。将所有部件集成到小车上，可以通过蓝牙远程控制小车行动。

二、实验原理：

1）L298N 电机驱动模块原理：

L298N 可同时驱动 2 个电动机， OUT1, OUT2 和 OUT3, OUT4 之间可分别接电机， 5, 7, 10, 12 脚接输入控制电平，控制电机的正反转。 E_{nA}，E_{nB} 接控制使能端，控制电机的停转。

E_{nA}	In1	In2	运转状态
0	×	×	停止
1	1	0	正转
1	0	1	反转
1	1	1	刹停
1	0	0	停止

表 1 L298N 功能逻辑

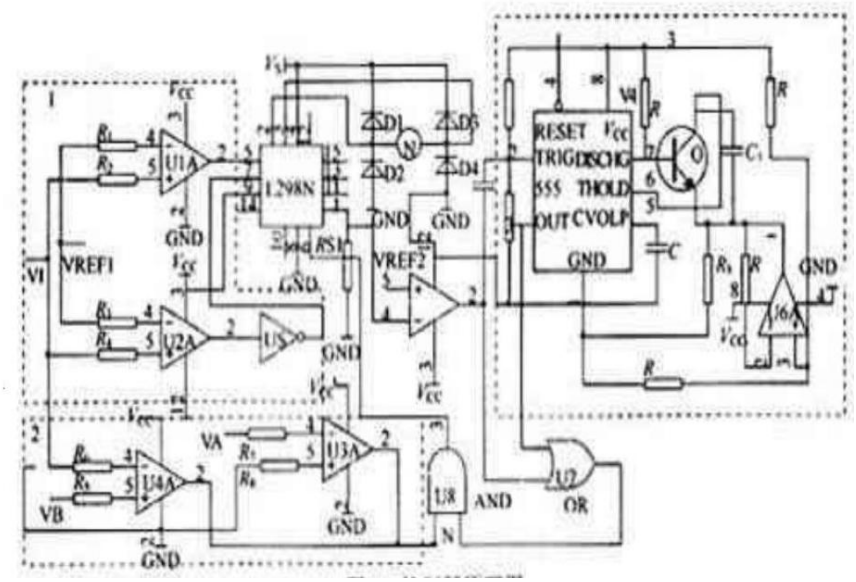


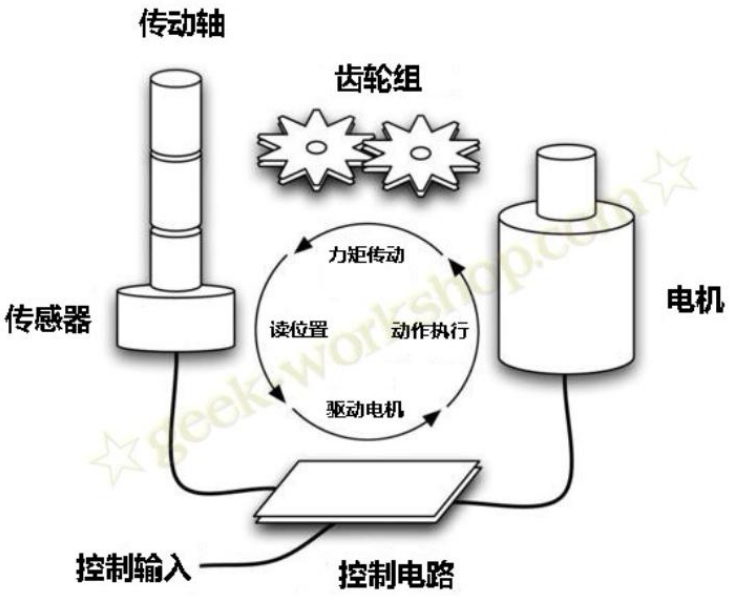
图 1 L298N 内部结构

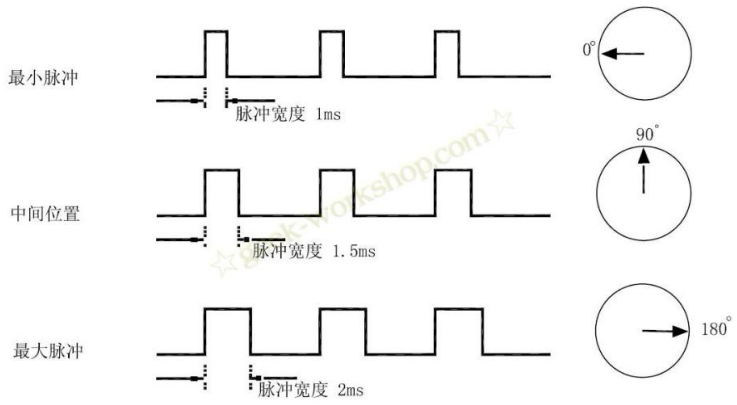
虚线框图 1 控制电机正反转，框图 2 控制电机停转，框图 3 为长延时电路，吸收电机启

动过流电压波形，从而使电机正常启动。

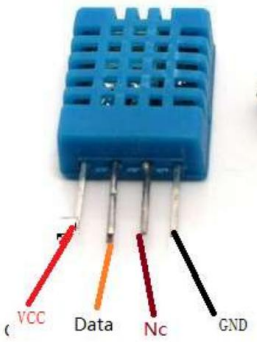
2）舵机原理：

舵机输入脉冲信号，一个周期 20ms，其中脉冲宽度 1.5ms 时舵机转到中间位置，脉冲宽度小于 1.5ms，通常脉冲宽度在 1~2ms 间变化，不同脉冲宽度对应舵机不同角度。



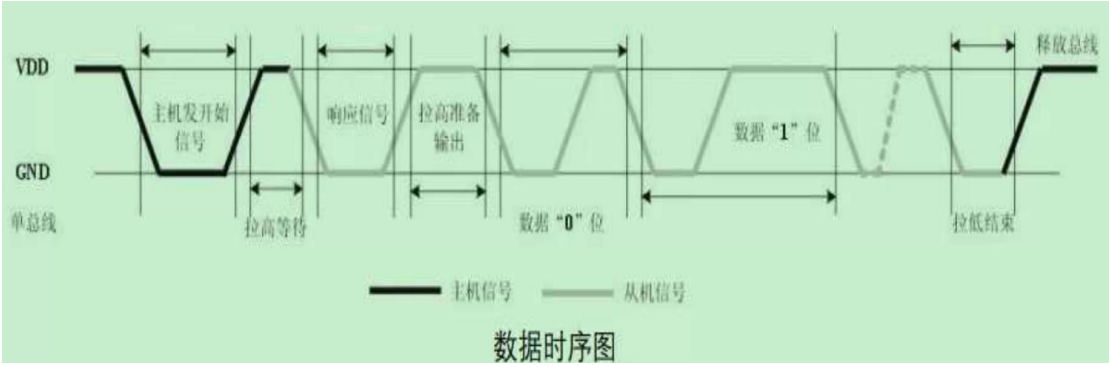


3）DHT11 温湿度传感器原理：



DHT11 是通过单总线与微处理器通讯，只需要一根线，一次传送 40 位数据，高位先出。数据格式 ：8bit 湿度整数数据 ＋8bit 湿度小数数据 ＋8bint 温度整数数据 ＋8bit 温度小数数据 ＋8bit 校验位。

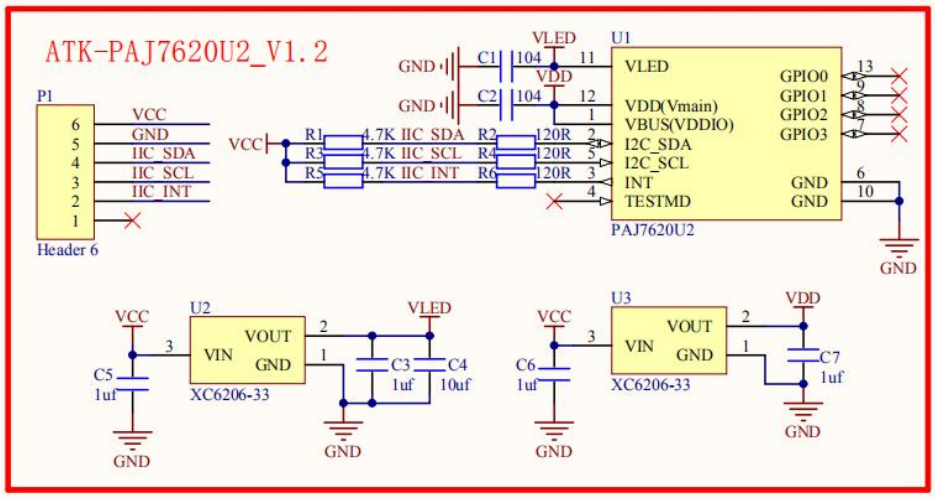
详细流程 ： MCU 发送起始信号(>18ms LOW) -> DHT 响应信号(80us LOW) -> DHT 通知 MCU 准备接受信号(80us HIGH) -> DHT 发送准备好的数据(“0”:50us LOW + 26~28us HIGH , “1”:50us LOW + 70us HIGH) -> DHT 结束信号(50us LOW)-> DHT 内部重测环境温湿度数据并记录数据等待下一次 MCU 的起始信号。由流程可知，每一次 MCU 获取的数据总是 DHT 上一次采集的数据，要想得到实时的数据，连续两次获取即可，官方不建议连续多次读取 DHT，每次读取的间隔时间大于 5 秒就足够获取到准确的数据，上电时 DHT 需要 1s 的时间稳定。



