드론 프로젝트 발표

임베디드스쿨1기

Lv1과정

2020. 11. 02 ~ 2021.04.03

김인겸

목차

1. 하드웨어 - 1) 개발 개요

- **개발 내용**
  - 8bit MCU를 이용한 쿼드콥터 드론 비행 프로젝트

- **적용 기술**
  - UART통신
  - I2C 통신
  - SPI통신
  - 수신기 데이터 파싱
  - ESC를 이용한 BLDC모터 제어
  - PID 제어

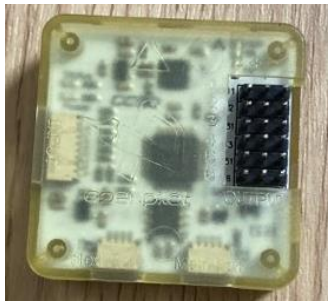
- **개발 환경**
  - Atmel Studio 7.0 (AVR통합 개발 환경)
  - Tera Term (UART통신)
  - C언어 기반

1. 하드웨어 - 2) BOM 리스트

번호	품명	수량	가격(원)	조달경로	구매사이트
1	CC3D FC	1	114,239	해외	알리 익스프레스
2	F330 프레임 키트	1	〃	〃	〃
3	8038 프로펠러	4	〃	〃	〃
4	XXD A2212 KV930 BLDC 모터	4	〃	〃	〃
5	Simon 20A ESC	4	〃	〃	〃
6	FLYSKY 조종기	1	〃	〃	〃
7	FS-IA6B 수신기	1	〃	〃	〃
8	배터리 체커	1	3,997	해외	알리 익스프레스
9	리포 배터리 충전기	1	6,340	해외	알리 익스프레스
10	2800mAh 3S 60C 리튬폴리머 배터리	1	32,000	국내	팰콘샵
11	아두이노 나노 호환보드	1	4,070	국내	협신 전자
12	MPU 6050	1	1,630	국내	협신 전자
			162,276		
13	JMOD-128-1(ATmega128)	1	30,000(택포)	국내	제이씨넷
14	MPU9250	1	5060		협신 전자
			197,336		

