实训例程实验指导书

## 信息采集实验

### 实验目的

掌握光照度传感器基本原理。

掌握温湿度传感器基本原理。

掌握在Contiki操作系统下的传感器驱动开发。

### 实验环境

原理机、ARM JLink 仿真器、PC 机。

软件：Windows XP/7/8/10，IAR 集成开发环境，Contiki3.0。

### 实验原理

此次实验使用的是原理机板载的BH1750光照度传感器和HTU21D温湿度传感器，原理机使用IIC读取光照度传感器和温湿度传感器数据。从本节开始，所有操作都是通过创建contiki操作系统的进程来完成的。

**BH1750光照度传感器**

BH1750FVI 是一种用于两线式串行总线接口的数字型光强度传感器集成电路。这种集成电路可以根据收集的光线强度数据来调整液晶或者键盘背景灯的亮度。利用它的高分辨率可以探测较大范围的光强度变化。（1lx-65535lx）

**产品特点**

1. 支持 I2C BUS 接口(f/s Mode Support)。

2. 接近视觉灵敏度的光谱灵敏度特性(峰值灵敏度波长典型值:560nm)。

3. 输出对应亮度的数字值。

4. 对应广泛的输入光范围(相当于 1-65535lx)。

5. 通过降低功率功能,实现低电流化。

6. 通过 50Hz/60Hz 除光噪音功能实现稳定的测定

7. 支持 1.8V 逻辑输入接口。

8. 无需其他外部件。

9. 光源依赖性弱（白炽灯，荧光灯，卤素灯，白光 LED，日光灯）。

10. 有两种可选的 I2C slave 地址。

11. 可调的测量结果影响较大的因素为光入口大小。

12. 使用这种功能能计算 1.1 lx 到 100000 lx 马克斯/分钟的范围。

13. 最小误差变动在±20%。

14. 受红外线影响很小。

**HTU21D温湿度传感器**

HTU21温湿度传感器采用基于法国Humirel高性能的湿度感应元件制成，HTU21系列模块为OEM客户应用提供了准确可靠的温湿度测量数据。HTU21不仅输出方式多样化，作电压范围比较宽，同时具有很高的温度精度和湿度精度。

HTU21系列模块专为低功耗小体积应用设计，具有良好的品质、快的响应速度、抗干扰能力强、性价比高等优点，微小的体积、极低的功耗，使HTU21成为各类应用的首选。

**基本参数**

量程： 湿度：0-100%RH；

封装：微型DFN

工作温度范围：-40～125℃

精确度：RH:+/-2%RH – HTU21；T

供电电源：1.8-3.6V

特点：微型的DFN封装;温湿度一体测量，可选择的多种输出方式；免标定；符合无铅标准，适合回流焊接；低功耗；快速响应，非常小的温度效应；抗结露特性

类型：温湿度传感器模块

电气连接：贴片式

### 实验内容

打开光盘实验例程->实训例程->信息采集工程，点开zonesion分组，主函数在contiki\_main.c文件内，内容如下

void main(void)

{

clock\_init();

NVIC\_PriorityGroupConfig(NVIC\_PriorityGroup\_2);

uart1\_init(115200);

debug\_init(uart1\_putc);

printf("\r\nStarting ");

printf(CONTIKI\_VERSION\_STRING);

printf(" on STM32F40x\r\n");

process\_start(&etimer\_process, NULL);

ctimer\_init();

#if AUTOSTART\_ENABLE

autostart\_start(autostart\_processes);

#endif

while(1)

{

do

{}

while(process\_run() > 0);

idle\_count++;

}

}

主函数首先初始化系统时钟、中断优先级、串口，向串口打印版本信息，然后启动一个etimer\_process进程，初始化操作系统提供的定时器定时事件功能，此功能非常方便，在之后的开发中会频繁使用。

接着有一个AUTOSTART\_ENABLE宏定义，宏定义在程序中定义为1，里面的函数autostart\_start用来自动启动用户定义的进程，启动自定义进程后主函数进入while（1），不断运行启动的进程。点击autostart\_start函数参数autostart\_processes，右键go to definition跳转到autoapps.c文件，可以看到这里定义了一个数组，数组中是要自动启动的进程列表。在本次实验中，定义了led、lcd、htu21d\_update和user进程。

led进程使led1周期性闪烁，指示程序运行状态，点击进程列表里的led，右键go to definition，进程内容如下。

PROCESS\_THREAD(led, ev, data)

{

static struct etimer led\_timer;

PROCESS\_BEGIN();

led\_init();

etimer\_set(&led\_timer,1000);

while (1)

{

PROCESS\_WAIT\_EVENT();//等待事件发生

if(ev == PROCESS\_EVENT\_TIMER)

{

etimer\_reset(&led\_timer); //重置定时器

led1 = !led1;

}

}

PROCESS\_END();

}

led进程使用操作系统提供的定时器函数设置了1000毫秒时间的etimer事件，进入while循环，等待事件发生，若发生事件且为定时器事件，复位定时器变量，产生新的1000ms定时器事件，接着对led1进行取反操作，继续等待定时器事件，使led1周期性闪烁。

htu21d\_update进程周期性测量并更新温度和湿度度数据，进程如下。

float htu21dValue\_t = 0; //htu21d温度值

float htu21dValue\_h = 0; //htu21d湿度值

PROCESS\_THREAD(htu21d\_update, ev, data)

{

static struct etimer htu21d\_time;

static char htu21d\_command = 0;

static char htu21d\_buf[4]={0};

PROCESS\_BEGIN();

htu21d\_init();

while(1)

{

/\*更新温度值\*/

for(uint8 i = 0;i<4;i++)

{

htu21d\_buf[i] = 0; //清htu21d\_buf

}

htu21d\_command = 0xf3;

i2c\_write(HTU21D\_ADDR, &htu21d\_command, 1);

etimer\_set(&htu21d\_time,50); //设置etimer定时器

PROCESS\_WAIT\_EVENT\_UNTIL(ev == PROCESS\_EVENT\_TIMER);

if (i2c\_read(HTU21D\_ADDR, htu21d\_buf, 2) == 2)

{

if ((htu21d\_buf[1]&0x02) == 0)

{

htu21dValue\_t = -46.85f + 175.72f \* ((htu21d\_buf[0]<<8 | htu21d\_buf[1])&0xfffc) / (1<<16);

}

}

/\*更新湿度值\*/

for(uint8 i = 0;i<4;i++)

{

htu21d\_buf[i] = 0; //清htu21d\_buf

}

htu21d\_command = 0xf5;

i2c\_write(HTU21D\_ADDR, &htu21d\_command, 1);

etimer\_set(&htu21d\_time,50); //设置etimer定时器

PROCESS\_WAIT\_EVENT\_UNTIL(ev == PROCESS\_EVENT\_TIMER);

if (i2c\_read(HTU21D\_ADDR, htu21d\_buf, 2) == 2)

