쿼드콥터 모터 제어를 위한 임베디드 리눅스 시스템

임성락*, 김현기* 손태영* 김능환*
*호서대학교 컴퓨터 공학과
e-mail : srrim@hoseo.edu

An Embedded Linux System for Controlling of Quadcopter Motor

Seong-Rak Rim*, Hyun-Ki Kim*, Tae-Yeong Son*, Nung-Hwan Kim*
*Dept. of Computer Engineering, Hoseo University

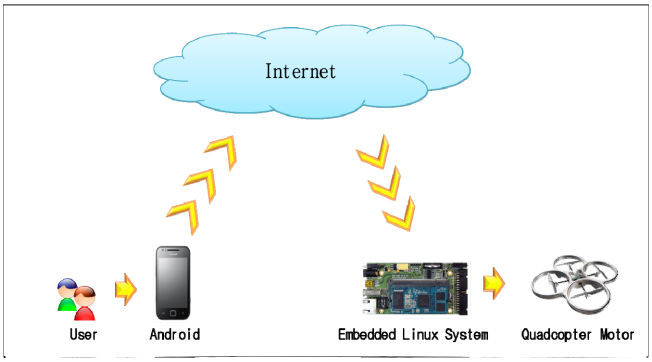
요 약

본 논문에서는 안드로이드 폰을 이용하여 쿼드콥터의 모터를 제어하기 위한 임베디드 리눅스 시스템을 제시한다. 제시한 시스템은 메시지 수신 및 파싱 모듈과 모터 제어 모듈로 나누어 설계한다. 제시한 시스템의 기능적 타당성을 검토하기 위하여 각각의 모듈을 구현하고, ARM 프로세서 기반의 임베디드 보드가 탑재된 실험용 쿼드콥터를 이용하여 실험을 하였다.

1. 서론

최근 무인 항공기의 임무수행 능력이 비약적으로 증가하고 있다. 특히, 무인 항공기의 임무수행 영역이 민간 비행공역으로 확대되고 있으며 저고도에서의 임무수행 횟수가 증가하고 있을 뿐만 아니라 동시에 여러 기체를 운용하는 임무수행 요구가 증가하고 있다[1]. MIT 공대의 Robotics 클럽은 리눅스 기반의 쿼드콥터를 개발하여 다양한 방면에서 리눅스 기반의 무인기 개발을 시도하고 있다[2]. 쿼드콥터를 제어하기 위한 여러가지 방법이 있다. 이 중 모바일 클라우드 서비스[3]를 제공하기 위하여 안드로이드 폰을 사용한다.

본 논문에서는 (그림 1)과 같이 안드로이드 폰을 이용하여 무인 항공기 쿼드콥터의 모터를 제어하기 위한 임베디드 리눅스 시스템을 제시한다.



(그림 1) 전체 시스템

(그림 1)에서 임베디드 리눅스 시스템은 기본적으로 안드로이 폰으로부터 모터의 채널과 속도로 구성된 JSON 포맷[4]으로 인코딩된 메시지를 인터넷으로부터 수신하여 해당 모터의 속도를 제어하는 기능이 요구된다. 이를 위하여 본 논문에서 JSON 포맷의 메시지 수신 및 파싱 부분과 물리적인 모터 제어를 담당하는 디바이스 드라이버 부분으로 구분하여 설계한다.