1. 하드웨어 - 2) BOM 리스트

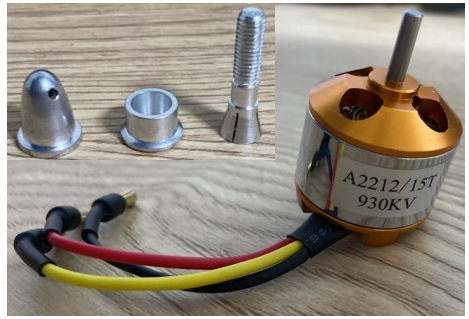
1) CC3D FC



2) F330 프레임 키트

3) 8038 프로펠러

4) 930KV BLDC 모터 * 4



5) Simon 20A ESC * 4

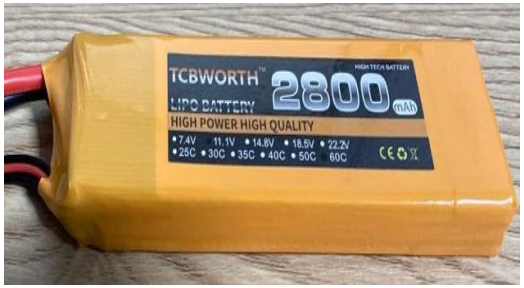
6) FLYSKY 조종기

7) FS-IA6B 수신기

8) 배터리 체커

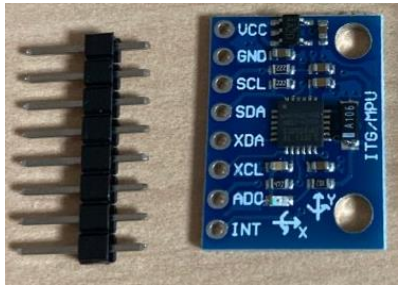
9) 리포 배터리 충전기

10) 리튬폴리머 배터리

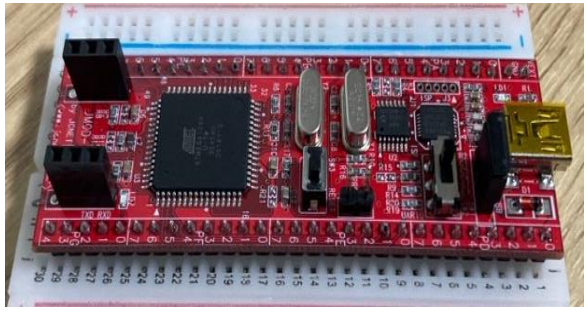


11) 아두이노 나노

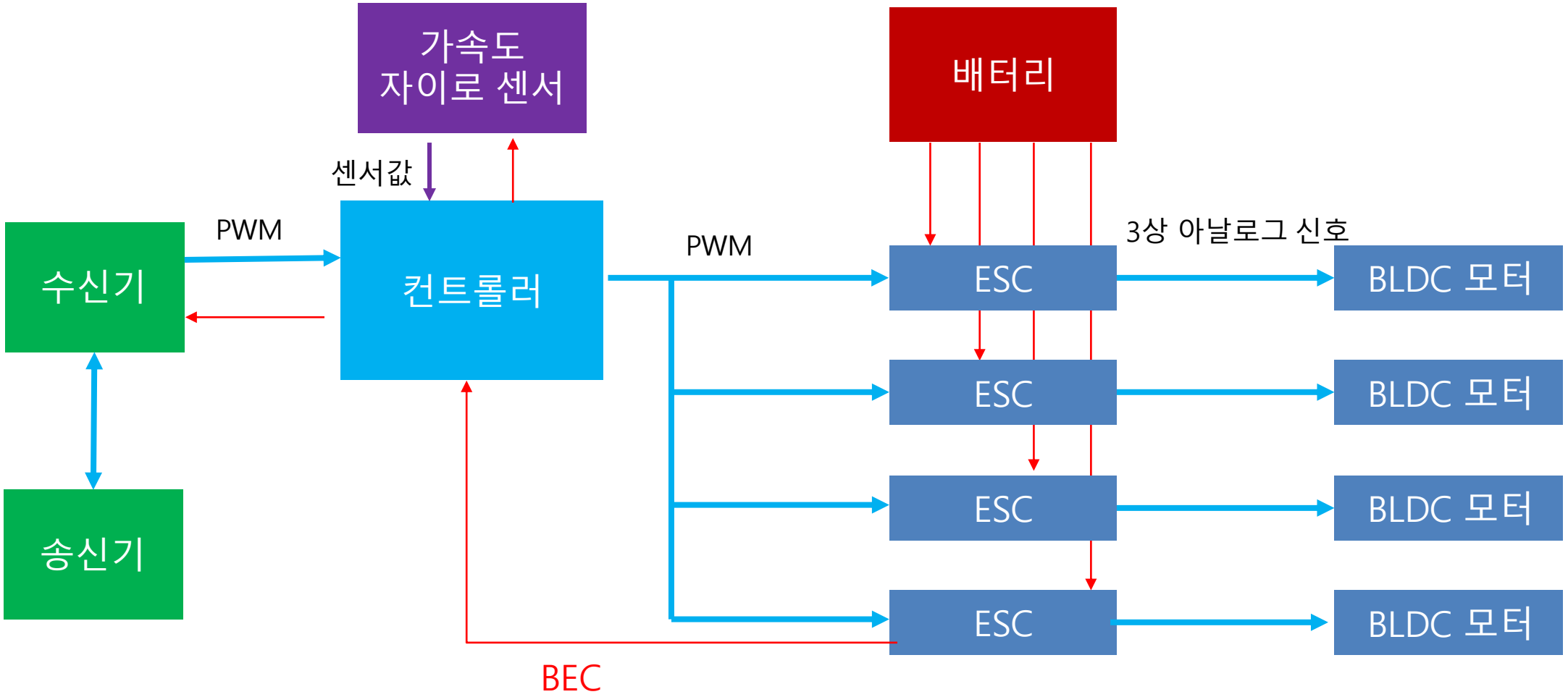
12) MPU6050



13) ATmega128보드



1. 하드웨어 – 3) 블록도


가속도
자이로 센서
센서값
배터리
수신기
PWM
컨트롤러
PWM
3상 아날로그 신호
ESC
BLDC 모터
ESC
BLDC 모터
ESC
BLDC 모터
ESC
BLDC 모터
송신기
BEC

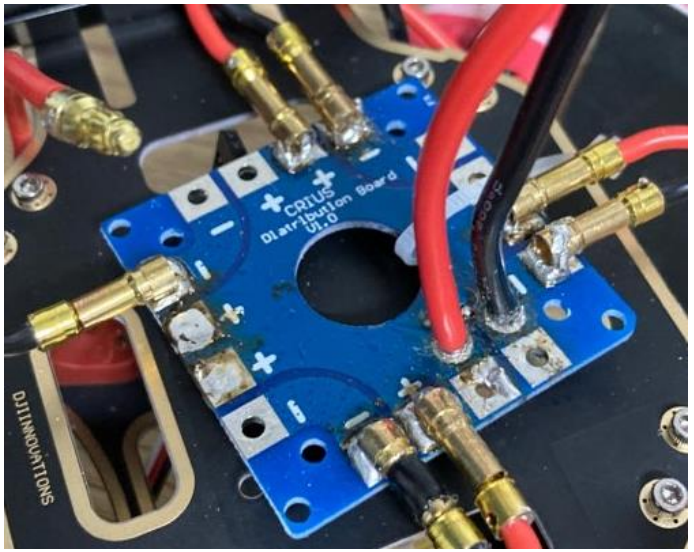
1. 하드웨어 – 4) 부품 조립 및 시험 비행

부품 조립

시험 비행

1. 하드웨어 – 5) 트러블 슈팅

① PDB 및 배터리 커넥터 납땜에 어려움을 겪음



② Librepilot 설치가 안되는 문제

[해결 방법]

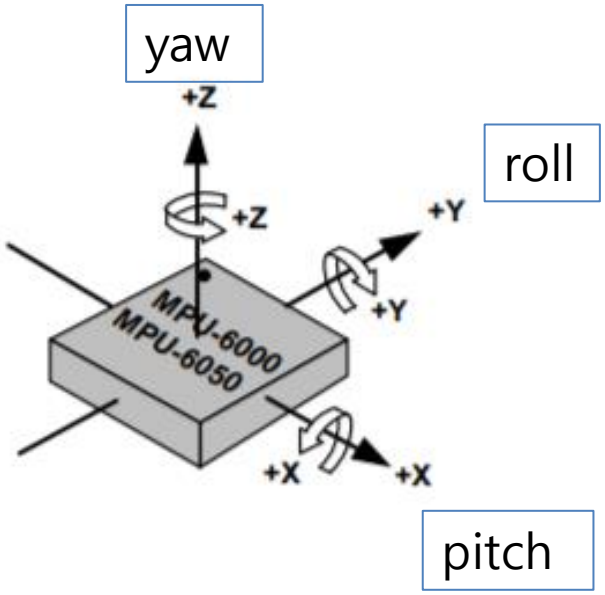
③ CC3D FC 세팅하는데 어려움을 겪음.

2. 펌웨어 - 1)MPU9250

- 특징
- 가속도, 각속도, 온도 측정
- 16비트 ADC
- 400kHz 고속 모드 I2C
- 1MHz SPI
- 가속도 범위 : 초당 ± 250, ± 500, ± 1000 또는 ± 2000도 (dps)
- 각속도 : ± 2g, ± 4g, ± 8g ± 16g
- ADC 샘플 속도 : 초당 8,000 ~3.9개의 샘플
- DMP기능

- 용도
- 모션 지원 게임 어플리케이션
- 3D 원격 제어
- 웨어러블 센서
- 만보계
- 제스처 인식

GY-521모듈



* DMP(Digital Motion Processor) : 가속도, 각속도 연산 프로그램으로서 pitch, roll, yaw값, 쿼터니안, 오일러 값 계산 가능

MPU-6000-Datasheet1.pdf

MPU-6050-Register-Map.pdf

2. 펌웨어 - 1)MPU9250

<써야 할 레지스터>

19	25	SMPLRT_DIV	R/W	SMPLRT_DIV[7:0]						
1A	26	CONFIG	R/W	-	FIFO_MODE	EXT_SYNC_SET[2:0]		DLPF_CFG[2:0]		
1B	27	GYRO_CONFIG	R/W	XG_ST	YG_ST	ZG_ST	GYRO_FS_SEL [1:0]	-	FCHOICE_B[1:0]	
1C	28	ACCEL_CONFIG	R/W	XA_ST	YA_ST	ZA_ST	ACCEL_FS_SEL[1:0]	-		
1D	29	ACCEL_CONFIG 2	R/W	-			ACCEL_FC HOICE_B	A_DLPF_CFG		
6B	107	PWR_MGMT_1	R/W	DEVICE_RESET	SLEEP	CYCLE	GYRO_STANDBY	TEMP_DIS	CLKSEL[2:0]	

When set, the chip is set to sleep mode.