{

if ((htu21d\_buf[1]&0x02) == 0x02)

{

htu21dValue\_h = -6 + 125 \* ((htu21d\_buf[0]<<8 | htu21d\_buf[1])&0xfffc) / (1<<16);

}

}

etimer\_set(&htu21d\_time,200); //设置etimer定时器

PROCESS\_WAIT\_EVENT\_UNTIL(ev == PROCESS\_EVENT\_TIMER);

}

PROCESS\_END();

}

进程先通过IIC发送读取温度的命令，然后等待温度数据，获取温度数据后，发送读取湿度命令并等待湿度数据，获取温度和湿度数据后会更新全局变量htu21dValue\_t与htu21dValue\_h的值，供其他进程使用。

User进程如下。

PROCESS\_THREAD(user, ev, data)

{

static struct etimer user\_timer;

PROCESS\_BEGIN();

sensor\_init(); //传感器初始化

etimer\_set(&user\_timer,200); //设置etimer定时器

while(1)

{

sensor\_poll(); //传感器轮询函数

//等待etimer定时器溢出事件

PROCESS\_WAIT\_EVENT\_UNTIL(ev == PROCESS\_EVENT\_TIMER);

etimer\_reset(&user\_timer); //设置etimer定时器

}

PROCESS\_END();

}

进程先调用sensor\_init()初始化使用的传感器，然后设置了一个200ms的etimer定时器，接着进入死循环调用sensor\_poll()函数，然后退出进程等待etimer定时器事件，复位定时器开始后继续调用sensor\_poll()，周期性循环，sensor\_poll()为传感器轮询函数，用来查询光照度值和RTC时间，代码如下。

float A0, A1, A2; //光照度、温度、湿度

RTC\_TimeTypeDef RTC\_TimeStruct; //时间

RTC\_DateTypeDef RTC\_DateStruct; //日期

void sensor\_poll(void)

{

RTC\_GetTime(RTC\_Format\_BIN,&RTC\_TimeStruct);

RTC\_GetDate(RTC\_Format\_BIN, &RTC\_DateStruct);

A0 = bh1750\_get(); //读取光照度值

A1 = htu21dValue\_t; //读取温度

A2 = htu21dValue\_h; //读取湿度

}

函数前面定义的A0、A1、A2分别存放光照度，温度和湿度。

bh1750\_get()为读取光照度数据函数， htu21d\_update进程更新的htu21dValue\_t和htu21dValue\_h在这里赋值给A1和A2。

温湿度传感器的每次读取需要长时间的延时等待，所以这里没有把温湿度传感器程序放在sensor\_poll()中，而是单独创建一个进程读取，这样延时的时间可以让操作系统处理其他进程，提高系统效率。

lcd进程功能是把传感器数据和RTC时间在原理机屏幕上进行显示。

### 实验步骤

（1）准备一个原理机，正确的将 J-Link 通过排线接到原理机上。将 USB 一端插入原理机上，另一端接电脑，第一次使用需要安装驱动（win7 环境下自动安装）。

（2）将产品光盘中 03-系统代码目录下的 contiki 系统源码解压拷贝到 PC 机任意目录，比如 E 盘的根目录，打开例程：将光盘中的例程“AI-Carobot\05-实验例程\02-实训例程\1-CarRobot-信息采集”整个文件夹拷贝到系统源码目录的 E:\contiki-3.0\zonesion 文件夹下，双击 contiki-3.0\ zonesion\1-CarRobot-信息采集\ide\iar 目录下的“Smartcar-stm32.eww”文件打开工程。选择 Project ->Rebuild All，编译工程。

（3）打开原理机电源开关，点击 IAR 界面上的 Download and Debug 按钮，将程序烧写到原理机中。点击全速运行 Go 按钮和 Stop Debugging 退出调试按钮，退出调试。

（4）观察实验结果。

### 实验结果

（1）程序运行后在LCD屏上显示当前环境光照度、温度、湿度参数并显示RTC时间。

（2）用手按住光照度传感器，屏幕上显示的光照值减小，手指拿开，光照度值增大。

（3）对着温湿度传感器哈气，温度值和湿度值会发生变化。

## 寻迹巡航实验

### 实验目的

掌握E18D80NK红外避障传感器基本原理。

掌握在Contiki操作系统下的传感器驱动开发。

学习在Contiki操作系统下控制电机寻迹驱动的开发。

### 实验环境

AI车型机器人实验平台、E18D80NK红外避障传感器、ARM JLink 仿真器、PC 机。

软件：Windows XP/7/8/10，IAR 集成开发环境，Contiki3.0。

### 实验原理

本节实验使用了实验平台自带的E18-D80NK红外避障传感器。E18-D80NK是一种集发射与接收于一体的光电传感器， 发射光经过调制后发出， 接收头对反射光进行解调输出。有效的避免了可见光的干扰。透镜的使用， 也使得这款传感器最远可以检测80 厘米距离的问题（由于红外光的特性，不同颜色的物体，能探测的最大距离也有不同；白色物体最远，黑色物体最近）。检测障碍物的距离可以根据要求通过尾部的电位器旋钮进行调节。

该传感器具有探测距离远、受可见光干扰小、价格便宜、易于装配、使用方便等特点，可以广泛应用于机器人避障、流水线计件等众多场合。

传感器正常情况下是高电平输出，检测到物体后红外线返回，输出低电平。通过传感器的这一特性，我们把传感器安装在车底，照射地面。传感器照射到普通地面时，红外线发生反射，传感器输出低电平，当照射到地面黑线位置，红外线被吸收，此时输出高电平。可以通过判断高低电平完成黑线的探测。

电机控制是通过控制电机驱动来驱动电机，直接给电机驱动信号端口高低电平即可控制电机启停。不过要完成寻迹功能，简单的启停控制显得有些粗糙，为了达到比较好的寻迹效果，需要比较灵活的控制。改用脉冲宽度调制技术，即PWM波进行控制

本节实验使用原理机上定时器输出四路稳定的PWM波，控制两个电机的正转、反转、停止、加速、减速等，非常灵活，拓展性强，满足实验要求。

### 实验内容

打开光盘实验例程->实训例程->寻迹巡航实验工程，点开zonesion分组，文件contiki\_main.c里的主函数不变，点击autostart\_start函数参数autostart\_processes，右键go to definition跳转到autoapps.c文件的启动进程列表，可以看到在本次实验中，新增了key、motor进程。

key进程是按键驱动，用来检测板载按键的状态，在按键按下后会产生按键事件，根据对应的按键和按下的类型把事件发送到其他进程。

PROCESS\_THREAD(key, ev, data)