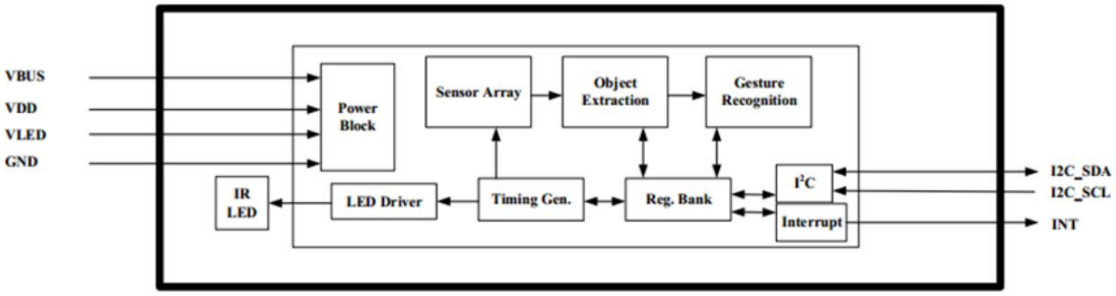
数据时序图

4）PAJ7620 手势识别传感器原理：

该模块采用原相科技（Pixart）公司的 PAJ7620U 芯片，内部集成了光学数组式传感器，以使复杂的手势和光标模式输出，支持上、下、左、右、前、后、顺时针旋转、逆时针旋转和挥动的手势识别，并支持物体接近检测。PAJ7620U2 工作时通过内部 LED 驱动器，驱动红外 LED 向外发射红外线信号，当传感器阵列在有效的距离中探测到物体时，目标信息提取阵列会对探测目标进行特征原始数据的获取，获取的数据会存在寄存器中，同时手势识别阵列会对原始数据进行识别处理，最后将手势结果存到寄存器中，用户可根据 I2C 接口对原始数据和手势识别的结果进行读取。



原理图

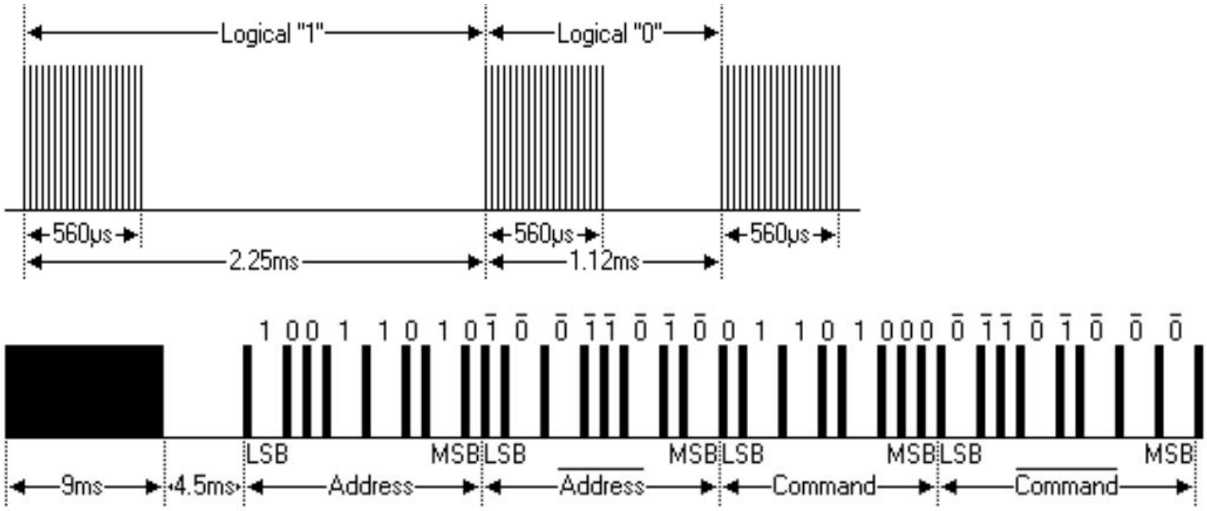


模块功能框图

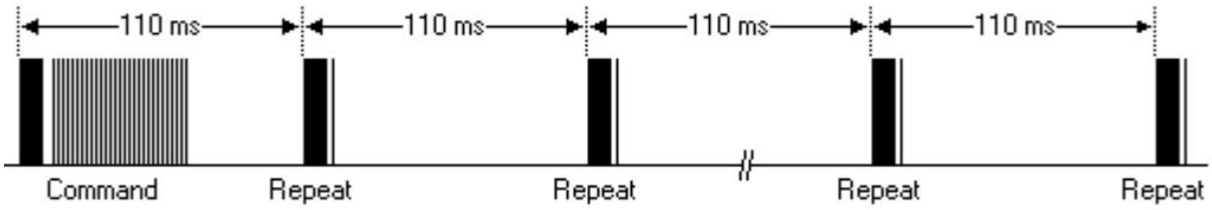
5）红外遥控原理：

红外遥控器将一连串的二进制脉冲码调制在 38kHz 的载波频率上然后再经红外发射二极管发射出去，红外传感器的集成接收和调制红外线。而红外线接收装置将接收到的信号解调成二进制脉冲码并滤除他杂波。

NEC 协议：8 个地址位+8 个命令位，载波频率 38kHz.

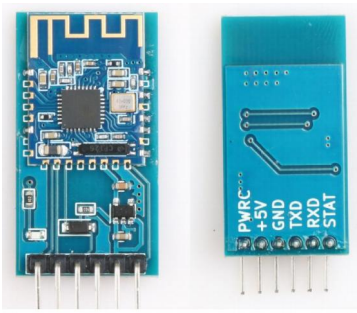


按下按钮立即释放



按下按钮持续一段时间

6）JDY-16 蓝牙模块原理：



蓝牙模块可以通过串口（SPI、IIC）和 MCU 控制设备进行数据传输。本实验中采用串口通信，手机作为主端，JDY-16 模块作为从端。

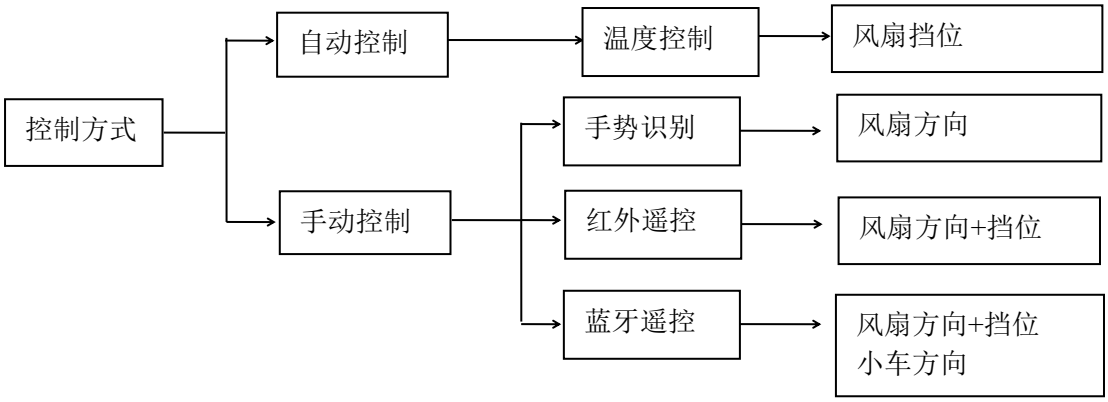
主端设备发起呼叫，查找附近蓝牙设备，若从端设备可被连接，则与其配对。配对后，从端蓝牙设备会记录主端的信任信息，配对后主端发起呼叫可被从端接收。

三、实验装置及设计：

实验装置：计算机，Arduino UNO 板，小车底座，DHT11 温湿度传感器，L298N 电

机驱动模块，电机，风扇，舵机，红外接收管，红外遥控器，有蓝牙功能的手机，JDY16 蓝牙模块，PAJ7620 手势识别传感器，电池，面包板，跳线若干。

本实验中风扇有两种控制模式：温度传感器自动控制挡位和手动控制，手动控制需要兼容手势识别，红外遥控，蓝牙控制三种控制方式，这三种方式均可控制风扇方向和挡位，同时蓝牙控制还可以切换模式控制小车运动。



四、实验过程和结果：

1）用 state 变量储存控制方式变量，默认 state=0 自动控制，若检测到红外/手势/蓝牙操作切换为 state=1 手动操作，通过红外遥控器或蓝牙可切换为自动控制。

2）由于舵机库和 PWM 输出冲突（在 Arduino 里的库封装里，它们用了同一个定时器 1），PWM 输出函数 analogWrite()转变为 01 输出。为了解决这个问题，重新编写函数控制舵机方向，直接通过 delay()函数来给舵机脉冲，控制舵机方向。

3）风扇设四个挡位，四个转动方向，每次转 15°，切换控制模式，控制小车方向等不同操作设不同 key，每个循环从各传感器获取 key，最后执行操作。

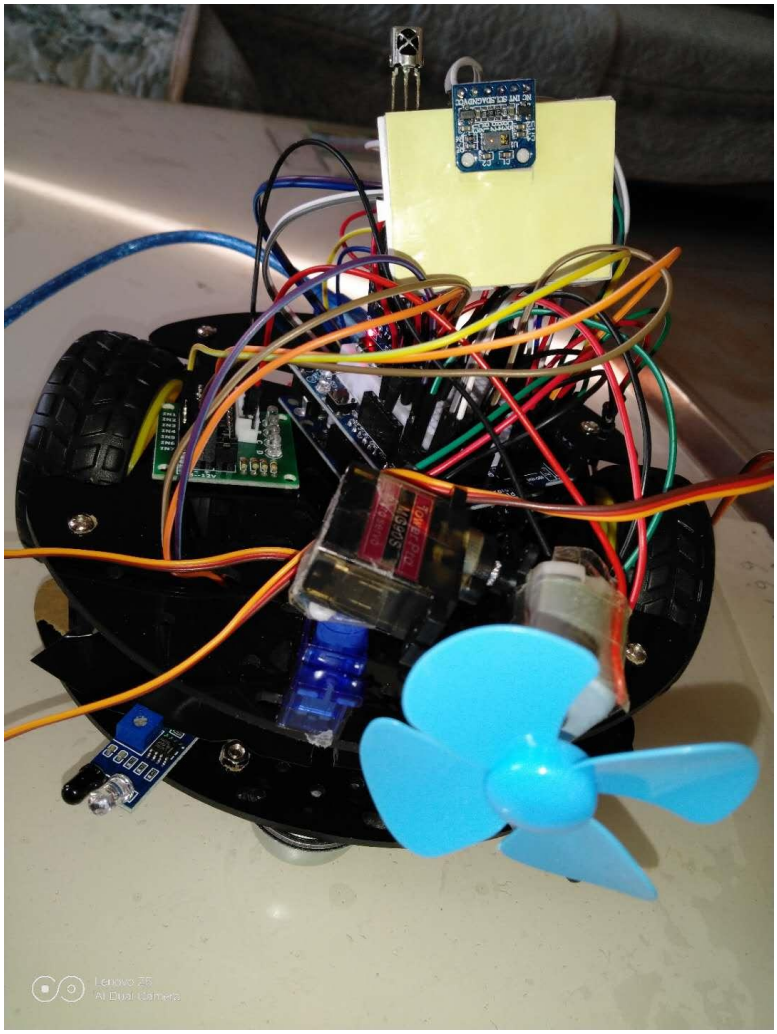
4）DHT11 传感器精度：0.01°C

5）手势识别有效范围：20cm

6）红外遥控有效范围：1m

7）蓝牙遥控界面：

*最终效果图：



五、实验结论：

本实验通过多种方式控制的智能风扇的设计，实现了温度自动控制风扇挡位，手势识别控制方向，也可通过红外遥控或手机蓝牙控制风扇挡位方向。将所有部件集成到小车上，可以通过蓝牙远程控制小车行动，并可自动避障，学习了多种传感器的共同控制。