SAMPLE_RATE = INTERNAL_SAMPLE_RATE / (1 + SMPLRT_DIV)

where INTERNAL_SAMPLE_RATE = 1kHz

Code	Clock Source
0	Internal 20MHz oscillator
1	Auto selects the best available clock source – PLL if ready, else use the Internal oscillator
2	Auto selects the best available clock source – PLL if ready, else use the Internal oscillator
3	Auto selects the best available clock source – PLL if ready, else use the Internal oscillator
4	Auto selects the best available clock source – PLL if ready, else use the Internal oscillator
5	Auto selects the best available clock source – PLL if ready, else use the Internal oscillator
6	Internal 20MHz oscillator
7	Stops the clock and keeps timing generator in reset

2. 펌웨어 - 1)MPU9250

<읽어야 할 레지스터>

3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT_H[15:8]
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT_L[7:0]
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT_H[15:8]
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT_L[7:0]
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT_H[15:8]
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT_L[7:0]
41	65	TEMP_OUT_H	R	TEMP_OUT_H[15:8]
42	66	TEMP_OUT_L	R	TEMP_OUT_L[7:0]
43	67	GYRO_XOUT_H	R	GYRO_XOUT_H[15:8]
44	68	GYRO_XOUT_L	R	GYRO_XOUT_L[7:0]
45	69	GYRO_YOUT_H	R	GYRO_YOUT_H[15:8]
46	70	GYRO_YOUT_L	R	GYRO_YOUT_L[7:0]
47	71	GYRO_ZOUT_H	R	GYRO_ZOUT_H[15:8]
48	72	GYRO_ZOUT_L	R	GYRO_ZOUT_L[7:0]

I^2C ADDRESS	AD0 = 0 AD0 = 1		1101000 1101001

2. 펌웨어 - 1)MPU9250 (I2C)

<I2C 데이터 쓰기>

Single-Byte Write Sequence

Register Address

Master	S	AD+W		RA		DATA		P
Slave			ACK		ACK		ACK	

Burst Write Sequence

Master	S	AD+W		RA		DATA		DATA		P
Slave			ACK		ACK		ACK		ACK	

```
void mpu6050_write(uint8_t reg_address, uint8_t data)
{
    i2c_start();
    i2c_SLA_W(MPU_ADDR1);
    i2c_write_data(reg_address);
    i2c_write_data(data);
    i2c_stop();
}
```

2. 펌웨어 - 1)MPU9250 (I2C)

<I2C 데이터 읽기> *Single-Byte Read Sequence*

Master	S	AD+W		RA		S	AD+R			NACK	P
Slave			ACK		ACK			ACK	DATA		

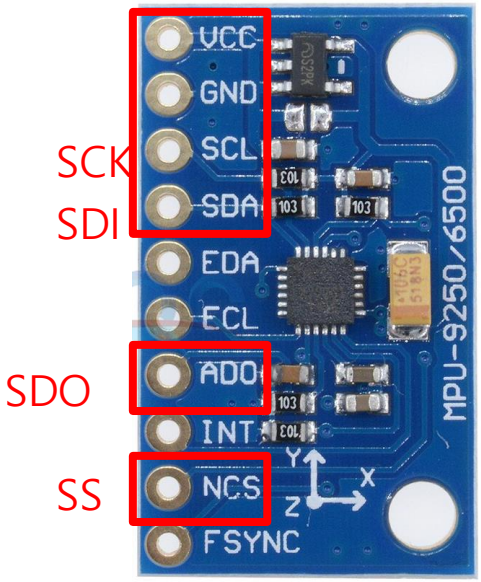
Burst Read Sequence

Master	S	AD+W		RA		S	AD+R			ACK		NACK	P
Slave			ACK		ACK			ACK	DATA		DATA		

```
void mpu6050_read(uint8_t reg_address, uint8_t len, uint8_t* buffer)
{
    i2c_start();
    i2c_SLA_W(MPU_ADDR1);
    i2c_write_data(reg_address);
    i2c_rep_start();
    i2c_SLA_R(MPU_ADDR1);

    for(int i = 0; i < (len-1); i++)
    {
        buffer[i] = i2c_readACK();
    }
    buffer[len-1] = i2c_readNACK();

    i2c_stop();
}
```

2. 펌웨어 - 1)MPU9250 (SPI)

- 특징
- 9축 가속도, 자이로, 지자계 센서
- SPI, I2C통신
- MPU6050 기능에 SPI통신, 지자계 센서 추가



SPI Operational Features

1. Data is delivered MSB first and LSB last
2. Data is latched on the rising edge of SCLK
3. Data should be transitioned on the falling edge of SCLK
4. The maximum frequency of SCLK is 1MHz
5. SPI read and write operations are completed in 16 or more clock cycles (two or more bytes). The first byte contains the SPI Address, and the following byte(s) contain(s) the SPI data. The first bit of the first byte contains the Read/Write bit and indicates the Read (1) or Write (0) operation. The following 7 bits contain the Register Address. In cases of multiple-byte Read/Writes, data is two or more bytes:

SPI Address format

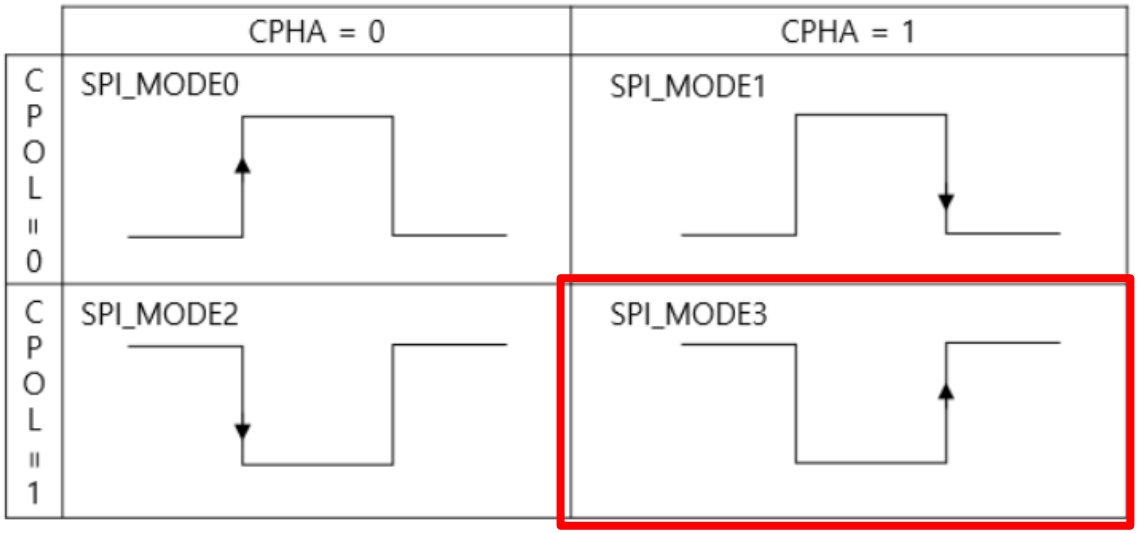
MSB							**LSB**
R/W	A6	A5	A4	A3	A2	A1	A0

SPI Data format

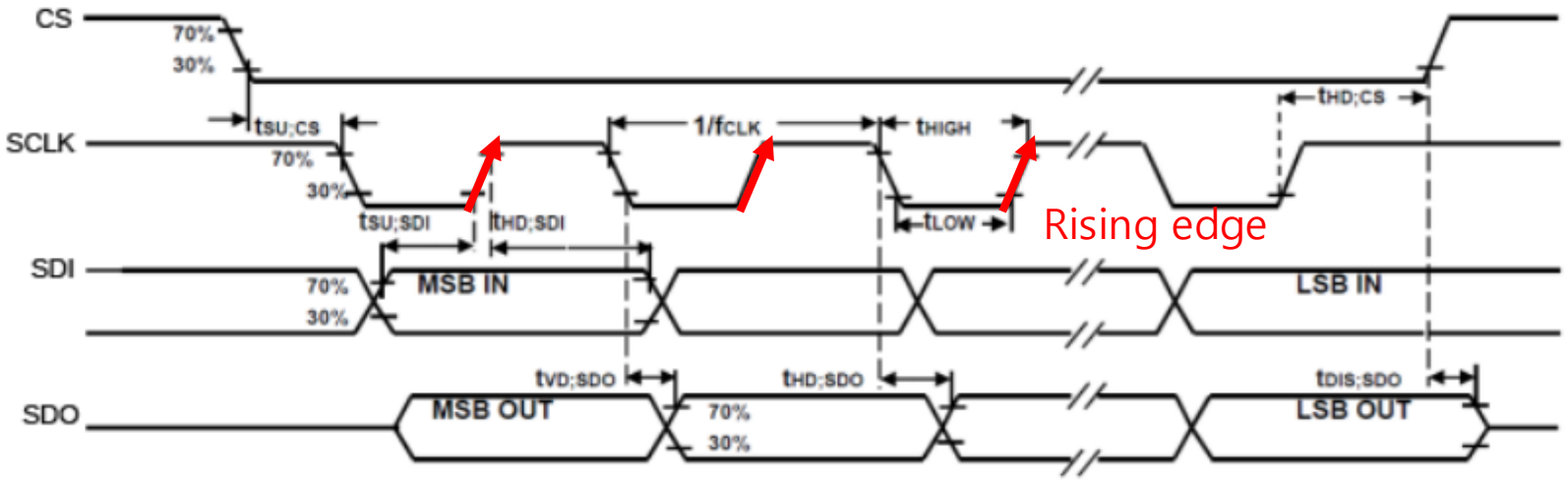
MSB							**LSB**
D7	D6	D5	D4	D3	D2	D1	D0

2. 펌웨어 - 1)MPU9250 (SPI)

- SCLK mode



- MPU9250 SPI SCLK Timing



SPI Bus Timing Diagram

2. 펌웨어 - 1)MPU9250 (SPI)

```c
void SPI_init(void)
{
    //SPCR = (1<<SPE)|(1<<MSTR)|(1<<CPHA)|(1<<SPR0); //mode1, 1MHz

    SPCR = (1<<SPE)|(1<<MSTR)|(1<<CPOL)|(1<<CPHA)|(1<<SPR0); //mode3, 1MHz

    sbi(DDRB, 0); //SS
    sbi(PORTB, 1);

    sbi(DDRB, 1); //SCK
    sbi(DDRB, 2); //MOSI
    cbi(DDRB, 3); //MISO


}
```

```c
void SPI_write(uint8_t reg_address, uint8_t data)
{
    cbi(PORTB, 0); //SS clear

    SPDR = (WRITE<<7) | reg_address;
    while(!(SPSR & (1<<SPIF)));

    SPDR = data;
    while(!(SPSR & (1<<SPIF)));

    sbi(PORTB, 0); //SS set
}
```

```c
uint8_t SPI_read(uint8_t reg_address)
{
    uint8_t data;

    cbi(PORTB, 0); //SS clear

    SPDR = (READ<<7) | reg_address;
    while(!(SPSR & (1<<SPIF)));

    SPDR = 0x00; //데이터 수신을 위한 임의의 데이터 전송
    while(!(SPSR & (1<<SPIF)));

    data = SPDR;

    sbi(PORTB, 0); //SS set

    return data;
}
```