{

static struct etimer key\_timer;

static unsigned char key\_status\_tmp = 0;

PROCESS\_BEGIN();

key\_adc\_init();

adc\_init();

process\_event\_key = process\_alloc\_event();

while (1)

{

//K1，K2检测

if(get\_key\_status(KEY1\_2) > 2000)

{

key\_status\_tmp = 0;

for(;;)

{

etimer\_set(&key\_timer, 10);

PROCESS\_YIELD();

if(get\_key\_status(KEY1\_2) > 2000)

key\_status\_tmp++;

else break;

}

if(key\_status\_tmp >= LONG\_TIME)

{

key\_status = KEY1\_LONG\_PRESSED;

}

else

{

key\_status = KEY1\_PRESSED;

process\_post(&user,process\_event\_key,&key\_status);

}

}

else if(get\_key\_status(KEY1\_2) > 1000)

{

key\_status\_tmp = 0;

for(;;)

{

etimer\_set(&key\_timer, 10);

PROCESS\_YIELD();

if((get\_key\_status(KEY1\_2) > 1000) && (get\_key\_status(KEY1\_2) < 2000))

key\_status\_tmp++;

else break;

}

if(key\_status\_tmp >= LONG\_TIME)

{

key\_status = KEY2\_LONG\_PRESSED;

}

else

{

key\_status = KEY2\_PRESSED;

process\_post(&user,process\_event\_key,&key\_status);

}

}

//K3，K4检测

if(get\_key\_status(KEY3\_4) > 2000)

{

key\_status\_tmp = 0;

for(;;)

{

etimer\_set(&key\_timer, 10);

PROCESS\_YIELD();

if(get\_key\_status(KEY3\_4) > 2000)

key\_status\_tmp++;

else break;

}

if(key\_status\_tmp >= LONG\_TIME)

{

key\_status = KEY3\_LONG\_PRESSED;

}

else

{

key\_status = KEY3\_PRESSED;

process\_post(&user,process\_event\_key,&key\_status);

}

}

else if(get\_key\_status(KEY3\_4) > 1000)

{

key\_status\_tmp = 0;

for(;;)

{

etimer\_set(&key\_timer, 10);

PROCESS\_YIELD();

if((get\_key\_status(KEY3\_4) > 1000) && (get\_key\_status(KEY3\_4) < 2000))

key\_status\_tmp++;

else break;

}

if(key\_status\_tmp >= LONG\_TIME)

{

key\_status = KEY4\_LONG\_PRESSED;

}

else

{

key\_status = KEY4\_PRESSED;

process\_post(&user,process\_event\_key,&key\_status);

}

}

etimer\_set(&key\_timer,20);

PROCESS\_YIELD();

}

PROCESS\_END();

}

本次程序先使用ADC检测是K1、K2是否按下，然后判断是短按还是长按，把判断结果做为按键事件的数据，将事件发送给对应进程。下面同理，先检测是K3、K4是否按下，然后判断是短按还是长按，把判断结果做为按键事件的数据，将事件发送给对应进程，这里全部发送给user进程。

user进程做了在上节基础上增加了部分代码

PROCESS\_THREAD(user, ev, data)

{

static struct etimer user\_timer;

PROCESS\_BEGIN();

sensor\_init(); //传感器初始化

etimer\_set(&user\_timer,100); //设置etimer定时器

while(1)

{

PROCESS\_WAIT\_EVENT();

if(ev == process\_event\_key) //判断是否发生按键事件

{

if((\*(int\*)data)==KEY1\_PRESSED) //判断事件传入的值

{

motor\_run = !motor\_run;

}

else if((\*(int\*)data)==KEY3\_PRESSED) //判断事件传入的值

{

if(motor\_speed<8000)

motor\_speed += 1000;

}

else if((\*(int\*)data)==KEY4\_PRESSED) //判断事件传入的值

{

if(motor\_speed>5000)

motor\_speed -= 1000;

}

}

if(ev == PROCESS\_EVENT\_TIMER)

{

etimer\_reset(&user\_timer);

sensor\_poll();

}

}

PROCESS\_END();

}

主要增加了通过按键事件对电机操作的代码，先判断是否发生按键事件，如果发生，根据按键事件传过来的数据做相应处理。如果K1按下，改变电机运行状态，即停止或启动电机，如果K3按下，增加电机速度，如果K4按下，减小电机速度。

motor进程为寻迹功能实现程序。

PROCESS\_THREAD(motor, ev, data)

{

static struct etimer motor1\_timer;

PROCESS\_BEGIN();

motor\_init(); //电机初始化

E18D80NK\_init(); //红外寻迹初始化

etimer\_set(&motor1\_timer,20); //设置etimer定时器

while (1)

{

PROCESS\_WAIT\_EVENT();

if(ev == PROCESS\_EVENT\_TIMER)

{

etimer\_reset(&motor1\_timer);

infraredSta = get\_state();

if(motor\_run)

{

if((infraredSta&0x03)==0x03)

{

led3 = 0;

led4 = 0;

motor\_control(motor\_speed,motor\_speed,STOP);//停止

}

else if((infraredSta&0x01)==0x01)

{

led3 = 1;

led4 = 0;

motor\_control(motor\_speed,motor\_speed,LEFT);//左转

}

else if((infraredSta&0x02)==0x02)

{

led3 = 0;

led4 = 1;

motor\_control(motor\_speed,motor\_speed,RIGHT);//右转

}

else

{

led3 = 1;

led4 = 1;

motor\_control(motor\_speed,motor\_speed,FRONT);//前进

}

}

}

}

PROCESS\_END();

}

进程首先进行电机初始化，这里是对原理机上STM32F407的定时器4进行PWM模式初始化，产生4路10KHz的PWM波。接着初始化E18D80NK红外避障传感器，初始化完成后设定一个20ms的etimer事件，在接下来的死循环不断触发此事件，周期性读取红外传感器检测的状态，根据状态判断地面黑线位置，控制小车的运行，以此完成寻迹功能。在此基础上增加了led2和led3对传感器输出结果的指示。

Lcd进程在屏幕右上角增加了电机运行状态显示，屏幕下方增加了红外传感器检测状态显示以及电机速度显示。

### 实验步骤

（1）准备一个原理机，正确的将 J-Link 通过排线接到原理机上。将 USB 一端插入原理机上，另一端接电脑，第一次使用需要安装驱动（win7 环境下自动安装）。

（2）将产品光盘中 03-系统代码目录下的 contiki 系统源码解压拷贝到 PC 机任意目录，比如 E 盘的根目录，打开例程：将光盘中的例程“AI-Carobot\05-实验例程\02-实训例程\2-CarRobot-寻迹巡航”整个文件夹拷贝到系统源码目录的 E:\contiki-3.0\zonesion 文件夹下，双击 contiki-3.0\ zonesion\2-CarRobot-寻迹巡航\ide\iar 目录下的“Smartcar-stm32.eww”文件打开工程。选择 Project ->Rebuild All，编译工程。