六、参考文献：

[1]https://www.arduino.cc/

[2] https://www.cnblogs.com/sjsxk/p/5832406.html

[3]https://wenku.baidu.com/view/0f6886d226fff705cc170a48.html

[4]Arduino 实验例程 Keywish Tech firmware

[5]http://10.107.0.71/doku.php

[6]L298N 中文数据手册

[7]JDY-16 数据手册

[8]PAJ7620 手势识别传感器数据手册

附录（程序）

```
//IR_Remote_Fan
#include "IR_remote.h"
#include "dht11.h"
#include "paj7620.h"
#include <Wire.h>
//#include <LiquidCrystal.h>
//#include <SoftwareSerial.h>

//#define Software_TX 2
//#define Software_RX 3
#define IN1 13
#define IN2 12
#define IN3 11
#define IN4 10
#define   DHT11PIN     5        //温湿度传感器（十分垃圾）
#define   servopin     2        //舵机（有噪声）
#define   servopin2    4
#define   MotorPin     3        //电机（别忘了加控制板哦）
#define   IRPin        6        //红外接收器（十分垃圾）
#define GES_REACTION_TIME        500             // You can adjust the reaction time according to the actual circumstance.
#define GES_ENTRY_TIME           800             // When you want to recognize the Forward/Backward gestures, your gestures' reaction time must less than GES_ENTRY_TIME(0.8s).
#define GES_QUIT_TIME        1000
#define dangle          15

//SoftwareSerial BLE_JDY_16(Software_RX, Software_TX);//
dht11 DHT11;                    //咱也不懂
IRremote ir(IRPin);           //红外探头
//LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
//LiquidCrystal_I2C lcd(0x27, 16, 2);             // set the LCD address to 0x27 for a 16 chars and 2 line display

unsigned char keycode;
int flag = 0;        //遥控器 OK 键
int state = 0;       //主控选择
int angle=90;
int angle2=90;
//byte read_dat, jdy_dat;

void servopulse( int angle , int pin )
{
   int pulsewidth=(angle*11)+500;
   digitalWrite(pin,HIGH);
   delayMicroseconds(pulsewidth);
   digitalWrite(pin,LOW);
   delayMicroseconds(20000-pulsewidth);
}
```

```
void init_pin()                                    //电机初始化
{
   pinMode(IN1,OUTPUT);
   pinMode(IN2,OUTPUT);
   pinMode(IN3,OUTPUT);
   pinMode(IN4,OUTPUT);
   return;
 }

void car_forward()
{
   digitalWrite(IN3,HIGH);
   digitalWrite(IN4,LOW);
   digitalWrite(IN1,HIGH);
   digitalWrite(IN2,LOW);
   delay(500);
   digitalWrite(IN3,LOW);
   digitalWrite(IN4,LOW);
   digitalWrite(IN1,LOW);
   digitalWrite(IN2,LOW);
}

void car_back()
{
   digitalWrite(IN3,LOW);
   digitalWrite(IN4,HIGH);
   digitalWrite(IN1,LOW);
   digitalWrite(IN2,HIGH);
   delay(500);
   digitalWrite(IN3,LOW);
   digitalWrite(IN4,LOW);
   digitalWrite(IN1,LOW);
   digitalWrite(IN2,LOW);
}

/*
int readby(byte se)
{
   byte s0='0';
   return int(se-'0');
}
*/
void just(int set)
{
    if(angle<5){angle=0;}
    else if(angle>175){angle=180;}
    if(angle2<5){angle2=0;}
    else if(angle2>175){angle2=180;}

    switch (set)
    {
     case 0:
              digitalWrite(MotorPin,LOW);
              break;
     case 11:
              state=1;
              Serial.println("IR_KEYCODE_OK key");
```

```
            flag = !flag;
            //lcd.print("state:");
            //lcd.print(flag);
            //lcd.print("                ");
            digitalWrite(MotorPin, flag);
            break;
  case 5:
            Serial.println("LEFT");
           // lcd.print("Left                    ");
            if(angle>10){
               angle=angle-dangle;}
            servopulse(angle,servopin);
            break;
  case 6:
            Serial.println("RIGHT");
            //lcd.print("Right               ");
            if(angle<170){
               angle=angle+dangle;}
            //myservo.write(angle);
            servopulse(angle,servopin);
            break;
  case 7:
            Serial.println("UP");
            //lcd.print("Up                      ");
            if(angle2>10){
               angle2=angle2-dangle;}
            servopulse(angle2,servopin2);
            break;
  case 8:
            Serial.println("DOWN");
            //lcd.print("Down                    ");
            if(angle2<170){
               angle2=angle2+dangle;}
            servopulse(angle2,servopin2);
            break;
  case 9:
            Serial.println("IR_KEYCODE_STAR");
            state=0;
            break;
// case IR_KEYCODE_POUND:
  //           Serial.println("IR_KEYCODE_POUND");
     //        break;
  case 1:
            Serial.println("Mode1");
            state=1;flag=1;
            //lcd.print("On Mode1                ");
            digitalWrite(MotorPin,int(3.5*255/5) );
            break;
  case 2:
            Serial.println("Mode2");
            state=1;flag=1;
            //lcd.print("On Mode2                ");
            digitalWrite(MotorPin,int(4*255/5) );
            break;
  case 3:
            Serial.println("Mode3");
            state=1;flag=1;
```

```
                //lcd.print("On Mode3                    ");
                digitalWrite(MotorPin,int(4.5*255/5) );
                break;
        case 4:
                Serial.println("Mode4");
                state=1;flag=1;
                //lcd.print("On Mode4                    ");
                digitalWrite(MotorPin,int(5*255/5) );
                break;
        case 12:
                Serial.println("Forward");
                car_forward();
                break;
        case 13:
                Serial.println("Backward");
                car_back();
                break;
     }

}

void setup()
{
      Serial.begin(9600);
      ir.begin();
      //BLE_JDY_16.begin(9600);
      pinMode(servopin,OUTPUT);
      pinMode(servopin2,OUTPUT);
      servopulse(90,servopin);
      servopulse(90,servopin2);
      //lcd.begin(16, 2);
      pinMode(MotorPin,OUTPUT);
      pinMode(DHT11PIN,INPUT);
      digitalWrite(MotorPin, 0);
      if (paj7620Init())
   {
      Serial.print("INIT ERROR");
   }
   else
   {
      Serial.println("INIT OK");
   }
   Serial.println("Please input your gestures:\n");

}

void loop()
{
   int set=10;
   /*
          if (BLE_JDY_16.available()) {
            jdy_dat = BLE_JDY_16.read();
            set=readby(jdy_dat);
            just(set);
            //Serial.print(int(jdy_dat-'0'));
            delay(2);
          // Serial.write(jdy_dat);
```

```
        }
        if (Serial.available() > 0)
        {
              //read_dat = Serial.read();
              delay(2);
              //BLE_JDY_16.write(read_dat);
        }
        */
    byte ir_key = ir.getCode();
    uint8_t data = 0, error;

    int chk = DHT11.read(DHT11PIN);
      set=ir.IRsen(ir.getIrKey(ir_key));
      just(set);
    //Read gesture
   error = paj7620ReadReg(0x43, 1, &data);
   set=pajsen(data);

   if(!error){
      just(set);
      }
   //温度感应判断
    if(state==0){
         set=DHT11.temsen(DHT11.temperature);
         just(set);
         }

    delay(110);
}

    //LCD 显示温湿度
    /*
    lcd.setCursor(0, 0);
    lcd.print((float)DHT11.temperature);
    lcd.print("'C      ");
    lcd.print((float)DHT11.humidity, 2);
    lcd.print("%");
    lcd.setCursor(0, 1);

    Serial.print("Tep: ");
    Serial.print((float)DHT11.temperature);
    Serial.print("'C        ");
    Serial.print("Hum: ");
    Serial.print((float)DHT11.humidity, 2);
    Serial.println("%");
    */


//Blue
#include <SoftwareSerial.h>
#include "IR_remote.h"
#include "dht11.h"
#include <Wire.h>

#define Software_TX 9
#define Software_RX 8
//电机定义端口
```

```
#define IN1 13
#define IN2 12
#define IN3 11
#define IN4 10
#define IR1 A1    //left
#define IR2 A2    //forward
#define IR3 A0    //right
#define IR4 7    //back
#define    DHT11PIN     5        //温湿度传感器（十分垃圾）
#define    servopin     2        //舵机（有噪声）
#define    servopin2    4
#define    MotorPin     3        //电机（别忘了加控制板哦）
#define    IRPin        6        //红外接收器（十分垃圾）
#define dangle          15

SoftwareSerial BLE_JDY_16(Software_RX, Software_TX);
dht11 DHT11;                      //咱也不懂
IRremote ir(IRPin);            //红外探头

byte jdy_dat;
unsigned char keycode;
int flag = 0;          //遥控器 OK 键
int state = 0;       //主控选择
int angle=90;
int angle2=90;

//pwm 调速端口
//int left_pin = 5;
//int right_pin = 6;
int car_speed = 100;
int SNUM[4]={1,1,1,1};

void servopulse( int angle , int pin )
{
   int pulsewidth=(angle*11)+500;
   digitalWrite(pin,HIGH);