2. 설계

2.1 메시지 수신 및 파싱

인터넷으로부터의 JSON 메시지 수신은 UDP 프로토콜과 Poll 메커니즘을 이용하여 (그림 2)와 같이 설계한다.

```
sd = soket(PF_INET, SOCK_DGRAM, 0 );                    // 소켓 생성

// IP와 PORT 등록
serv_addr.sin_family      = AF_INET;
serv_addr.sin_port        = htons( port );
serv_addr.sin_addr.s_addr  = htonl( INADDR_ANY );

bind( sd, (struct sockaddr*) serv _addr, sizeof(serv_addr); //소켓 정보 커널에 등록

pds[0].fd    = sd;                                      // 디스크립터 지정
pds[0].events = POLLIN;                                 // POLLIN 설정

event = poll(event, event_count, timeout );             // 이벤트 확인
if( event > 0 ) {                                       // 이벤트 발생 경우
if( psd.revent & POLLIN ) {                             // 수신 이벤트인 경우
   recvfrom(                                            // 데이터 수신
              sd, rx_data, strlen(rx_data), 0,
              (struct sockaddr *) &cli_addr, &cli_len
          );
   }
}
```

(그림 2) 메시지 수신

socket()에 'SOCK_DGRAM'를 부여하여 UDP 소켓을 생성한다. sockaddr_in 구조체에 자신의 IP 와 PORT를 설정한 후 bind()를 이용하여 소켓정보를 등록한다. 소켓 디스크립터를 감시대상으로 지정하고 poll()를 이용하여 메시지 도착 여부를 검사하도록 한다. 메시지가 도착하면 recvfrom()를 이용하여 메시지를 수신한다.

수신된 메시지는 JSON 포맷 데이터이다. 이 데이터는 모터를 제어하기 위한 채널과 속도 값이 인코딩되어있다. JSON 포맷 메시지로부터 채널 및 속도 값의 디코딩은 API 함수[5]를 이용하여 (그림 3)과 같이 설계한다.

```
root   = json_loads(rx_data);              // 오브젝트 생성

obj_ch = json_object_get(root, "channel"); // key에 해당 값 얻기
obj_sp = json_object_get(root, "speed");   // key에 해당 값 얻기

channel = json_integer_value(obj_ch);      // 정수 값으로 변경
speed  = json_integer_value(obj_sp);       // 정수 값으로 변경
```

(그림 3) JSON 메시지 파싱

json_loads()를 이용하여 JSON 오브젝트로 만들고, json_object_get()를 이용하여 string 에 해당하는 value 를 얻는다. 마지막으로 json_integer_value()를 이용하여 string 의 value 를 정수 값으로 변환한다.

2.2 모터 속도 제어

모터의 속도 제어는 모터의 속도를 직접적으로 제어하는 ESC(Electronic Speed Control)와 연결된 임베디드 시스템의 GPIO 에 일정한 주기 동안에 계산된 횟수의 ON/OFF 를 반복적으로 설정함으로써 모터의 속도를 제어하게 된다. 따라서 모터 속도 제어를 위한 응용 프로그램과 GPIO 에 ON/OFF 를 설정하기 위한 디바이스 드라이버, 그리고 주기적으로 실행시키기 위한 타이머 인터럽트의 콜백 함수가 요구된다.

모터의 속도 제어를 위한 응용 프로그램은 (그림 4)와 같이 설계한다.

```
// 디바이스를 열기
fd = open(DEV_FILE_NAME, O_RDWR | O_NDELAY );

// 타이머 인터럽트 주기 설정
cycle_data.freq_tick  = freq_usec;
cycle_data.period_tick = period_usec / freq_usec;
cycle_data.center_tick = center_usec / freq_usec;
ioctl(fd, IOCTL_CYCLE_TIME, &cycle_data );

// 타이머 인터럽트 활성화
ioctl(fd, IOCTL_PWM_START );

// 모터 속도 제어
pwm_data.channel= channel;
pwm_data.speed  = speed;
ioctl( fd, IOCTL_PWM_CTRL, &pwm_data );
```

(그림 4) 응용 프로그램

open()를 이용하여 모터 제어를 담당하는 디바이스 파일을 연다. 타이머 인터럽트의 콜백 함수를 주기적으로 호출하기 위해 타이머 인터럽트 주기를 설정하고 타이머 인터럽트를 활성화 한다. JSON 파싱 데이터인 채널 값과 속도 값을 이용하여 채널에 해당하는 모터 속도를 제어한다.