（3）打开原理机电源开关，点击 IAR 界面上的 Download and Debug 按钮，将程序烧写到原理机中。点击全速运行 Go 按钮和 Stop Debugging 退出调试按钮，退出调试。

（4）准备一块带有黑线的场地，将小车放在黑线上方 。

（5）按下K1键将屏幕右上角调到“运行中”，观察实验结果。

（6）按下K3或K4键来增加或减小电机速度，观察实验结果。

### 实验结果

（1）按下K1键开始运行后，小车会沿着地面上的黑线运行，完成寻迹功能。

（2）按下K3键小车寻迹时前进速度增加。

（3）按下K3键小车寻迹时前进速度减小。

## 智能避障实验

### 实验目的

掌握超声波传感器基本原理。

掌握在Contiki操作系统下的超声波传感器驱动开发。

掌握在Contiki操作系统下控制电机避障驱动的开发。

### 实验环境

AI车型机器人实验平台、HC-SR04超声波测距模块、ARM JLink 仿真器、PC 机。

软件：Windows XP/7/8/10，IAR 集成开发环境，Contiki3.0。

### 实验原理

超声波测距模块是用来测量距离的一种产品，通过发送和就接收超声波，利用时间差和声音传播速度，计算出模块到前方障碍物的距离。本节实验使用的是HC-SR04超声波测距模块。

**产品特点：**

HC-SR04 超声波测距模块可提供2cm-400cm 的非接触式距离感测功能， 测距精度可达高到3mm；模块包括超声波发射器、接收器与控制电路。

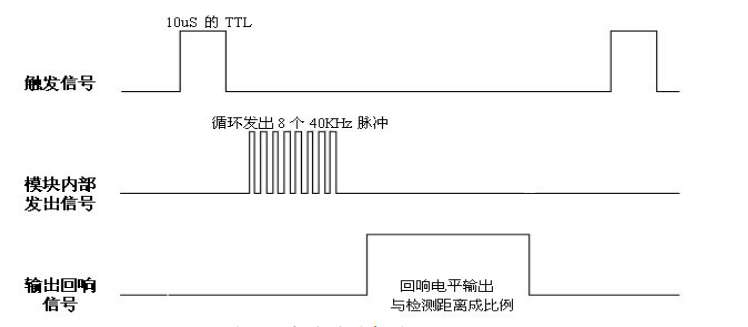
**基本工作原理：**

(1) 采用IO 口TRIG触发测距，给最少10us 的高电平信呈。

(2) 模块自动发送8 个40khz 的方波，自动检测是否有信号返回；

(3) 有信号返回，通过IO 口ECHO输出一个高电平，高电平持续的时间就是超声波从发射到返回的时间。测试距离=( 高电平时间\*声速(340M/S))/2;

**超神波时序图：**



可以看到超声波测距模块的使用，分为触发和接收两部分，一般直接使用IO口控制，但这样会有一个问题，就是在触发超声波测距之后，等待超声波返回的这段时间是不确定的，且这段时间在几毫秒到几十毫秒之间，在这段时间程序要一直检测IO口的电平，占用了CPU大量时间，会严重影响其他程序的运行，所以本次实验我们不使用IO口控制，使用原理机主控自带的TIM输入捕获功能实现超声波测距。

### 实验内容

打开光盘实验例程->实训例程->智能避障实验工程，点开zonesion分组，文件contiki\_main.c里的主函数不变，点击autostart\_start函数参数autostart\_processes，右键go to definition跳转到autoapps.c文件的启动进程列表，可以看到在本次实验中，新增了ultrasonic进程。

点击ultrasonic右键go to definition跳转到进程函数。

PROCESS\_THREAD(ultrasonic, ev, data)

{

static struct etimer ultrasonic\_timer;

PROCESS\_BEGIN();

Ultrasonic\_init();

etimer\_set(&ultrasonic\_timer,50); //设置etimer定时器

while (1)

{

PROCESS\_WAIT\_EVENT();

if(ev == PROCESS\_EVENT\_TIMER)

{

etimer\_reset(&ultrasonic\_timer); //重置etimer定时器

trigger\_ultrasonic(); //触发一次测距

}

}

PROCESS\_END();

}

进程先初始化超声波，然后设置一个50ms的etimer事件，接着进入死循环，等待etimer事件，周期性运行trigger\_ultrasonic()函数，根据注释此函数是触发一次测距，并没有看到接收超声波的函数。前面我们说过，由于时效性问题，本次实验不使用IO口控制方法，使用的是原理机主控自带的输入捕获功能实现超声波测距，我们go to到Ultrasonic\_init()函数。

void Ultrasonic\_init(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure;

RCC\_AHB1PeriphClockCmd(RCC\_trig, ENABLE); //使能GPIOB时钟

GPIO\_InitStructure.GPIO\_Pin = trig\_pin;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_OUT; //输出模式

GPIO\_InitStructure.GPIO\_OType = GPIO\_OType\_PP; //推挽输出

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz; //输出速度50MHz

GPIO\_InitStructure.GPIO\_PuPd = GPIO\_PuPd\_DOWN; //下拉

GPIO\_Init(trig\_port, &GPIO\_InitStructure); //根据上述参数配置GPIO

GPIO\_ResetBits(trig\_port, trig\_pin);

GPIO\_InitStructure.GPIO\_Pin = echo\_pin;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_IN; //输入模式

GPIO\_InitStructure.GPIO\_OType = GPIO\_OType\_PP; //推挽

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz; //输出速度50MHz

GPIO\_InitStructure.GPIO\_PuPd = GPIO\_PuPd\_DOWN; //下拉

GPIO\_Init(echo\_port, &GPIO\_InitStructure); //根据上述参数配置GPIO

/\*10us计数，定时30ms\*/

TIM12CH2\_Cap\_Init(3000,839);

}

函数前面是对输入捕获的IO口进行初始化，最后初始化了定时器12通道2输入捕获配置，为了避免出现超声波无返回导致程序一直检测的情况，这里设定了一个30ms的溢出时间，超过此时间就不再等待超声波返回，定时器12的中断程序如下。

void TIM8\_BRK\_TIM12\_IRQHandler(void)

{

if(TIM\_GetITStatus(TIM12, TIM\_IT\_Update) != RESET) //溢出

{

TIM\_Cmd(TIM12,DISABLE ); //关闭定时器

TIM\_OC2PolarityConfig(TIM12,TIM\_ICPolarity\_Rising);//CC1P=0 设置为上升沿捕获

if(TIM12CH2\_CAPTURE\_STA&0x01) //检测到信号，但距离太远

{

ultrasonicValue = 510;

}

else //没有信号

{

ultrasonicValue = 0;

}

}

if(TIM\_GetITStatus(TIM12, TIM\_IT\_CC2) != RESET) //发生捕获事件2

{

if(TIM12CH2\_CAPTURE\_STA&0x01) //捕获到一个上升沿

{

TIM\_Cmd(TIM12,DISABLE ); //关闭定时器

TIM12CH2\_CAPTURE\_STA = 0x02; //标记成功捕获到一次高电平脉宽

TIM12CH2\_CAPTURE\_VAL=TIM\_GetCapture2(TIM12); //获取当前的捕获值.