   delayMicroseconds(pulsewidth);
   digitalWrite(pin,LOW);
   delayMicroseconds(20000-pulsewidth);
}
/*
void set_speed()    //[0 ~255]                        //调速函数
{
      analogWrite(left_pin,car_speed);
      analogWrite(right_pin,car_speed);
}
*/
void init_pin()                                        //电机初始化
{
   pinMode(IN1,OUTPUT);
   pinMode(IN2,OUTPUT);
   pinMode(IN3,OUTPUT);
   pinMode(IN4,OUTPUT);
   return;
  }
```

```
void left_motor_forward()                    //左电机直行
{
  digitalWrite(IN3,HIGH);
  digitalWrite(IN4,LOW);
}
void left_motor_back()                       //左电机后退
{
  digitalWrite(IN3,LOW);
  digitalWrite(IN4,HIGH);
}
void right_motor_forward()                   //右电机直行
{
  digitalWrite(IN1,HIGH);
  digitalWrite(IN2,LOW);
}
void right_motor_back()                      //右电机后退
{
  digitalWrite(IN1,LOW);
  digitalWrite(IN2,HIGH);
}
void car_forward()                           //直行
{
  left_motor_forward();
  right_motor_forward();
  delay(1000);
}
void left_stop()                             //左电机停
{
  digitalWrite(IN3,LOW);
  digitalWrite(IN4,LOW);
  }
  void right_stop()                          //右电机停
{
  digitalWrite(IN1,LOW);
  digitalWrite(IN2,LOW);
}
void car_left()                              //车左转弯
{ left_motor_back();
  right_motor_forward();
  delay(1000);
}
void car_right()                             //车右转弯
{ left_motor_forward();
  right_motor_back();
  delay(1000);
}
void car_stop()                              //车停
{
  digitalWrite(IN1,LOW);
  digitalWrite(IN2,LOW);
  digitalWrite(IN3,LOW);
  digitalWrite(IN4,LOW);
  delay(1000);
  }
void car_back()                              //车后退
{
```

```
   left_motor_back();
   right_motor_back();
   delay(1000);
   }

void just(int set)
{
    if(angle<5){angle=0;}
    else if(angle>175){angle=180;}
    if(angle2<5){angle2=0;}
    else if(angle2>175){angle2=180;}

    switch (set)
    {
     case 0:
               digitalWrite(MotorPin,LOW);
               break;
     case 11:
               state=1;
               Serial.println("IR_KEYCODE_OK key");
               flag = !flag;
               //lcd.print("state:");
               //lcd.print(flag);
               //lcd.print("               ");
               digitalWrite(MotorPin, flag);
               break;
     case 5:
               Serial.println("LEFT");
              // lcd.print("Left                    ");
               if(angle>10){
                  angle=angle-dangle;}
               servopulse(angle,servopin);
               break;
     case 6:
               Serial.println("RIGHT");
               //lcd.print("Right               ");
               if(angle<170){
                  angle=angle+dangle;}
               //myservo.write(angle);
               servopulse(angle,servopin);
               break;
     case 7:
               Serial.println("UP");
               //lcd.print("Up                      ");
               if(angle2>10){
                  angle2=angle2-dangle;}
               servopulse(angle2,servopin2);
               break;
     case 8:
               Serial.println("DOWN");
               //lcd.print("Down                    ");
               if(angle2<170){
                  angle2=angle2+dangle;}
               servopulse(angle2,servopin2);
               break;
     case 9:
               Serial.println("IR_KEYCODE_STAR");
```

```
                state=0;
                break;
      // case IR_KEYCODE_POUND:
       //          Serial.println("IR_KEYCODE_POUND");
          //       break;
       case 1:
                Serial.println("Mode1");
                state=1;flag=1;
                //lcd.print("On Mode1                ");
                digitalWrite(MotorPin,int(3.5*255/5) );
                break;
       case 2:
                Serial.println("Mode2");
                state=1;flag=1;
                //lcd.print("On Mode2                ");
                digitalWrite(MotorPin,int(4*255/5) );
                break;
       case 3:
                Serial.println("Mode3");
                state=1;flag=1;
                //lcd.print("On Mode3                ");
                digitalWrite(MotorPin,int(4.5*255/5) );
                break;
       case 4:
                Serial.println("Mode4");
                state=1;flag=1;
                //lcd.print("On Mode4                ");
                digitalWrite(MotorPin,int(5*255/5) );
                break;
     }

}

void setup() {
   // put your setup code here, to run once:
      Serial.begin(9600);
      BLE_JDY_16.begin(9600);
      ir.begin();
      init_pin();
      pinMode(IR1,INPUT);
      pinMode(IR2,INPUT);
      pinMode(IR3,INPUT);
      pinMode(IR4,INPUT);
      pinMode(servopin,OUTPUT);
      pinMode(servopin2,OUTPUT);
      servopulse(90,servopin);
      servopulse(90,servopin2);
      //lcd.begin(16, 2);
      pinMode(MotorPin,OUTPUT);
      pinMode(DHT11PIN,INPUT);
      digitalWrite(MotorPin, 0);
}

void loop() {
   // put your main code here, to run repeatedly:
      int set=10;
      jdy_dat=0;
```

```
      if (BLE_JDY_16.available()) {
            jdy_dat = BLE_JDY_16.read();
            Serial.print("Read=");
            Serial.println(jdy_dat);
            SNUM[0] = digitalRead(IR1);//左传感器赋值
            //SNUM[1] = digitalRead(IR2); //中传感器赋值
            SNUM[2] = digitalRead(IR3); //右传感器赋值
            SNUM[3] = digitalRead(IR4);//后传感器
            for(int i=0;i<4;i++){Serial.println(SNUM[i]);}
            switch(jdy_dat)
            {
               case 10:
                  Serial.println("Forward");
                  if ((SNUM[0]==1)&&(SNUM[1]==1)&&(SNUM[2]==1))
                  {car_forward();}
                  break;
               case 11:
                  Serial.println("Backward");
                  if(SNUM[3]==1)
                  {car_back();}
                  break;
               case 12:
                  Serial.println("Left");
                  if((SNUM[0]==1)&&SNUM[1]==1)
                  {car_left();}
                  break;
               case 13:
                  Serial.println("Right");
                  if((SNUM[1]==1)&&(SNUM[2]==1))
                  {car_right();}
                  break;
               case 16:
                  Serial.println("Stop!");
                  car_stop();
                  break;
               default:break;
            }
            }
            byte ir_key = ir.getCode();
    uint8_t data = 0, error;

    int chk = DHT11.read(DHT11PIN);
      set=ir.IRsen(ir.getIrKey(ir_key));
      just(set);

   if(!error){
      just(set);
      }
   //温度感应判断
    if(state==0){
        set=DHT11.temsen(DHT11.temperature);
        just(set);
        }
           delay(200);
}

//IR_remote.cpp
```

```
#include "IR_remote.h"
#include "Keymap.h"

#ifndef __AVR_ATmega32U4__
#include <avr/interrupt.h>

volatile irparams_t irparams;
bool MATCH(uint8_t measured_ticks, uint8_t desired_us)
{
  return(measured_ticks >= desired_us - (desired_us>>2)-1 && measured_ticks <= desired_us + (desired_us>>2)+1);
}

ISR(TIMER_INTR_NAME)
{
  uint8_t irdata = (uint8_t)digitalRead(irparams.recvpin);
  irparams.timer++; // One more 50us tick
  if (irparams.rawlen >= RAWBUF)
  {
    // Buffer overflow
    irparams.rcvstate = STATE_STOP;
  }
  switch(irparams.rcvstate)
  {
    case STATE_IDLE: // In the middle of a gap
      if (irdata == MARK)
      {
        irparams.rawlen = 0;
        irparams.timer = 0;
        irparams.rcvstate = STATE_MARK;
      }
      break;
    case STATE_MARK: // timing MARK
      if (irdata == SPACE)
      {
        irparams.rawbuf[irparams.rawlen++] = irparams.timer;
        irparams.timer = 0;
        irparams.rcvstate = STATE_SPACE;
      }
      break;
    case STATE_SPACE: // timing SPACE
      if (irdata == MARK)
      {
        // SPACE just ended, record it
        irparams.rawbuf[irparams.rawlen++] = irparams.timer;
        irparams.timer = 0;
        irparams.rcvstate = STATE_MARK;
      }
      else
      { // SPACE
        if (irparams.timer > GAP_TICKS)
        {
          irparams.rcvstate = STATE_STOP;