2. 펌웨어 - 1)MPU9250

① 가속도 센서를 이용해서 roll, pitch를 구하는 방법

- Roll 값

$$\text{angle(Y)} = tan^{-1}\left(\frac{-AcX}{\sqrt{Ac^2Y + Ac^2Z}}\right) \times \left(\frac{180}{\pi}\right)$$

- pitch 값

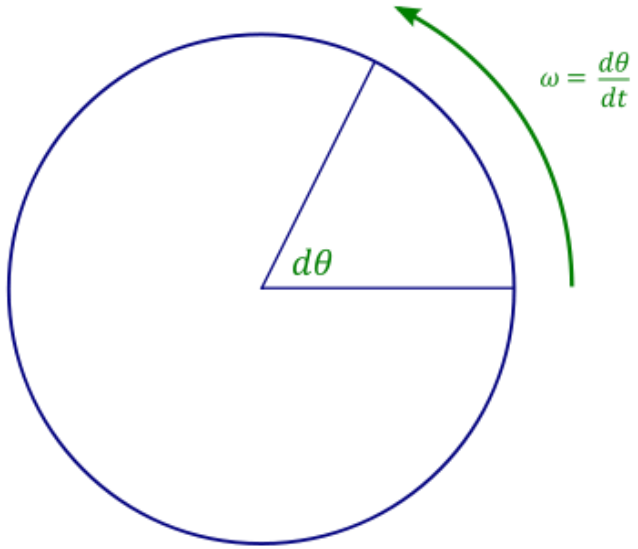
$$\text{angle(X)} = tan^{-1}\left(\frac{-AcY}{\sqrt{Ac^2X + Ac^2Z}}\right) \times \left(\frac{180}{\pi}\right)$$

- 가속도 센서를 이용하면 yaw값을 구할 수 없다
- 모터의 진동과 같은 순간적인 잡음에 취약하다.
- 시간이 지남에 따라 값이 안정적으로 유지됨.

```
Accel_roll = atan2((-1) * Ax_tmp, sqrt(Ay_tmp*Ay_tmp + Az_tmp*Az_tmp)) * 180.0F / (double)PI;
Accel_pitch = atan2(Ay_tmp, sqrt(Ax_tmp*Ax_tmp + Az_tmp*Az_tmp)) * 180.0F / (double)PI;
```

2. 펌웨어 - 1)MPU9250

② 자이로 센서를 이용하여 roll, pitch 구하는 방법



- 자이로 센서를 통해 얻은 값 = 각속도(w)
- 구하고자 하는 각도 = 각속도(w) * dt
- 따라서 적분하면 dt주기로 적분하면 됨

- 초기 값은 안정적
- 적분에 의해 오차가 누적되어 값이 표류하는 드리프트 현상 발생

```
void get_Gy_roll_pitch_yaw(accel_t_gyro* Paccel_gyro, accel_t_gyro* Paccel_gyro_ave, angle* Pangle)
{
    double Gx_tmp, Gy_tmp, Gz_tmp;

    Gx_tmp = (double)(Paccel_gyro->raw.x_gyro - Paccel_gyro_ave->raw.x_gyro) / FS_SEL3;
    Gy_tmp = (double)(Paccel_gyro->raw.y_gyro - Paccel_gyro_ave->raw.y_gyro) / FS_SEL3;
    Gz_tmp = (double)(Paccel_gyro->raw.z_gyro - Paccel_gyro_ave->raw.z_gyro) / FS_SEL3;

    Pangle->roll = Gx_tmp * DT + Pangle->roll;
    Pangle->pitch = Gy_tmp * DT + Pangle->pitch;
    Pangle->yaw = Gz_tmp * DT + Pangle->yaw;
}
```

2. 펌웨어 - 1)MPU9250

③ 상보필터를 이용해 roll, pitch를 구하는 방법

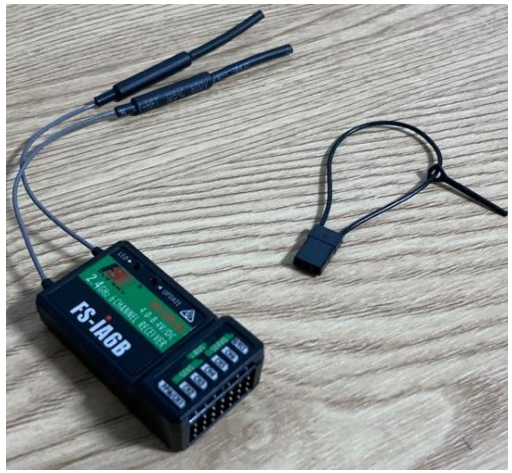
```
const float ALPHA = 0.96;
float tmp_angle_x, tmp_angle_y, tmp_angle_z;

tmp_angle_x = filtered_angle_x + gyro_x * dt;
tmp_angle_y = filtered_angle_y + gyro_y * dt;
tmp_angle_z = filtered_angle_z + gyro_z * dt;

filtered_angle_x = ALPHA * tmp_angle_x + (1.0 - ALPHA) * accel_angle_x;
filtered_angle_y = ALPHA * tmp_angle_y + (1.0 - ALPHA) * accel_angle_y;
filtered_angle_z = tmp_angle_z;
```