TIM\_OC2PolarityConfig(TIM12,TIM\_ICPolarity\_Rising);//CC1P=0 设置为上升沿捕获

ultrasonicValue = (TIM12CH2\_CAPTURE\_VAL \* 0.17);//计算距离

}

else //还未开始,第一次捕获上升沿

{

TIM12CH2\_CAPTURE\_STA = 0x01; //标记捕获到了上升沿

TIM\_Cmd(TIM12,DISABLE ); //关闭定时器

TIM\_SetCounter(TIM12,0);

TIM\_OC2PolarityConfig(TIM12,TIM\_ICPolarity\_Falling);//CC1P=1 设置为下降沿捕获

TIM\_Cmd(TIM12,ENABLE ); //使能定时器

}

}

TIM\_ClearITPendingBit(TIM12, TIM\_IT\_CC2|TIM\_IT\_Update); //清除中断标志位

}

定时器12产生溢出事件和捕获事件都会进入此中断，在中断中，若发生定时器溢出，根据是否捕获到高电平更新超声波测量距离，510mm是30ms内超声波能测量的最大值。若发生捕获事件，根据捕获的电平类型计算高电平的时间，然后计算超声波测量距离。

所以在ultrasonic进程中，只是周期性触发一次测距，超声波的接收与距离计算都在定时器12的中断中进行。

Motor进程实现避障功能，程序比较简单。

PROCESS\_THREAD(motor, ev, data)

{

static struct etimer motor1\_timer;

PROCESS\_BEGIN();

motor\_init(); //电机初始化

E18D80NK\_init(); //红外寻迹初始化

etimer\_set(&motor1\_timer,20); //设置etimer定时器

while (1)

{

PROCESS\_WAIT\_EVENT();

if(ev == PROCESS\_EVENT\_TIMER)

{

etimer\_reset(&motor1\_timer);

if(motor\_run)

{

if(ultrasonicValue<30)

{

motor\_control(0,0,0);

}

else

{

motor\_control(motor\_speed,motor\_speed,FRONT);

}

}

}

}

PROCESS\_END();

}

进程会周期性判断超声波探测的距离，若距离小于30mm，则判定前方有障碍物，小车停止，若距离大于30mm，则判定前方无障碍物，小车前进。

Lcd进程增加了超声波距离显示。

### 实验步骤

（1）准备一个原理机，正确的将 J-Link 通过排线接到原理机上。将 USB 一端插入原理机上，另一端接电脑，第一次使用需要安装驱动（win7 环境下自动安装）。

（2）将产品光盘中 03-系统代码目录下的 contiki 系统源码解压拷贝到 PC 机任意目录，比如 E 盘的根目录，打开例程：将光盘中的例程“AI-Carobot\05-实验例程\02-实训例程\3-CarRobot-智能避障”整个文件夹拷贝到系统源码目录的 E:\contiki-3.0\zonesion 文件夹下，双击 contiki-3.0\ zonesion\3-CarRobot-智能避障\ide\iar 目录下的“Smartcar-stm32.eww”文件打开工程。选择 Project ->Rebuild All，编译工程。

（3）打开原理机电源开关，点击 IAR 界面上的 Download and Debug 按钮，将程序烧写到原理机中。点击全速运行 Go 按钮和 Stop Debugging 退出调试按钮，退出调试。

（4）将小车放在空旷位置按下K1键将屏幕右上角调到“运行中”，观察实验结果。

（5）在小车前方放置障碍物，观察实验结果。

（6）按下K3或K4键来增加或减小电机速度，观察实验结果。

### 实验结果

（1）按下K1键开始运行后，小车会向前运行。

（2）在小车前方放置障碍物，小车距离障碍物小于30mm后，小车会停止前进。

（3）按下K3键小车前进时前进速度增加。

（4）按下K3键小车前进时前进速度减小。

## 无线控制实验

### 实验目的

掌握在Contiki操作系统下的传感器驱动开发。

掌握原理机与CC2540串口通信。

学习ZXBee通信协议。

学习原理机与APP之间ZXBee通信协议驱动的开发。

### 实验环境

原理机、ARM JLink 仿真器、CC2540、SmartRF 、PC 机，Android手机。

软件：Windows XP/7/8/10，IAR 集成开发环境，Contiki3.0、AI车型机器人APP。

### 实验原理

本节实验在信息采集实验的基础上，通过CC2540蓝牙模块与手机蓝牙建立数据传送链接，原理机以CC2540为透传工具，使用ZXBee通信协议把传感器数据上报到手机端，手机端配合我们开发的APP可显示数据内容，也可以使用其他蓝牙工具查看收到的数据，建议使用我们配套的APP查看，方便直观。

### 实验内容

打开光盘实验例程->实训例程->无线控制实验工程，点开zonesion分组，文件contiki\_main.c里的主函数不变，点击autostart\_start函数参数autostart\_processes，右键go to definition跳转到autoapps.c文件的启动进程列表，可以看到在本次实验中，新增了config、gsm、utcp、ble\_recv、qrcode进程。

config进程很简单，首先对W25Q64进行初始化，然后初始化系统配置，主要是初始化连接模式，逻辑如下：

读取特定位置标志位，若系统是第一次开机，保存当前连接模式。若系统不是第一次开机，则读取上次关机是保存的连接模式，程序如下。

void config\_init(void)

{

FlaManageInit();

int flag = 0;

FlaVPageRead(1, 0, &flag, 4);

if (flag != 0x5a5a5a5c)

{

FlaFormat();

flag = 0x5a5a5a5c;

FlaVPageWrite(1, 0, &flag, 4);

config\_save\_mode();

}

else

{

config\_read\_mode();

}

}

gsm和utcp进程是为实现GSM连接的驱动，程序如下

PROCESS\_THREAD(gsm, ev, data)

{

static struct etimer et;

PROCESS\_BEGIN(); //开始进程

if(connect\_mode == 1)