타이머 인터럽트의 주기 설정, 인터럽트 활성화 그리고 모터 속도 제어를 위한 디바이스 드라이버 ioctl()은 (그림 5)와 같이 설계한다.

```
// 디바이스 드라이버의 ioctl()
static int motor_ioctl(
                        struct inode *inode, struct file *filp,
                        unsigned int cmd, unsigned long arg
                      ){
    switch(cmd)
    {
    case IOCTL_CYCLE_TIME:
    // 타이머 인터럽트 주기 설정
    copy_from_user(&cycle_data, cycle, sizeof(cycle_info) );
    motor_pwm_timer_freq=cycle_data.freq_tick;
    motor_pwm_period=cycle_data.period_tick;
    motor_pwm_duty_center=cycle_data.center_tick;
    break;

    case IOCTL_PWM_START:
    // 타이머 인터럽트 활성화
    pwm_base_count = 0;
    hwtimer_start(USE_HWTIMER, motor_pwm_freq,
    motor_pwm_mng );
    break;

    case IOCTL_PWM_CTRL:
    // 모터 속도 설정
    copy_from_user(&pwm_data, pwm sizeof( pwm_info_t));
    pwm_load_offset[pwm_data.channel] = pwm_data.speed;
    break;
    }
}
```

(그림 5) 디바이스 드라이버의 ioctl()

ioctl()이 호출되면 switch()를 이용하여 case 에 맞는 명령을 처리한다. IOCTL_CYCLE_TIME 인 경우 copy_from_user()를 이용하여 응용 프로그램에서 타이머 인터럽트 주기 데이터를 커널 영역으로 복사하여 주기를 설정한다. IOCTL_PWM_START 인 경우 타이머 인터럽트를 활성화하여 인터럽트 주기마다 콜백 함수가 호출된다. IOCTL_PWM_CTRL 인 경우 copy_from_user()를 이용하여 응용 프로그램에서 채널 값과 속도 값을 커널 영역으로 복사하여 설정한다.

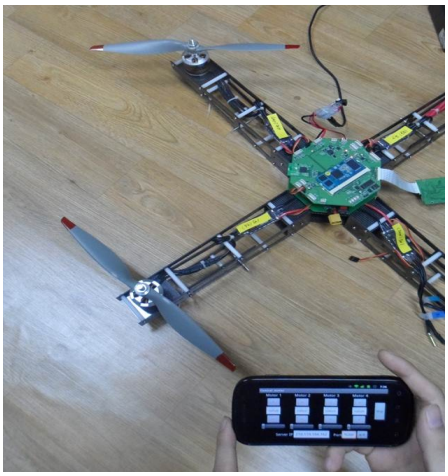
타이머 인터럽트에 의해 주기적으로 호출되는 콜백 함수는 (그림 6)과 같이 설계 한다. 타이머 인터럽트 주기에 맞추어 베이스 카운터 값을 증가시키고, 베이스 카운터 값이 주기 시간보다 클 경우 카운터 값을 0 값으로 초기화 한다. 모터 속도를 일정하게 하기 위해 채널 별 속도를 설정한다. 베이스 카운트 값이 0 일 경우 현재 채널에 해당하는 속도와 GPIO 를 설정한다. 모터 속도 값이 베이스 카운터 값과 같지 않을 경우에는 지속적으로 GPIO 를 활성화 하고, 같을 경우 GPIO 에 ON/OFF 를 설정함으로써 각 모터의 속도를 제어한다.

```
static void motor_pwm_mng( void )
{
    pwm_base_count++;
    if (pwm_base_count>= pwm_period) pwm_base_count = 0;

    // 채널 별 속도 설정
    if( pwm_base_count == 0 ){
        for( ch = 0; ch < PWM_MAX; ch++ )
            pwm_off_count[ch]=pwm_duty_center+pwm_offset[ch];

        // 재주기 GPIO 설정
        for( ch = 0; ch < PWM_MAX; ch++ ){
            if((pwm_enable[ch]==0)||(pwm_off_count[ch]==0)){
                motor_gpio_off_channel(ch);
            }
            else{
                motor_gpio_on_channel(ch);
            }
        }
    }

    // 모터 속도 조절
    for( ch = 0; ch < PWM_MAX; ch++){
        if(pwm_base_count != pwm_off_count[ch]) continue;
            motor_gpio_off_channel(ch);
    }
}
```