<아두이노 MPU6050 코드 참고>

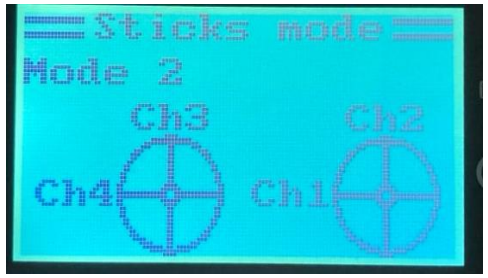
2. 펌웨어 - 2) 수신기

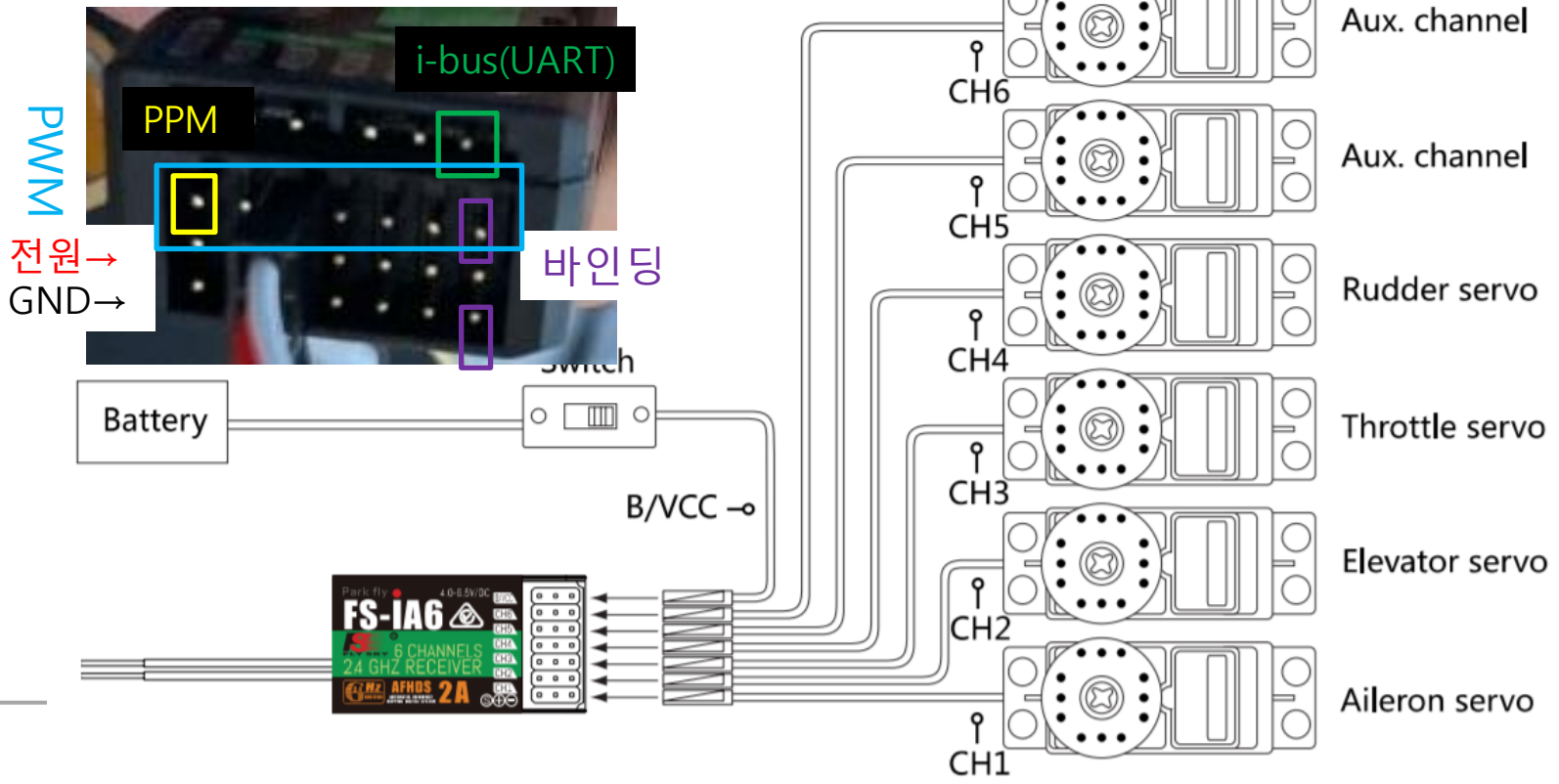
FLYSKY 조종기

FS-IA6B 수신기



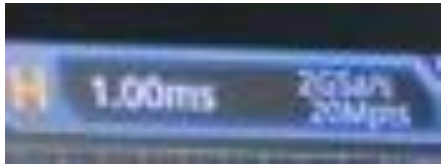
- 6채널
- RF범위 : 2.4 ~ 2.8Hz
- 대역폭 : 500kHz
- 전원 : 4.0 ~ 6.5V