{

mg2639\_init(); //gsm模块初始化

event\_at\_finish = process\_alloc\_event();

etimer\_set(&et, CLOCK\_SECOND/10); // 100 ms

PROCESS\_WAIT\_EVENT\_UNTIL(ev == PROCESS\_EVENT\_TIMER); //等待事件触发

mytimer\_set(&gsm\_timer, \_gsm\_timer\_call, NULL);

mytimer\_add(&gsm\_timer, 1000);

ctimer\_set(&gsm\_timer, 1000, \_gsm\_timer\_call, NULL); //设置定时器

shell\_register\_command(&gsm\_command); //登录指令

while (1)

{

\_poll\_request(); //轮询请求

\_poll\_response(); //轮询响应

etimer\_set(&et, CLOCK\_SECOND/100); //定时器设置

PROCESS\_YIELD(); //切换进程

}

}

PROCESS\_END(); //结束进程

}

ble\_recv进程为蓝牙连接驱动，程序如下

PROCESS\_THREAD(ble\_recv, ev, data)

{

PROCESS\_BEGIN();

evt\_data = process\_alloc\_event();

ble\_init();

while (1)

{

PROCESS\_WAIT\_EVENT();

if (ev == evt\_data)

{

char \*p = data;

printf("ble >>> %s\r\n", p);

zxbeeBegin();

zxbee\_onrecv\_fun(p, strlen(p));

free(p);

char \*x = zxbeeEnd();

if (x)

{

ble\_write(x);

}

}

}

PROCESS\_END();

}

在串口接收中断回调函数里收到完整的{}之后会设置一个evt\_data事件，ble\_recv进程检测到此事件后调用zxbee\_onrecv\_fun函数，对{}中的内容进行解析并做相应处理，使用蓝牙连接后数据收发就是在这里进行的。

qrcode进程提供二维码编码显示功能，此进程要和key进程配合运行，key进程检测到按键按下后向qrcode进程发送按键事件，qrcode进程检测到按键事件后，根据当前连接模式在LCD屏上进行二维码显示与退出二维码显示操作，进程如下。

PROCESS\_THREAD(qrcode, ev, data)

{

PROCESS\_BEGIN();

while (1)

{

PROCESS\_WAIT\_EVENT(); //等待事件发生

if(ev == process\_event\_key) //判断是否发生按键事件

{

if((\*(int\*)data)==KEY1\_LONG\_PRESSED) //判断事件传入的值

{

qrcodeShow\_flag = 1;

if(connect\_mode==1)

DISPLAY\_RENCODE\_TO\_TFT((unsigned char \*)gsm\_info.imei);

else

DISPLAY\_RENCODE\_TO\_TFT((unsigned char \*)RADIO\_MAC);

}

if((\*(int\*)data)==KEY2\_PRESSED) //判断事件传入的值

{

qrcodeShow\_flag = 0;

lcd\_page1(1);

lcd\_page2();

}

}

}

PROCESS\_END();

}

在lcd进程新增了D1的信息显示。

User进程如下。

PROCESS\_THREAD(user, ev, data)

{

static struct etimer user\_timer;

PROCESS\_BEGIN();

sensor\_init(); //传感器初始化

etimer\_set(&user\_timer,100); //设置etimer定时器

while(1)

{

PROCESS\_WAIT\_EVENT();

if(ev == process\_event\_key) //判断是否发生按键事件

{

if((\*(int\*)data)==KEY1\_PRESSED)//事件传入的值

{

if(qrcodeShow\_flag==0)

{

connect\_mode = !connect\_mode;

if(connect\_mode==0) //蓝牙连接模式

{

ble\_init();

config\_save\_mode();

}

else //GSM连接模式

{

mg2639\_init();

config\_save\_mode();

}

}

}

}

if(ev == PROCESS\_EVENT\_TIMER)

{

etimer\_reset(&user\_timer);

sensor\_poll();

}

}

PROCESS\_END();

}

先初始化相关传感器，设置一个100ms的etimer事件，进入死循环，判断是否产生事件，若发生K1按下的按键事件， 改变网络连接模式，然后保存当前连接模式到flash里，供下次开机使用。若发生etimer事件，进入sensor\_poll()函数轮询传感器数据。

### 实验步骤

（1）准备一个原理机，正确的将 J-Link 通过排线接到原理机上。将 USB 一端插入原理机上，另一端接电脑，第一次使用需要安装驱动（win7 环境下自动安装）。

（2）将产品光盘中 03-系统代码目录下的 contiki 系统源码解压拷贝到 PC 机任意目录，比如 E 盘的根目录，打开例程：将光盘中的例程“AI-Carobot\05-实验例程\02-实训例程\4-CarRobot-无线控制实验”整个文件夹拷贝到系统源码目录的 E:\contiki-3.0\zonesion 文件夹下，双击 contiki-3.0\ zonesion\4-CarRobot-无线控制实验\ide\iar 目录下的“Smartcar-stm32.eww”文件打开工程。选择 Project ->Rebuild All，编译工程。

（3）打开原理机电源开关，点击 IAR 界面上的 Download and Debug 按钮，将程序烧写到原理机中。点击全速运行 Go 按钮和 Stop Debugging 退出调试按钮，退出调试。

（4）打开“AI-Carobot\02-出厂镜像\安卓APP 目录”找到“AI车型机器人.apk”文件，将此apk安装到手机上，APP与原理机的连接以及使用参考“AI车型机器人产品手册”。

（5）用APP发送led控制命令，观察实验现象。

### 实验结果

连接成功后，APP端可以看到显示出传感器采集的数据，使用APP发送led控制命令后led点亮或者熄灭。

## 地图定位实验

### 实验目的

掌握在Contiki操作系统下的传感器驱动开发。

掌握原理机与CC2540串口通信。

掌握Contiki操作系统下S1315R-GPS模块驱动开发。

掌握ZXBee通信协议。

掌握原理机与APP之间ZXBee通信协议驱动的开发。

### 实验环境

原理机、ARM JLink 仿真器、CC2540、SmartRF 、PC 机，Android手机。

软件：Windows XP/7/8/10，IAR 集成开发环境，Contiki3.0、AI车型机器人APP。

### 实验原理

S1315R是一个小体积的GPS模块解决方案，适用于广泛的OEM产品，满足快速、简单的系统集成和最小的开发需要。用户只需要提供3.0V ~3.6V的直流电源。

S1315R采用65通道GPS接收机，首次修复时间快，改善了-148dBm冷启动灵敏度。优越的冷启动灵敏度使它能够在困难的弱信号环境中自动获取、跟踪和获取位置定位。接收机s165dBm跟踪灵敏度允许在几乎所有的应用环境中进行连续的位置覆盖。高性能搜索引擎能够每秒测试8,000,000个时间频率假设，提供行业领先的信号采集和TTFF速度。

S1315R将以标准的NMEA-0183协议格式输出定位信息。

NMEA 0183是美国国家海洋电子协会（National Marine Electronics Association ）为海用电子设备制定的标准格式。目前业已成了GPS导航设备统一的RTCM（Radio Technical Commission for Maritime services）标准协议。