          irparams.lastTime = millis();
        }
      }
```

```
            break;
        case STATE_STOP: // waiting, measuring gap
            if(millis() - irparams.lastTime > 120)
            {
                irparams.rawlen = 0;
                irparams.timer = 0;
                irparams.rcvstate = STATE_IDLE;
            }
            else if (irdata == MARK)
            {
                // reset gap timer
                irparams.timer = 0;
            }
            break;
    }
    // irparams.lastTime = new_time;
}
IRremote::IRremote(int pin)
{
    pinMode(pin,INPUT);
    irparams.recvpin = pin;
    // attachInterrupt(INT0, irISR, CHANGE);

    irDelayTime = 0;
    irIndex = 0;
    irRead = 0;
    irReady = false;
    irBuffer = "";
    irPressed = false;
    begin();
}

void IRremote::begin()
{
    cli();
    TIMER_CONFIG_NORMAL();
    TIMER_ENABLE_INTR;
    sei();    // enable interrupts
    irparams.rawlen = 0;
    irparams.rcvstate = STATE_IDLE;
}

void IRremote::end()
{
    EIMSK &= ~(1 << INT0);
}


ErrorStatus IRremote::decode()
{
    rawbuf = irparams.rawbuf;
    rawlen = irparams.rawlen;
    if (irparams.rcvstate != STATE_STOP)
    {
        return ERROR;
    }
```

```
    if (decodeNEC())
    {
      begin();
      return SUCCESS;
    }
    begin();
    return ERROR;
}

ErrorStatus IRremote::decodeNEC()
{
    static unsigned long repeat_value = 0xFFFFFFFF;
    static byte repeta_time = 0;
    uint32_t data = 0;
    int offset = 0; // Skip first space
    // Initial mark
    if (!MATCH(rawbuf[offset], NEC_HDR_MARK/50))
    {
      return ERROR;
    }
    offset++;
    // Check for repeat
    if (rawlen == 3 &&
      MATCH(rawbuf[offset], NEC_RPT_SPACE/50) &&
      MATCH(rawbuf[offset+1], NEC_BIT_MARK/50))
    {
        rawbuf[offset] = 0;
        rawbuf[offset+1] = 0;
        repeta_time++;
            repeta_time = 0;
            bits = 0;
            value = repeat_value;
            decode_type = NEC;
            return SUCCESS;
    }
    if (rawlen < (2 * NEC_BITS + 3))
    {
      return ERROR;
    }
    // Initial space
    if (!MATCH(rawbuf[offset], NEC_HDR_SPACE/50))
    {
      return ERROR;
    }
    rawbuf[offset] = 0;
    offset++;
    for (int i = 0; i < NEC_BITS; i++)
    {
      if (!MATCH(rawbuf[offset], NEC_BIT_MARK/50))
      {
        return ERROR;
      }
      rawbuf[offset] = 0;
      offset++;
      if (MATCH(rawbuf[offset], NEC_ONE_SPACE/50))
      {
        data = (data >> 1) | 0x80000000;
```

```
        }
        else if (MATCH(rawbuf[offset], NEC_ZERO_SPACE/50))
        {
            data >>= 1;
        }
        else
        {
            return ERROR;
        }
        offset++;
    }
    bits = NEC_BITS;
    value = data;
    repeat_value = data;
    decode_type = NEC;
    repeta_time = 0;
    return SUCCESS;
}


void IRremote::mark(uint16_t us)
{
    // Sends an IR mark for the specified number of microseconds.
    // The mark output is modulated at the PWM frequency.
    TIMER_ENABLE_PWM; // Enable pin 3 PWM output
    delayMicroseconds(us);
}

/**
  * \par Function
  *        space
  * \par Description
  *        Sends an IR mark for the specified number of microseconds.
  * \param[in]
  *        us - THe time of a PWM.
  * \par Output
  *        None
  * \par Return
  *        None
  * \par Others
  *        None
  */
/* Leave pin off for time (given in microseconds) */
void IRremote::space(uint16_t us)
{
    // Sends an IR space for the specified number of microseconds.
    // A space is no output, so the PWM output is disabled.
    TIMER_DISABLE_PWM; // Disable pin 3 PWM output
    delayMicroseconds(us);
}

/**
  * \par Function
  *        enableIROut
  * \par Description
  *        Enable an IR for the specified number of khz.
  * \param[in]
```

```
 *       us - THe time of a INTR.
 * \par Output
 *       None
 * \par Return
 *       None
 * \par Others
 *       None
 */
void IRremote::enableIROut(uint8_t khz)
{
  TIMER_DISABLE_INTR; //Timer2 disable Interrupt
  TIMER_CONFIG_KHZ(khz);
}

/**
 * \par Function
 *       enableIRIn
 * \par Description
 *       Enable an IR to write in.
 * \param[in]
 *       None
 * \par Output
 *       None
 * \par Return
 *       None
 * \par Others
 *       None
 */
// initialization
void IRremote::enableIRIn() {
  cli();
  // setup pulse clock timer interrupt
  //Prescale /8 (16M/8 = 0.5 microseconds per tick)
  // Therefore, the timer interval can range from 0.5 to 128 microseconds
  // depending on the reset value (255 to 0)
  TIMER_CONFIG_NORMAL();

  //Timer2 Overflow Interrupt Enable
  TIMER_ENABLE_INTR;

  //TIMER_RESET;

  sei();  // enable interrupts

  // initialize state machine variables
  irparams.rcvstate = STATE_IDLE;
  irparams.rawlen = 0;

  // set pin modes
  pinMode(irparams.recvpin, INPUT);
}

/**
 * \par Function
 *       sendRaw
 * \par Description
 *       Send the length of data with hz.
```

```
 * \param[in]
 *       buf[] - The data's buffer.
  * \param[in]
 *       len - The data's length.
  * \param[in]
 *       hz - The hz for sending data.
 * \par Output
 *       None
 * \par Return
 *       None
 * \par Others
 *       None
 */
void IRremote::sendRaw(unsigned int buf[], int len, uint8_t hz)
{
  enableIROut(hz);
  for (int i = 0; i < len; i++)
  {
    if (i & 1)
    {
      space(buf[i]);
    }
    else
    {
      mark(buf[i]);
    }
  }
  space(0); // Just to be sure
}

/**
 * \par Function
 *       getString
 * \par Description
 *       Get string in a INTR.
 * \param[in]
 *       None
 * \par Output
 *       None
 * \par Return
 *       Return the result in a IRQ.
 * \par Others
 *       None
 */
String IRremote::getString()
{
  if(decode())
  {
    irRead = ((value >> 8) >> 8) & 0xff;
    if(irRead == 0xa || irRead == 0xd)
    {
      irIndex = 0;
      irReady = true;
    }
    else
    {
      irBuffer += irRead;
```

```
            irIndex++;
        }
        irDelayTime = millis();
    }
    else
    {
        if(irRead > 0)
        {
            if(millis() - irDelayTime > 100)
            {
                irPressed = false;
                irRead = 0;
                irDelayTime = millis();
                Pre_Str = "";
            }
        }
    }
    if(irReady)
    {
        irReady = false;
        String s = String(irBuffer);
        Pre_Str = s;
        irBuffer = "";
        return s;
    }
    return Pre_Str;
}

/**
  * \par Function
  *       getCode
  * \par Description
  *       Get the reading code.
  * \param[in]
  *       None
  * \par Output
  *       None
  * \par Return
  *       Return the result of reading.
  * \par Others
  *       None
  */
unsigned char IRremote::getCode()
{
    irIndex = 0;
    loop();
    return irRead;
}
String IRremote::getKeyMap(byte keycode )
{
     byte i;
     for (i = 0; i < KEY_MAX; i++) {
             if (irkeymap[i].keycode == keycode)
               return irkeymap[i].keyname;
       }
       return "";
}
```

```
byte IRremote::getIrKey(byte keycode)
{
    byte i;
    for (i = 0; i < KEY_MAX; i++) {
            if (irkeymap[i].keycode == keycode)
              return i;
     }
     return 0xFF;
}
/**
 * \par Function
 *       sendString
 * \par Description
 *       Send data.
 * \param[in]
 *       s - The string you want to send.
 * \par Output
 *       None
 * \par Return
 *       None
 * \par Others
 *       None
 */
void IRremote::sendString(String s)
{
   unsigned long l;
   uint8_t data;
   s.concat('\n');
   for(int i = 0;i < s.length();i++)
   {
      data = s.charAt(i);
      l = 0x0000ffff & (uint8_t)(~data);
      l = l << 8;
      l = l + ((uint8_t)data);
      l = l << 16;
      l = l | 0x000000ff;
      sendNEC(l,32);
      delay(20);
   }
   enableIRIn();
}

/**
 * \par Function
 *       sendString
 * \par Description
 *       Send data.
 * \param[in]
 *       v - The string you want to send.
 * \par Output
 *       None
 * \par Return
 *       None
 * \par Others
 *       None
 */
```

```
void IRremote::sendString(float v)
{
   dtostrf(v,5, 8, floatString);
   sendString(floatString);
}

/**
 * \par Function
 *       sendNEC
 * \par Description