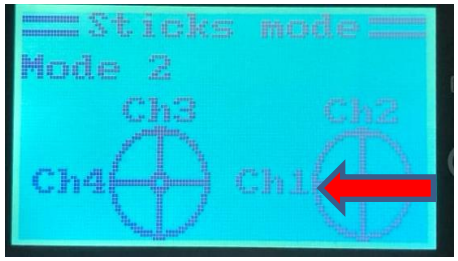




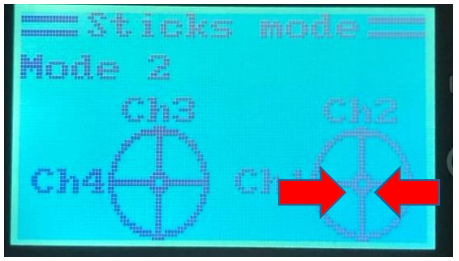
2. 펌웨어 - 2) 수신기

- **PWM 파형 분석**

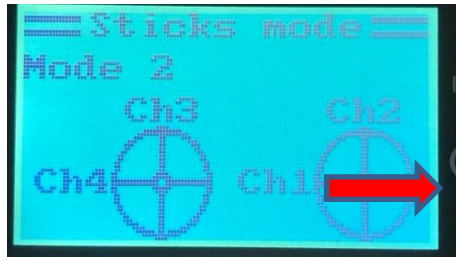




- 펄스폭(최소) : 1ms



- 펄스폭(중간) : 1.5ms



- 펄스폭(최대) : 2ms

2. 펌웨어 - 2) 수신기

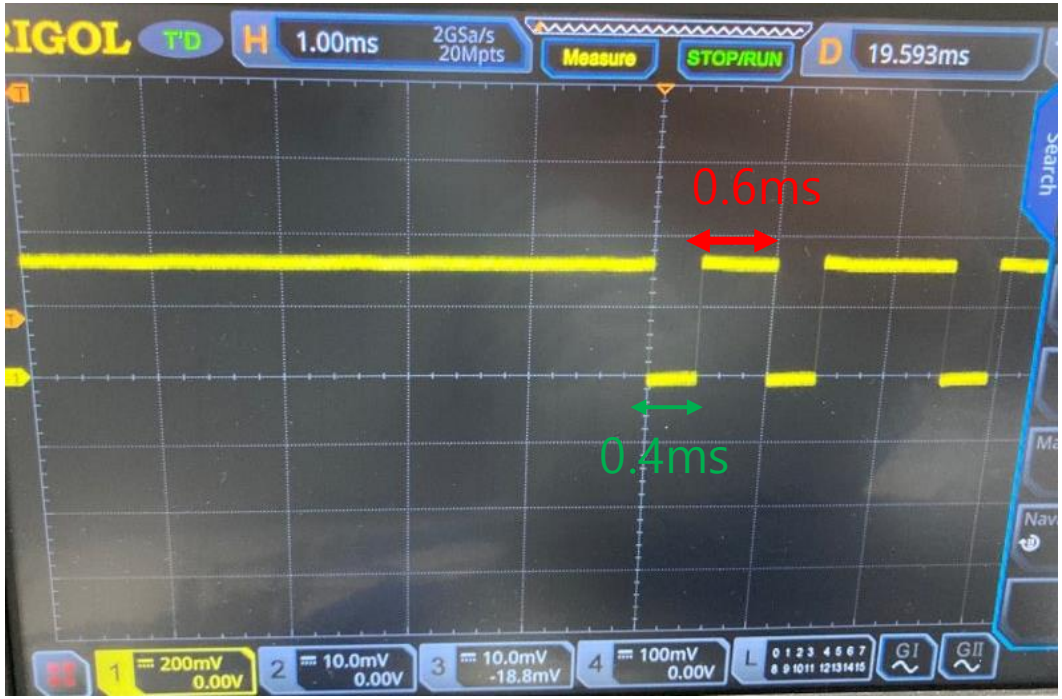
- **PPM 파형 분석**

- 주기 : 20ms(고정)
- Channel : 8채널
- Low Level은 0.4ms로 고정되고 High Level만 증감한다

* 수신기 PPM 설정 방법
: 조종기 메뉴 -> System -> RX Setup -> PPM Output을 ON으로 설정

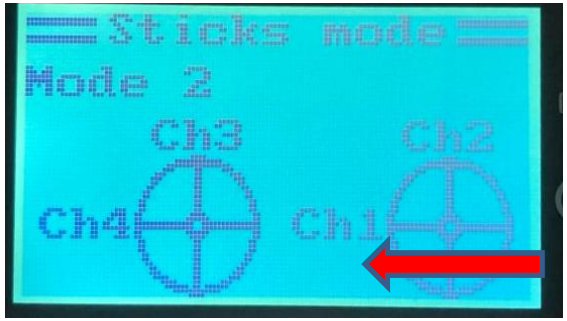
2. 펌웨어 - 2) 수신기

- **PPM 파형 분석**

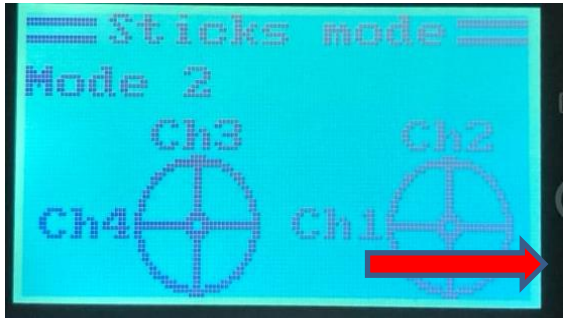


Ch1 최솟값

Ch1 최댓값



Ch1최소



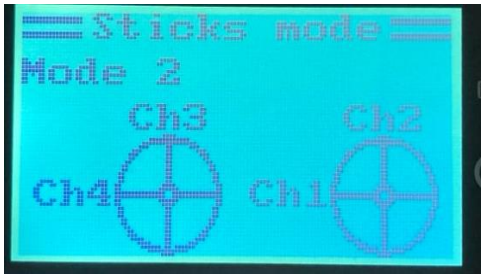
Ch1최대

2. 펌웨어 - 2) 수신기

```
ISR (INT7_vect)
{
    if(EICRB == 0xC0) //rising edge이면
    {
        t1 = TCNT1;

        EICRB = 0x80;  //falling edge ON
        sbi(EIFR, INTF7);

        rising_cnt++;
    }
    else//falling edge이면
    {
        t2 = TCNT1;

        falling_cnt++;

        //시간 계산(us단위)
        if(t1 > t2)
        {
            duty = 39999 - t1 + t2;
        }
        else
        {
            duty = t2 - t1;
        }

        duty = duty * COUNT_s * s_to_us;

        EICRB = 0xC0;  //rising edge ON
        sbi(EIFR, INTF7);

    }
    //채널이 밀리지 않고 파싱되게끔 해주는 코드
    if(duty > 4000) //마지막 잉여 채널은 4600~10500us 범위를 가짐
    {
        rising_cnt = 9;
        falling_cnt = 9;
    }

    //channel 할당
    switch(rising_cnt + falling_cnt) //업, 다운 한 번씩 발생할 때마다 주기를 파싱
    {
        case 2:
        channel.ch1 = duty;
        break;
        case 4:
        channel.ch2 = duty;
        break;
        case 6:
        channel.ch3 = duty;
        break;
        case 8:
        channel.ch4 = duty;
        break;
        case 10:
        channel.ch5 = duty;
        break;
        case 12:
        channel.ch6 = duty;
        break;
        case 14:
        channel.ch7 = duty;
        break;
        case 16:
        channel.ch8 = duty;
        break;
        case 18: //
        channel.remainder_value = duty;
        falling_cnt = 0;
        rising_cnt = 0;
        default:
        //
        break;
    }
}
```

는 건 아무것도 없다.