(그림 6) 타이머 인터럽트 콜백 함수

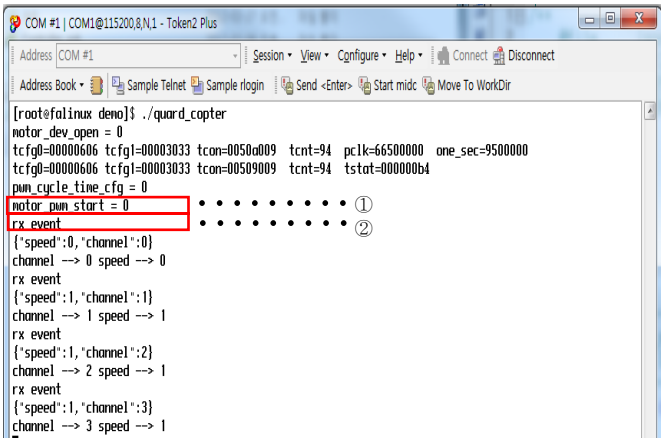
3. 구현 및 실험

설계한 부분들을 리눅스(Ubuntu 10.04)에서 구현하여 (그림 7)과 같이 ARM 프로세서 기반의 임베디드 보드가[6]가 탑재된 실험용 쿼드콥터를 이용하여 실험하였다.



(그림 7) 실험용 쿼드콥터

(그림 8)은 실험과정에서 인터넷으로부터 수신된 JSON 메시지와 이를 파싱한 결과를 PC 에서 확인한 화면이다. 맨 처음 수신된 JSON 메시지의 내용은 {“speed” :0, “channel” :0} 이고(①), 이를 파싱한 결과는 channel → 0 speed → 0 이다(②). 이러한 방법으로 메시지 수신과 파싱 결과를 확인하고 이 값에 따라 (그림 7)과 같은 실험용 쿼드콥터 모터의 속도가 정상적으로 제어됨을 확인하였다.



(그림 8) 실험 결과 화면

4. 결론

본 논문에서는 인터넷을 통하여 쿼드콥터의 모터를 제어하기 위한 임베디드 리눅스 시스템을 설계하고 그 기능적 타당성을 제시하였다. 제시한 시스템은 기본적인 인터넷으로부터 JSON 포맷의 메시지를 수신하고 이를 파싱한 후 해당 채널의 모터 속도를 제어하는 것이다. 이를 위하여 응용 프로그램과 디바이스 드라이버 그리고 주기적으로 실행하기 위한 타이머 인터럽트의 콜백 함수를 설계하고, ARM 프로세서가 탑재된 임베디드 보드를 이용하여 실험용 쿼드콥터의 모터 속도를 제어함으로써 그 타당성을 검토하였다.

참고문헌

[1] 조성욱, 허성식, 유동일, 심현철, 최형식 “무인항공기의 근거리 비행체 탐지 및 추적을 위한 영상처리 알고리즘”한국항공우주학회, 한국항공우주학회 2011년도 춘계학술대회

[2] http://lca2007.linux.org.au/talk/229.html

[3] 김환국, 정현철, 원유재(한국인터넷진흥원) “모바일 클라우드 보안 이슈 및 대응기술 요구사항” 정보처리학회지 2011 년 9 월 제 18 권 제 5 호

[4] http://www.json.org/json-ko.html

[5] http://www.digip.org/jansson/doc/2.3/apiref.html

[6] http://www.falinux.com