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **命令** | **说明** | **最大帧长** |
| 1 | $GPGGA | 全球定位数据 | 72 |
| 2 | $GPGSA | 卫星PRN数据 | 65 |
| 3 | $GPGSV | 卫星状态信息 | 210 |
| 4 | $GPRMC | 运输定位数据 | 70 |
| 5 | $GPVTG | 地面速度信息 | 34 |
| 6 | $GPGLL | 大地坐标信息 |  |
| 7 | $GPZDA | UTC时间和日期 |  |

注：发送次序$GPZDA、$GPGGA、$GPGLL、$GPVTG、$GPGSA、$GPGSV\*3、$GPRMC。

本次实验使用的是$GNGGA全球定位数据，在串口接收中断接收GPS模块数据，判断是否出现$GNGGA帧头，接收到此帧头后把后面附带的数据接受完成并解析，根据解析的数据转换为GPS坐标。

原理机按照ZXBee格式发送给自带的CC2540进行蓝牙透传，上位机收到坐标数据后在地图上显示位置。

### 实验内容

打开光盘实验例程->实训例程->地图定位实验工程，点开zonesion分组，文件contiki\_main.c里的主函数不变，点击autostart\_start函数参数autostart\_processes，右键go to definition跳转到autoapps.c文件的启动进程列表，可以看到在本次实验中，进程列表与上一节一样。本次实验实是在上一节实验的基础上增加了部分功能，我们跳转到user进程。

PROCESS\_THREAD(user, ev, data)

{

static struct etimer user\_timer;

PROCESS\_BEGIN();

sensor\_init(); //传感器初始化

etimer\_set(&user\_timer,100); //设置etimer定时器

while(1)

{

PROCESS\_WAIT\_EVENT();

if(ev == process\_event\_key) //判断是否发生按键事件

{

if((\*(int\*)data)==KEY1\_PRESSED)//事件传入的值

{

if(qrcodeShow\_flag==0)

{

connect\_mode = !connect\_mode;

if(connect\_mode==0) //蓝牙连接模式

{

ble\_init();

config\_save\_mode();

}

else //GSM连接模式

{

mg2639\_init();

config\_save\_mode();

}

}

}

}

if(ev == PROCESS\_EVENT\_TIMER)

{

etimer\_reset(&user\_timer);

sensor\_poll();

}

}

PROCESS\_END();

}

User进程和上一节完全一样，不同之处在于sensor\_init()和sensor\_poll()函数， sensor\_init()函数增加了GPS传感器的初始化，跳转到sensor\_poll()函数。

void sensor\_poll(void)

{

char buf[64];

static int t = 0;

RTC\_GetTime(RTC\_Format\_BIN,&RTC\_TimeStruct);

RTC\_GetDate(RTC\_Format\_BIN, &RTC\_DateStruct);

gps\_get(lat,lng);

A0 = bh1750\_get(); //读取光照度值

A1 = htu21dValue\_t; //读取温度

A2 = htu21dValue\_h; //读取湿度

A3 = atoi(lat)/100+(atoi(lat)%100)/60.0+((long)(atof(lat)\*100)%100)/3600.0;

A4 = atoi(lng)/100+(atoi(lng)%100)/60.0+((long)(atof(lng)\*100)%100)/3600.0;

if (V0 != 0)

{

if(t >= (V0\*10))

{

t = 0;

if(D0)

{

if(connect\_mode==1) //GSM模式

{

zxbeeBegin();

sprintf(buf, "%.1f", A0);

zxbeeAdd("A0", buf);

sprintf(buf, "%.1f", A1);

zxbeeAdd("A1", buf);

sprintf(buf, "%.1f", A2);

zxbeeAdd("A2", buf);

sprintf(buf, "%.4f", A3);

zxbeeAdd("A3", buf);

sprintf(buf, "%.4f", A4);

zxbeeAdd("A4", buf);

char \*p = zxbeeEnd();

if (p != NULL)

{

package\_data(p);

}

}

else //蓝牙模式

{

zxbeeBegin();

sprintf(buf, "%.1f", A0);

zxbeeAdd("A0", buf);

sprintf(buf, "%.1f", A1);

zxbeeAdd("A1", buf);

sprintf(buf, "%.1f", A2);

zxbeeAdd("A2", buf);

sprintf(buf, "%.4f", A3);

zxbeeAdd("A3", buf);

sprintf(buf, "%.4f", A4);

zxbeeAdd("A4", buf);

char \*p = zxbeeEnd();

if (p != NULL)

{

ble\_write(p); //通过串口蓝牙将打包的数据发送控制软件

}

}

}

}

else

t++;

}

}

可以看到函数前面增加了获取GPS相关数据的程序。后面将获取的GPS一起上传。跳转到gps\_get（）函数。

int gps\_get(char \*lat, char \*lng)

{

if (lat != NULL) strcpy(lat, \_lat);

if (lng != NULL) strcpy(lng, \_lng);

if (\_gps\_status == '1' || \_gps\_status == '2')

{

return 1;

}

return 0;

}

我们发现lat和lng分别来自\_lat和\_lng，继续追踪\_lat和\_lng到gps\_recv\_ch（）函数，此函数是串口2的接收回调函数。

static int gps\_recv\_ch(char ch)

{

static char f\_idx = 0;

static char tag[128];

if (f\_idx == 0) {

tag[0] = tag[1];

tag[1] = tag[2];

tag[2] = tag[3];

tag[3] = tag[4];

tag[4] = tag[5];

tag[5] = ch;

if (memcmp(tag, "$GNGGA", 6) == 0) {

f\_idx = 6;

return 0;

}

}

if (f\_idx) {

tag[f\_idx++] = ch;

if (tag[f\_idx-2]=='\r' && tag[f\_idx-1]=='\n') {

tag[f\_idx] = '\0';

char \*p = tag, \*pt;

p = next(p, &pt); //got $GPGGA

p = next(p, &pt); //utc time

p = next(p, &pt); //lat ddmm.mmmm

sprintf(\_lat, pt);

p = next(p, &pt); //N/S

if (\*pt=='S') {

char buf[16];

sprintf(buf, "-%s", \_lat);

sprintf(\_lat, buf);

}

p = next(p, &pt); //lng dddmm.mmmm

sprintf(\_lng, pt);

p = next(p, &pt); //E/W

if (\*pt == 'W') {

char buf[16];

sprintf(buf, "-%s", \_lng);

sprintf(\_lng, buf);

}

p = next(p, &pt); //st

\_gps\_status = \*pt;

f\_idx = 0;

}

}

return 0;

}

函数前面使用了一个数组队列，不断接收GPS模块NMEA 0183格式的数据，当接收到$GNGGA帧头，下面的数据便是我们需要的全球定位数据坐标，将坐标解析出来存放到\_lat和\_lng中，至此，我们便获取到GPS坐标数据。