 *       Send NEC.
 * \param[in]
 *       data - The data you want to send.
  * \param[in]
 *       nbits - The data bit you want to send.
 * \par Output
 *       None
 * \par Return
 *       None
 * \par Others
 *       None
 */
void IRremote::sendNEC(unsigned long data, int nbits)
{

   enableIROut(38);
   mark(NEC_HDR_MARK);
   space(NEC_HDR_SPACE);
   for (int i = 0; i < nbits; i++)
   {
      if (data & 1)
      {
         mark(NEC_BIT_MARK);
         space(NEC_ONE_SPACE);
      }
      else
      {
         mark(NEC_BIT_MARK);
         space(NEC_ZERO_SPACE);
      }
      data >>= 1;
   }
   mark(NEC_BIT_MARK);
   space(0);
}

/**
 * \par Function
 *       loop
 * \par Description
 *       A circle of operation.
 * \param[in]
 *       None
 * \par Output0
 *       None
 * \par Return
 *       None
```

```
 * \par Others
 *      None
 */
void IRremote::loop()
{
  if(decode())
  {
    irRead = ((value >> 8) >> 8) & 0xff;
    irPressed = true;
    if(irRead == 0xa || irRead == 0xd)
    {
      irIndex = 0;
      irReady = true;
    }
    else
    {
      irBuffer += irRead;
      irIndex++;
      if(irIndex > 64)
      {
        irIndex = 0;
        irBuffer = "";
      }
    }
    irDelayTime =    millis();
  }
  else
  {
    if(irRead > 0)
    {
     // Serial.println(millis() - irDelayTime);
      if(millis() - irDelayTime > 0)
      {
        irPressed = false;
        irRead = 0;
        irDelayTime = millis();
      }
    }
  }
 // Serial.println(irRead, HEX);
}

/**
 * \par Function
 *      keyPressed
 * \par Description
 *      Press key.
 * \param[in]
 *      None
 * \par Output
 *      None
 * \par Return
 *      Return you the pressed key or not.
 * \par Others
 *      None
 */
boolean IRremote::keyPressed(unsigned char r)
```

```
{

   irIndex = 0;
   loop();
   return irRead == r;
}
#endif // !defined(__AVR_ATmega32U4__)



int IRremote::IRsen(byte key)
{
    switch (key)
    {
     case IR_KEYCODE_OK:
              Serial.println("IR_KEYCODE_OK key");
              return 11;
     case IR_KEYCODE_LEFT:
              Serial.println("IR_KEYCODE_LEFT");
              return 5;
     case IR_KEYCODE_RIGHT:
              Serial.println("IR_KEYCODE_RIGHT");
              return 6;
     case IR_KEYCODE_UP:
              Serial.println("IR_KEYCODE_UP");
              return 7;
     case IR_KEYCODE_DOWN:
              Serial.println("IR_KEYCODE_DOWN");
              return 8;
     case IR_KEYCODE_STAR:
              Serial.println("IR_KEYCODE_STAR");
              return 9;
     case IR_KEYCODE_POUND:
              Serial.println("IR_KEYCODE_POUND");
              return 10;
     case IR_KEYCODE_1:
              Serial.println("Mode1");
              return 1;
     case IR_KEYCODE_2:
              Serial.println("Mode2");
              return 2;
     case IR_KEYCODE_3:
              Serial.println("Mode3");
              return 3;
     case IR_KEYCODE_4:
              Serial.println("Mode4");
              return 4;
     default:
              return 10;
    }
}

//IR_remote.h

#ifndef IR_remote_h
#define IR_remote_h
```

```
/* Includes ------------------------------------------------------------------*/
#include <stdint.h>
#include <stdbool.h>
#include <Arduino.h>
#include "Keymap.h"

#ifdef ME_PORT_DEFINED
#endif // ME_PORT_DEFINED
#ifndef __AVR_ATmega32U4__
#define MARK    0
#define SPACE 1
#define NEC_BITS 32
#define USECPERTICK 50    // microseconds per clock interrupt tick
#define RAWBUF 80 // Length of raw duration buffer

typedef enum {ERROR = 0, SUCCESS = !ERROR} ErrorStatus;

#define NEC_HDR_MARK   9000
#define NEC_HDR_SPACE  4500
#define NEC_BIT_MARK    560
#define NEC_ONE_SPACE  1600
#define NEC_ZERO_SPACE 560
#define NEC_RPT_SPACE   2250
#define NEC_RPT_PERIOD 110000


#define _GAP 5000 // Minimum map between transmissions

// receiver states
#define STATE_IDLE         2
#define STATE_MARK         3
#define STATE_SPACE        4
#define STATE_STOP         5


// Values for decode_type
#define NEC 1
#define SONY 2
#define RC5 3
#define RC6 4
#define DISH 5
#define SHARP 6
#define PANASONIC 7
#define JVC 8
#define SANYO 9
#define MITSUBISHI 10
#define SAMSUNG 11
#define LG 12
#define UNKNOWN -1

#define TOPBIT 0x80000000


#ifdef F_CPU
#define SYSCLOCK F_CPU          // main Arduino clock
#else
#define SYSCLOCK 16000000    // main Arduino clock
```

```
#endif


#define _GAP 5000 // Minimum map between transmissions
#define GAP_TICKS (_GAP/USECPERTICK)


#define TIMER_DISABLE_INTR      (TIMSK2 = 0)
#define TIMER_ENABLE_PWM        (TCCR2A |= _BV(COM2B1))
#define TIMER_DISABLE_PWM       (TCCR2A &= ~(_BV(COM2B1)))
#define TIMER_ENABLE_INTR       (TIMSK2 = _BV(OCIE2A))
#define TIMER_DISABLE_INTR      (TIMSK2 = 0)
#define TIMER_INTR_NAME         TIMER2_COMPA_vect
#define TIMER_CONFIG_KHZ(val) ({ \
  const uint8_t pwmval = F_CPU / 2000 / (val); \
  TCCR2A = _BV(WGM20); \
  TCCR2B = _BV(WGM22) | _BV(CS20); \
  OCR2A = pwmval; \
  OCR2B = pwmval / 3; \
})

#define TIMER_COUNT_TOP         (SYSCLOCK * USECPERTICK / 1000000)
#if (TIMER_COUNT_TOP < 256)
#define TIMER_CONFIG_NORMAL() ({ \
  TCCR2A = _BV(WGM21); \
  TCCR2B = _BV(CS20); \
  OCR2A = TIMER_COUNT_TOP; \
  TCNT2 = 0; \
})
#else
#define TIMER_CONFIG_NORMAL() ({ \
  TCCR2A = _BV(WGM21); \
  TCCR2B = _BV(CS21); \
  OCR2A = TIMER_COUNT_TOP / 8; \
  TCNT2 = 0; \
})
#endif

// information for the interrupt handler
typedef struct {
  uint8_t recvpin;                // pin for IR data from detector
  volatile uint8_t rcvstate;              // state machine
  volatile uint32_t lastTime;
  unsigned int timer;         //
  volatile uint8_t rawbuf[RAWBUF]; // raw data
  volatile uint8_t rawlen;            // counter of entries in rawbuf
} irparams_t;


class IRremote
{
public:

  IRremote(int pin);
  ErrorStatus decode();
  int IRsen(byte key);
  void begin();
```

```
    void end();
    void loop();
    boolean keyPressed(unsigned char r);
    // void resume();

    int8_t decode_type; // NEC, SONY, RC5, UNKNOWN
    unsigned long value; // Decoded value
    uint8_t bits; // Number of bits in decoded value
    volatile uint8_t *rawbuf; // Raw intervals in .5 us ticks
    int rawlen; // Number of records in rawbuf.
    String getString();
    unsigned char getCode();
    String getKeyMap( byte keycode);
    byte getIrKey(byte keycode);
    void sendString(String s);
    void sendString(float v);
    void sendNEC(unsigned long data, int nbits);
    void sendRaw(unsigned int buf[], int len, uint8_t hz);
    void enableIROut(uint8_t khz);
    void enableIRIn();
    void mark(uint16_t us);
    void space(uint16_t us);
private:
    ErrorStatus decodeNEC();
    int16_t irIndex;
    char irRead;
    char floatString[5];
    boolean irReady;
    boolean irPressed;
    String irBuffer;
    String Pre_Str;
    double irDelayTime;
};
#endif // !__AVR_ATmega32U4__
#endif

//Keymap.cpp
#include "Keymap.h"

ST_KEY_MAP irkeymap[KEY_MAX] = {
        {"1", 0x45},
        {"2", 0x46},
        {"3", 0x47},
        {"4", 0x44},
        {"5", 0x40},
        {"6", 0x43},
        {"7", 0x07},
        {"8", 0x15},
        {"9", 0x09},
        {"0", 0x19},
        {"*", 0x16},
        {"#", 0x0D},
        {"up", 0x18},
        {"down", 0x52},
        {"ok", 0x1C},
        {"left", 0x08},
        {"right", 0x5A}
```

```
};

//keymap.h
#ifndef _KEYMAY_H_
#define _KEYMAY_H_
#include <Arduino.h>
#define KEY_MAX 18
typedef struct
{
    String keyname;
    byte keycode;
}ST_KEY_MAP;

typedef enum {
    IR_KEYCODE_1 = 0,
    IR_KEYCODE_2,
    IR_KEYCODE_3,
    IR_KEYCODE_4,
    IR_KEYCODE_5,
    IR_KEYCODE_6,
    IR_KEYCODE_7,
    IR_KEYCODE_8,
    IR_KEYCODE_9,
    IR_KEYCODE_0,
    IR_KEYCODE_STAR,          // *
    IR_KEYCODE_POUND,         // #
    IR_KEYCODE_UP,
    IR_KEYCODE_DOWN,
    IR_KEYCODE_OK,
    IR_KEYCODE_LEFT,
    IR_KEYCODE_RIGHT,
}E_IR_KEYCODE;

extern ST_KEY_MAP irkeymap[];
#endif    /* _KEYMAY_H_ */

//dht11.h
```

```
#ifndef dht11_h
#define dht11_h

#if defined(ARDUINO) && (ARDUINO >= 100)
#include <Arduino.h>
#else
#include <WProgram.h>
#endif

#define DHT11LIB_VERSION "0.4.1"