### 实验步骤

（1）准备一个原理机，正确的将 J-Link 通过排线接到原理机上。将 USB 一端插入原理机上，另一端接电脑，第一次使用需要安装驱动（win7 环境下自动安装）。

（2）将产品光盘中 03-系统代码目录下的 contiki 系统源码解压拷贝到 PC 机任意目录，比如 E 盘的根目录，打开例程：将光盘中的例程“AI-Carobot\05-实验例程\02-实训例程\5-CarRobot-地图定位”整个文件夹拷贝到系统源码目录的 E:\contiki-3.0\zonesion 文件夹下，双击 contiki-3.0\ zonesion\5-CarRobot-地图定位\ide\iar 目录下的“Smartcar-stm32.eww”文件打开工程。选择 Project ->Rebuild All，编译工程。

（3）打开原理机电源开关，点击 IAR 界面上的 Download and Debug 按钮，将程序烧写到原理机中。点击全速运行 Go 按钮和 Stop Debugging 退出调试按钮，退出调试。

（4）将实验平台带到室外空旷地带，安装GPS天线。

（5）打开AI车型机器人APP，进入地图定位实验，连接到原理机上的蓝牙，观察实验现象。

### 实验结果

APP连接成功后，等待GPS数据到来，在APP上能看到在地图上显示的位置。

## 机械臂控制实验

### 实验目的

掌握舵机控制基本原理。

掌握contiki操作系统下舵机控制驱动的开发。

掌握原理机与APP之间ZXBee通信协议的使用。

### 实验环境

AI车型机器人实验平台、ARM JLink 仿真器、CC2540、SmartRF 、PC 机，Android手机。

软件：Windows XP/7/8/10，IAR 集成开发环境，Contiki3.0、AI车型机器人APP。

### 实验原理

AI车型机器人机械臂运动是依靠舵机驱动的，我们要控制机械臂的运动也是通过控制舵机来实现的。

舵机，是指在自动驾驶仪中操纵飞机舵面（操纵面）转动的一种执行部件。分有：①电动舵机，由电动机、传动部件和离合器组成。接受自动驾驶仪的指令信号而工作，当人工驾驶飞机时，由于离合器保持脱开而传动部件不发生作用。②液压舵机，由液压作动器和旁通活门组成。当人工驾驶飞机时，旁通活门打开，由于作动器活塞两边的液压互相连通而不妨害人工操纵。此外，还有电动液压舵机，简称“电液舵机”。

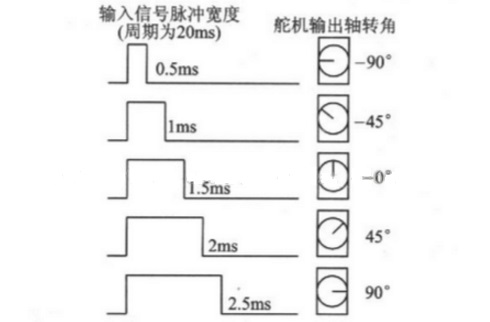
舵机是一种位置（角度）伺服的驱动器，适用于那些需要角度不断变化并可以保持的控制系统。目前，在高档遥控玩具，如飞机、潜艇模型，遥控机器人中已经得到了普遍应用。

**舵机的工作原理：**

由pwm波进入内部电路产生一个偏置电压，触发电机通过减速齿轮带动电位器移动，使电压差为零时，电机停转，从而达到伺服的效果。舵机PWM的协议都是相同的，但最新出现的舵机可能不一样。

**协议一般为：**

高电平宽度在0.5ms~2.5ms控制舵机转过不同的角度例如某一型号如下：



这样我们确定一个脉冲周期，通过控制高电平的时间来控制舵机旋转角度，本次实验选用的是200Hz的频率，脉冲周期为5ms。

为了方便控制，我们不直接控制高电平的时间，而是通过输入控制角度，程序再根据角度计算出高电平的时间。

### 实验内容

打开光盘实验例程->实训例程->地图定位实验工程，点开zonesion分组，文件contiki\_main.c里的主函数不变，点击autostart\_start函数参数autostart\_processes，右键go to definition跳转到autoapps.c文件的启动进程列表，可以看到在本次实验中，可以看到在本次实验中，新增了robot\_arm进程。右键go to definition跳转到SteeringEngine.c文件，进程如下。

PROCESS\_THREAD(robot\_arm, ev, data)

{

static struct etimer robotArm\_timer;

PROCESS\_BEGIN();

steeringEngine\_init();

etimer\_set(&robotArm\_timer,50); //设置etimer定时器

while(1)

{

PROCESS\_WAIT\_EVENT\_UNTIL(ev == PROCESS\_EVENT\_TIMER);

etimer\_reset(&robotArm\_timer);

steer1\_control(V1);

steer2\_control(V2);

steer3\_control(V3);

steer4\_control(V4);

}

PROCESS\_END();

}

先调用初始化舵机函数，然后设置一个50ms的etimer事件，进入死循环，通过etimer事件周期性运行4个舵机的控制函数。

void steer1\_control(float angle)

{

angle = 180-angle;

if(angle<20) angle = 20;

if(angle>180) angle = 180;

steer1Pwm = 500+(u16)(angle\*11.11);

}

这是舵机1的控制函数，函数首先是转换实际角度，然后限定角度范围，然后根据角度计算出PWM波高电平的时间，其他舵机控制函数类似。

### 实验步骤

（1）准备一个原理机，正确的将 J-Link 通过排线接到原理机上。将 USB 一端插入原理机上，另一端接电脑，第一次使用需要安装驱动（win7 环境下自动安装）。

（2）将产品光盘中 03-系统代码目录下的 contiki 系统源码解压拷贝到 PC 机任意目录，比如 E 盘的根目录，打开例程：将光盘中的例程“AI-Carobot\05-实验例程\02-实训例程\6-CarRobot-机械臂控制”整个文件夹拷贝到系统源码目录的 E:\contiki-3.0\zonesion 文件夹下，双击 contiki-3.0\ zonesion\6-CarRobot-机械臂控制\ide\iar 目录下的“Smartcar-stm32.eww”文件打开工程。选择 Project ->Rebuild All，编译工程。

（3）打开原理机电源开关，点击 IAR 界面上的 Download and Debug 按钮，将程序烧写到原理机中。点击全速运行 Go 按钮和 Stop Debugging 退出调试按钮，退出调试。

（4）打开APP，将手机连接到实验平台，进入机械臂控制实验，分别拖动机械臂角度控制的4个滑条，观察实验现象。

### 实验结果

拖动滑条时，机械臂上对应舵机角度变换，滑条角度越大，舵机角度变化越大。