#define DHTLIB_OK                       0
#define DHTLIB_ERROR_CHECKSUM     -1
#define DHTLIB_ERROR_TIMEOUT -2

class dht11
{
public:
        int read(int pin);
        int temsen(double tem);
                int humidity;
                int temperature;
};
#endif
//
// END OF FILE
//

//dht11.cpp
#include "dht11.h"

// Return values:
// DHTLIB_OK
// DHTLIB_ERROR_CHECKSUM
// DHTLIB_ERROR_TIMEOUT
int dht11::read(int pin)
{
                // BUFFER TO RECEIVE
```

```
uint8_t bits[5];
uint8_t cnt = 7;
uint8_t idx = 0;

// EMPTY BUFFER
for (int i=0; i< 5; i++) bits[i] = 0;

// REQUEST SAMPLE
pinMode(pin, OUTPUT);
digitalWrite(pin, LOW);
delay(18);
digitalWrite(pin, HIGH);
delayMicroseconds(40);
pinMode(pin, INPUT);

// ACKNOWLEDGE or TIMEOUT
unsigned int loopCnt = 10000;
while(digitalRead(pin) == LOW)
    if (loopCnt-- == 0) return DHTLIB_ERROR_TIMEOUT;

loopCnt = 10000;
while(digitalRead(pin) == HIGH)
    if (loopCnt-- == 0) return DHTLIB_ERROR_TIMEOUT;

// READ OUTPUT - 40 BITS => 5 BYTES or TIMEOUT
for (int i=0; i<40; i++)
{
    loopCnt = 10000;
    while(digitalRead(pin) == LOW)
        if (loopCnt-- == 0) return DHTLIB_ERROR_TIMEOUT;

    unsigned long t = micros();

    loopCnt = 10000;
    while(digitalRead(pin) == HIGH)
        if (loopCnt-- == 0) return DHTLIB_ERROR_TIMEOUT;

    if ((micros() - t) > 40) bits[idx] |= (1 << cnt);
```

```
            if (cnt == 0)      // next byte?
            {
                cnt = 7;       // restart at MSB
                idx++;           // next byte!
            }
            else cnt--;
        }

        // WRITE TO RIGHT VARS
        // as bits[1] and bits[3] are allways zero they are omitted in formulas.
        humidity       = bits[0];
        temperature = bits[2];

        uint8_t sum = bits[0] + bits[2];

        if (bits[4] != sum) return DHTLIB_ERROR_CHECKSUM;
        return DHTLIB_OK;
}

int dht11::temsen(double tem )
{
    if(tem<25){
      //lcd.print("Off                          ");
      //Serial.println("Off");
      return 0;}
    else{
      //lcd.print("On    ");
      if(tem<26.5){
        //lcd.print("Mode1");
        Serial.println("Mode1");
        return 1;}
      else if(tem<28){
        //lcd.print("Mode2");
        return 2;}
      else if(tem<29.5){
        //lcd.print("Mode3");
        return 3;}
      else{
```

```
            //lcd.print("Mode4");
            return 4;}
        }
}

/*
 //温度控制风扇挡位
void temsen(double tem,int flag )
{
      double voltage=0;
      if(tem<25){
         lcd.print("Off                        ");
         Serial.println("Off");
         voltage=0;
         flag=0;}
      else{
         lcd.print("On    ");
         if(tem<26.5){
            lcd.print("Mode1");
            Serial.println("Mode1");
            voltage=5;}
         else if(tem<28){
            lcd.print("Mode2");
            voltage=4;}
         else if(tem<29.5){
            lcd.print("Mode3");
            voltage=4.5;}
         else{
            lcd.print("Mode4");
            voltage=5;}
      }
      flag=1;
      double value=voltage*255/5;
      analogWrite(MotorPin, value);
}

 */
//
```

```
// END OF FILE
//

//paj7620.h

#ifndef __PAJ7620_H__
#define __PAJ7620_H__

#define BIT(x)    1 << x

// REGISTER DESCRIPTION
#define PAJ7620_VAL(val, maskbit)           ( val << maskbit )
#define PAJ7620_ADDR_BASE                   0x00

// REGISTER BANK SELECT
#define PAJ7620_REGITER_BANK_SEL        (PAJ7620_ADDR_BASE + 0xEF) //W

// DEVICE ID
#define PAJ7620_ID    0x73

// REGISTER BANK 0
#define PAJ7620_ADDR_SUSPEND_CMD      (PAJ7620_ADDR_BASE + 0x3)   //W
#define PAJ7620_ADDR_GES_PS_DET_MASK_0        (PAJ7620_ADDR_BASE + 0x41) //RW
#define PAJ7620_ADDR_GES_PS_DET_MASK_1        (PAJ7620_ADDR_BASE + 0x42) //RW
#define PAJ7620_ADDR_GES_PS_DET_FLAG_0        (PAJ7620_ADDR_BASE + 0x43) //R
#define PAJ7620_ADDR_GES_PS_DET_FLAG_1        (PAJ7620_ADDR_BASE + 0x44) //R
#define PAJ7620_ADDR_STATE_INDICATOR  (PAJ7620_ADDR_BASE + 0x45) //R
#define PAJ7620_ADDR_PS_HIGH_THRESHOLD   (PAJ7620_ADDR_BASE + 0x69) //RW
#define PAJ7620_ADDR_PS_LOW_THRESHOLD    (PAJ7620_ADDR_BASE + 0x6A)//RW
#define PAJ7620_ADDR_PS_APPROACH_STATE    (PAJ7620_ADDR_BASE + 0x6B)//R
#define PAJ7620_ADDR_PS_RAW_DATA        (PAJ7620_ADDR_BASE + 0x6C)//R

// REGISTER BANK 1
#define PAJ7620_ADDR_PS_GAIN              (PAJ7620_ADDR_BASE + 0x44) //RW
#define PAJ7620_ADDR_IDLE_S1_STEP_0         (PAJ7620_ADDR_BASE + 0x67) //RW
#define PAJ7620_ADDR_IDLE_S1_STEP_1         (PAJ7620_ADDR_BASE + 0x68) //RW
#define PAJ7620_ADDR_IDLE_S2_STEP_0         (PAJ7620_ADDR_BASE + 0x69) //RW
#define PAJ7620_ADDR_IDLE_S2_STEP_1         (PAJ7620_ADDR_BASE + 0x6A)//RW
```

```
#define PAJ7620_ADDR_OP_TO_S1_STEP_0   (PAJ7620_ADDR_BASE + 0x6B) //RW
#define PAJ7620_ADDR_OP_TO_S1_STEP_1   (PAJ7620_ADDR_BASE + 0x6C) //RW
#define PAJ7620_ADDR_OP_TO_S2_STEP_0   (PAJ7620_ADDR_BASE + 0x6D)//RW
#define PAJ7620_ADDR_OP_TO_S2_STEP_1   (PAJ7620_ADDR_BASE + 0x6E) //RW
#define PAJ7620_ADDR_OPERATION_ENABLE     (PAJ7620_ADDR_BASE + 0x72) //RW

// PAJ7620_REGITER_BANK_SEL
#define PAJ7620_BANK0           PAJ7620_VAL(0,0)
#define PAJ7620_BANK1     PAJ7620_VAL(1,0)

// PAJ7620_ADDR_SUSPEND_CMD
#define PAJ7620_I2C_WAKEUP  PAJ7620_VAL(1,0)
#define PAJ7620_I2C_SUSPEND PAJ7620_VAL(0,0)

// PAJ7620_ADDR_OPERATION_ENABLE
#define PAJ7620_ENABLE          PAJ7620_VAL(1,0)
#define PAJ7620_DISABLE         PAJ7620_VAL(0,0)

typedef enum {
        BANK0 = 0,
        BANK1,
} bank_e;

#define GES_RIGHT_FLAG                  PAJ7620_VAL(1,0)
#define GES_LEFT_FLAG                   PAJ7620_VAL(1,1)
#define GES_UP_FLAG                         PAJ7620_VAL(1,2)
#define GES_DOWN_FLAG                   PAJ7620_VAL(1,3)
#define GES_FORWARD_FLAG                PAJ7620_VAL(1,4)
#define GES_BACKWARD_FLAG                   PAJ7620_VAL(1,5)
#define GES_CLOCKWISE_FLAG                  PAJ7620_VAL(1,6)
#define GES_COUNT_CLOCKWISE_FLAG    PAJ7620_VAL(1,7)
#define GES_WAVE_FLAG                   PAJ7620_VAL(1,0)

/*
enum {
        // REGISTER 0
        GES_RIGHT_FLAG                  = BIT(0),
        GES_LEFT_FLAG                   = BIT(1),
```

```
        GES_UP_FLAG                 = BIT(2),
        GES_DOWN_FLAG               = BIT(3),
        GES_FORWARD_FLAG            = BIT(4),
        GES_BACKWARD_FLAG           = BIT(5),
        GES_CLOCKWISE_FLAG          = BIT(6),
        GES_COUNT_CLOCKWISE_FLAG = BIT(7),
        //REGISTER 1
        GES_WAVE_FLAG           = BIT(0),
};
*/


#define INIT_REG_ARRAY_SIZE (sizeof(initRegisterArray)/sizeof(initRegisterArray[0]))

int pajsen(uint8_t data);
uint8_t paj7620Init(void);
uint8_t paj7620WriteReg(uint8_t addr, uint8_t cmd);
uint8_t paj7620ReadReg(uint8_t addr, uint8_t qty, uint8_t data[]);
void paj7620SelectBank(bank_e bank);


#endif


//paj7620.cpp
#include <Wire.h>
#include "paj7620.h"
#include <Arduino.h>

#define GES_REACTION_TIME       500             // You can adjust the reaction time according to the actual circumstance.
#define GES_ENTRY_TIME          800             // When you want to recognize the Forward/Backward gestures, your gestures' reaction time must less than GES_ENTRY_TIME(0.8s).
#define GES_QUIT_TIME       1000

// PAJ7620U2_20140305.asc
/* Registers' initialization data */
unsigned char initRegisterArray[][2] = {    // Initial Gesture
```

```
{0xEF,0x00},
        {0x32,0x29},
        {0x33,0x01},
        {0x34,0x00},
        {0x35,0x01},
        {0x36,0x00},
        {0x37,0x07},
        {0x38,0x17},
        {0x39,0x06},
        {0x3A,0x12},
        {0x3F,0x00},
        {0x40,0x02},
        {0x41,0xFF},
        {0x42,0x01},
        {0x46,0x2D},
        {0x47,0x0F},
        {0x48,0x3C},
        {0x49,0x00},
        {0x4A,0x1E},
        {0x4B,0x00},
        {0x4C,0x20},
        {0x4D,0x00},
        {0x4E,0x1A},
        {0x4F,0x14},
        {0x50,0x00},
        {0x51,0x10},
        {0x52,0x00},
        {0x5C,0x02},
        {0x5D,0x00},
        {0x5E,0x10},
        {0x5F,0x3F},
        {0x60,0x27},
        {0x61,0x28},
        {0x62,0x00},
        {0x63,0x03},
        {0x64,0xF7},
        {0x65,0x03},
        {0x66,0xD9},
```

{0x67,0x03},
{0x68,0x01},
{0x69,0xC8},
{0x6A,0x40},
{0x6D,0x04},
{0x6E,0x00},
{0x6F,0x00},
{0x70,0x80},
{0x71,0x00},
{0x72,0x00},
{0x73,0x00},
{0x74,0xF0},
{0x75,0x00},
{0x80,0x42},
{0x81,0x44},
{0x82,0x04},
{0x83,0x20},
{0x84,0x20},
{0x85,0x00},
{0x86,0x10},
{0x87,0x00},
{0x88,0x05},
{0x89,0x18},
{0x8A,0x10},
{0x8B,0x01},
{0x8C,0x37},
{0x8D,0x00},
{0x8E,0xF0},
{0x8F,0x81},
{0x90,0x06},
{0x91,0x06},
{0x92,0x1E},
{0x93,0x0D},
{0x94,0x0A},
{0x95,0x0A},
{0x96,0x0C},
{0x97,0x05},
{0x98,0x0A},

```
{0x99,0x41},
{0x9A,0x14},
{0x9B,0x0A},
{0x9C,0x3F},
{0x9D,0x33},
{0x9E,0xAE},
{0x9F,0xF9},
{0xA0,0x48},
{0xA1,0x13},
{0xA2,0x10},
{0xA3,0x08},
{0xA4,0x30},
{0xA5,0x19},
{0xA6,0x10},
{0xA7,0x08},
{0xA8,0x24},
{0xA9,0x04},
{0xAA,0x1E},
{0xAB,0x1E},
{0xCC,0x19},
{0xCD,0x0B},
{0xCE,0x13},
{0xCF,0x64},
{0xD0,0x21},
{0xD1,0x0F},
{0xD2,0x88},
{0xE0,0x01},
{0xE1,0x04},
{0xE2,0x41},
{0xE3,0xD6},
{0xE4,0x00},
{0xE5,0x0C},
{0xE6,0x0A},
{0xE7,0x00},
{0xE8,0x00},
{0xE9,0x00},
{0xEE,0x07},
{0xEF,0x01},
```

```
{0x00,0x1E},
{0x01,0x1E},
{0x02,0x0F},
{0x03,0x10},
{0x04,0x02},
{0x05,0x00},
{0x06,0xB0},
{0x07,0x04},
{0x08,0x0D},
{0x09,0x0E},
{0x0A,0x9C},
{0x0B,0x04},
{0x0C,0x05},
{0x0D,0x0F},
{0x0E,0x02},
{0x0F,0x12},
{0x10,0x02},
{0x11,0x02},
{0x12,0x00},
{0x13,0x01},
{0x14,0x05},
{0x15,0x07},
{0x16,0x05},
{0x17,0x07},
{0x18,0x01},
{0x19,0x04},
{0x1A,0x05},
{0x1B,0x0C},
{0x1C,0x2A},
{0x1D,0x01},
{0x1E,0x00},
{0x21,0x00},
{0x22,0x00},
{0x23,0x00},
{0x25,0x01},
{0x26,0x00},
{0x27,0x39},
{0x28,0x7F},
```

```
{0x29,0x08},
{0x30,0x03},
{0x31,0x00},
{0x32,0x1A},
{0x33,0x1A},
{0x34,0x07},
{0x35,0x07},
{0x36,0x01},
{0x37,0xFF},
{0x38,0x36},
{0x39,0x07},
{0x3A,0x00},
{0x3E,0xFF},
{0x3F,0x00},
{0x40,0x77},
{0x41,0x40},
{0x42,0x00},
{0x43,0x30},
{0x44,0xA0},
{0x45,0x5C},
{0x46,0x00},
{0x47,0x00},
{0x48,0x58},
{0x4A,0x1E},
{0x4B,0x1E},
{0x4C,0x00},
{0x4D,0x00},
{0x4E,0xA0},
{0x4F,0x80},
{0x50,0x00},
{0x51,0x00},
{0x52,0x00},
{0x53,0x00},
{0x54,0x00},
{0x57,0x80},
{0x59,0x10},
{0x5A,0x08},
{0x5B,0x94},
```

```
        {0x5C,0xE8},
        {0x5D,0x08},
        {0x5E,0x3D},
        {0x5F,0x99},
        {0x60,0x45},
        {0x61,0x40},
        {0x63,0x2D},
        {0x64,0x02},
        {0x65,0x96},
        {0x66,0x00},
        {0x67,0x97},
        {0x68,0x01},
        {0x69,0xCD},
        {0x6A,0x01},
        {0x6B,0xB0},
        {0x6C,0x04},
        {0x6D,0x2C},
        {0x6E,0x01},
        {0x6F,0x32},
        {0x71,0x00},
        {0x72,0x01},
        {0x73,0x35},
        {0x74,0x00},
        {0x75,0x33},
        {0x76,0x31},
        {0x77,0x01},
        {0x7C,0x84},
        {0x7D,0x03},
        {0x7E,0x01},
};


/*****************************************************************
  * Function Name: paj7620WriteReg
  * Description:    PAJ7620 Write reg cmd
  * Parameters: addr:reg address; cmd:function data
  * Return: error code; success: return 0
*****************************************************************/
```

```
uint8_t paj7620WriteReg(uint8_t addr, uint8_t cmd)
{
        //char i = 1;
        Wire.beginTransmission(PAJ7620_ID);             // start transmission to device
        //write cmd
        Wire.write(addr);                               // send register address
        Wire.write(cmd);                                // send value to write
  //    i = Wire.endTransmission();                   // end transmission
        if(Wire.endTransmission())
    {
            Serial.print("Transmission error!!!\n");
        }
        return Wire.endTransmission();
}

/*********************************************************************
 * Function Name: paj7620ReadReg
 * Description:    PAJ7620 read reg data
 * Parameters: addr:reg address;
 *                         qty:number of data to read, addr continuously increase;
 *                         data[]:storage memory start address
 * Return: error code; success: return 0
*********************************************************************/
uint8_t paj7620ReadReg(uint8_t addr, uint8_t qty, uint8_t data[])
{
        //uint8_t error;
        Wire.beginTransmission(PAJ7620_ID);
        Wire.write(addr);
        //error = Wire.endTransmission();

        if(Wire.endTransmission())
    {
            Serial.print("Transmission error!!!\n");
            return Wire.endTransmission(); //return error code
        }

        Wire.requestFrom((int)PAJ7620_ID, (int)qty);
```

```
            while (Wire.available())
            {
                *data = Wire.read();

#ifdef debug        //debug
      Serial.print("addr:");
      Serial.print(addr++, HEX);
      Serial.print("    data:");
      Serial.println(*data, HEX);
#endif

            data++;
            }
            return 0;
}

/**********************************************************************
 * Function Name: paj7620SelectBank
 * Description:    PAJ7620 select register bank
 * Parameters: BANK0, BANK1
 * Return: none
**********************************************************************/
void paj7620SelectBank(bank_e bank)
{
      switch(bank){
            case BANK0:
                  paj7620WriteReg(PAJ7620_REGITER_BANK_SEL, PAJ7620_BANK0);
                  break;
            case BANK1:
                  paj7620WriteReg(PAJ7620_REGITER_BANK_SEL, PAJ7620_BANK1);
                  break;
            default:
                  break;
            }
}

/**********************************************************************
 * Function Name: paj7620Init
```

```
 * Description:    PAJ7620 REG INIT
 * Parameters: none
 * Return: error code; success: return 0
*****************************************************************/
uint8_t paj7620Init(void)
{
        //Near_normal_mode_V5_6.15mm_121017 for 940nm
        //int i = 0;
        //uint8_t error;
        uint8_t data0 = 0;//, data1 = 0;
        //wakeup the sensor
        delayMicroseconds(700);      //Wait 700us for PAJ7620U2 to stabilize

        Wire.begin();

        Serial.println("INIT SENSOR...");

        paj7620SelectBank(BANK0);
        paj7620SelectBank(BANK0);

        //error =
        paj7620ReadReg(0, 1, &data0);
 /*
        if (error)
        {
            return error;
        }
        error = paj7620ReadReg(1, 1, &data1);
        if (error)
        {
            return error;
        }

        Serial.print("Addr0 =");
        Serial.print(data0 , HEX);
        Serial.print(",    Addr1 =");
        Serial.println(data1 , HEX);
```

```
        if ( (data0 != 0x20 ) || (data1 != 0x76) )
        {
            return 0xff;
        }

        if ( data0 == 0x20 )
        {
            Serial.println("wake-up finish.");
        }
        */
        for (int i = 0; i < INIT_REG_ARRAY_SIZE; i++)
        {
            paj7620WriteReg(initRegisterArray[i][0], initRegisterArray[i][1]);
        }

        paj7620SelectBank(BANK0);    //gesture flage reg in Bank0

        Serial.println("Paj7620 initialize register finished.");
        return 0;
}


 int pajsen(uint8_t data)
 {
   switch (data)                          // When different gestures be detected, the variable 'data' will be set to different values by paj7620ReadReg(0x43, 1, &data).
     {
       case GES_RIGHT_FLAG:
         delay(GES_ENTRY_TIME);
         paj7620ReadReg(0x43, 1, &data);
         if(data == GES_FORWARD_FLAG) {
           Serial.println("Gesture_Forward");
           delay(GES_QUIT_TIME);
           return 10;
           }
         else if(data == GES_BACKWARD_FLAG) {
           Serial.println("Gesture_Backward");
           delay(GES_QUIT_TIME);
```

```
        return 10;
        }
    else{
        Serial.println("Gesture_Right");
        return 6;
        /*
        if(angle<180){
            lcd.print("Right              ");
            angle=angle+dangle;}
      servopulse(angle,servopin);
      */
        }
case GES_LEFT_FLAG:
    delay(GES_ENTRY_TIME);
    paj7620ReadReg(0x43, 1, &data);
    if(data == GES_FORWARD_FLAG) {
        Serial.println("Gesture_Forward");
        delay(GES_QUIT_TIME);
        return 10;
        }
    else if(data == GES_BACKWARD_FLAG) {
        Serial.println("Gesture_Backward");
        delay(GES_QUIT_TIME);
        return 10;
        }
    else{
        Serial.println("Gesture_Left");
        return 5;
        }
case GES_UP_FLAG:
    delay(GES_ENTRY_TIME);
    paj7620ReadReg(0x43, 1, &data);
    if(data == GES_FORWARD_FLAG) {
        Serial.println("Gesture_Forward");
        delay(GES_QUIT_TIME);
        return 10;
        }
    else if(data == GES_BACKWARD_FLAG) {
```

```
            Serial.println("Gesture_Backward");
            delay(GES_QUIT_TIME);
            return 10;
            }
          else{
            Serial.println("Gesture_Up");
            return 7;
        case GES_DOWN_FLAG:
          delay(GES_ENTRY_TIME);
          paj7620ReadReg(0x43, 1, &data);
          if(data == GES_FORWARD_FLAG) {
            Serial.println("Gesture_Forward");
            delay(GES_QUIT_TIME);
            return 10;
            }
          else if(data == GES_BACKWARD_FLAG) {
            Serial.println("Gesture_Backward");
            delay(GES_QUIT_TIME);
            return 10;
            }
          else {
            Serial.println("Gesture_Down");
            return 8;
            }
          break;
        case GES_FORWARD_FLAG:
          Serial.println("Gesture_Forward");
          delay(GES_QUIT_TIME);
          return 12;
        case GES_BACKWARD_FLAG:
          Serial.println("Gesture_Backward");
          delay(GES_QUIT_TIME);
          return 13;
        default:
          return 10;
      }
}
}
```