2. 펌웨어 - 2) 수신기




```
ch1 : 1104    ch2 : 1101    ch3 : 1129    ch4 : 1095
ch1 : 1104    ch2 : 1101    ch3 : 1129    ch4 : 1095
ch1 : 1104    ch2 : 1101    ch3 : 1129    ch4 : 1095
ch1 : 1104    ch2 : 1101    ch3 : 1129    ch4 : 1095
ch1 : 1104    ch2 : 1101    ch3 : 1129    ch4 : 1095
ch1 : 1373    ch2 : 1103    ch3 : 1129    ch4 : 1095
ch1 : 1602    ch2 : 1126    ch3 : 1129    ch4 : 1095
ch1 : 1601    ch2 : 1130    ch3 : 1129    ch4 : 1095
ch1 : 1602    ch2 : 1128    ch3 : 1129    ch4 : 1095
ch1 : 1601    ch2 : 1129    ch3 : 1129    ch4 : 1095
ch1 : 1602    ch2 : 1129    ch3 : 1129    ch4 : 1095
ch1 : 1601    ch2 : 1129    ch3 : 1129    ch4 : 1095
ch1 : 1601    ch2 : 1129    ch3 : 1129    ch4 : 1095
ch1 : 1602    ch2 : 1132    ch3 : 1129    ch4 : 1095
ch1 : 1602    ch2 : 1134    ch3 : 1129    ch4 : 1095
ch1 : 1602    ch2 : 1135    ch3 : 1129    ch4 : 1095
ch1 : 1601    ch2 : 1136    ch3 : 1129    ch4 : 1095
ch1 : 1601    ch2 : 1132    ch3 : 1129    ch4 : 1095
ch1 : 1601    ch2 : 1135    ch3 : 1129    ch4 : 1095
ch1 : 1601    ch2 : 1134    ch3 : 1129    ch4 : 1095
ch1 : 1602    ch2 : 1133    ch3 : 1129    ch4 : 1095
ch1 : 1602    ch2 : 1134    ch3 : 1129    ch4 : 1095
```

MIN : 600us
MID : 1100us
MAX : 1600us

2. 펌웨어 - 3) 모터

- XXD A2212 KV930 BLDC 모터 (4EA)

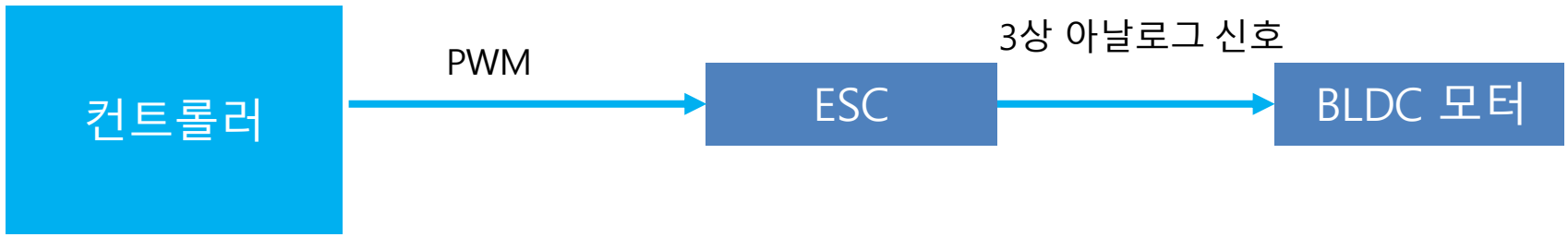
- KV930
- 무부하 전압 및 전류 : 10V, 0.5A
- Current Capacity(전류용량) : 12 A/60s
- 무게 : 64g

- Simon 20A ESC (4EA) : BLDC모터에 입력될 속도 제어 명령을 3가지 신호로 바꿔주는 역할

- 유지 전류 : 20A
- 피크 전류 : 30A(10s)
- 전압 : 2~3S lipo
- BEC : 5V/3A
- 무게 : 26g
- ESC의 전류 값 > 모터의 Max Amp
- SimonK 펌웨어 내장

- BEC : 모터용 베터리에서 수신기와 서보에 전류를 함께 공급해 줄 수 있도록 만들어진 회로이자 전압을 다운 시켜주는 Step Down Regulator
- 리니어 방식 : 다운된 전압 만큼을 열에너지로 소비하는 방식
- 스위치 방식 : On, Off 속도 조절로 전압을 다운 시키는 방식(잡음 발생 문제점)

2. 펌웨어 - 3) 모터



- ESC에 사용되는펌웨어 : Simon, BLHeli, BLHeli_S 등
- 20A Simon ESC update rate : 50~60Hz, 최소 1ms, 최대 2ms

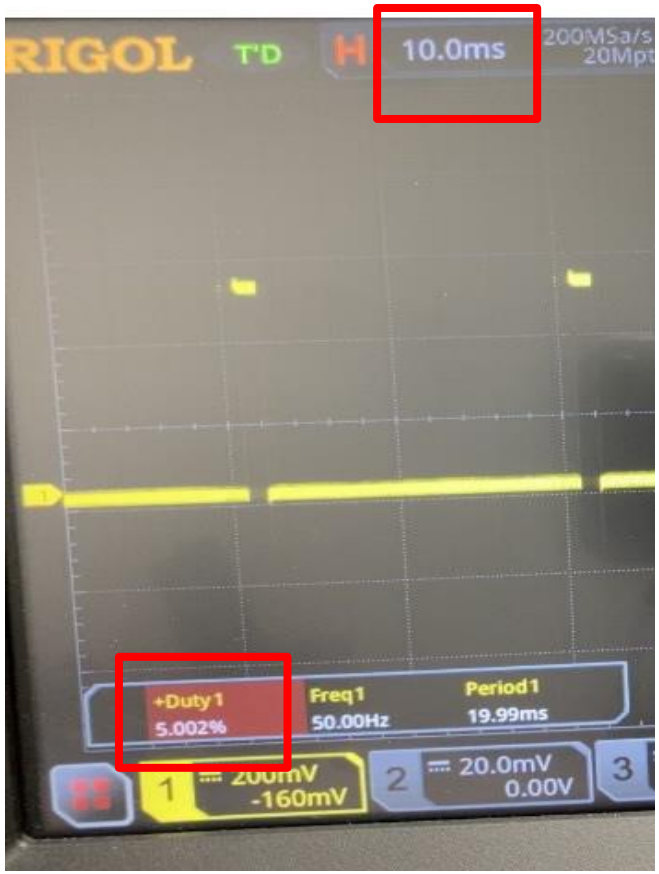
Control Signal:

20A BLDC ESC requires standard 50-60Hz PWM signal from any remote control as throttle input. You can also generate similar input signal from the microcontroller for making your own customized flying platform. Throttle speed is proportional to the width of the pulse. Maximum throttle position is user programmable. In general throttle is set at zero for 1mS pulse width and full at the 2mS pulse width.

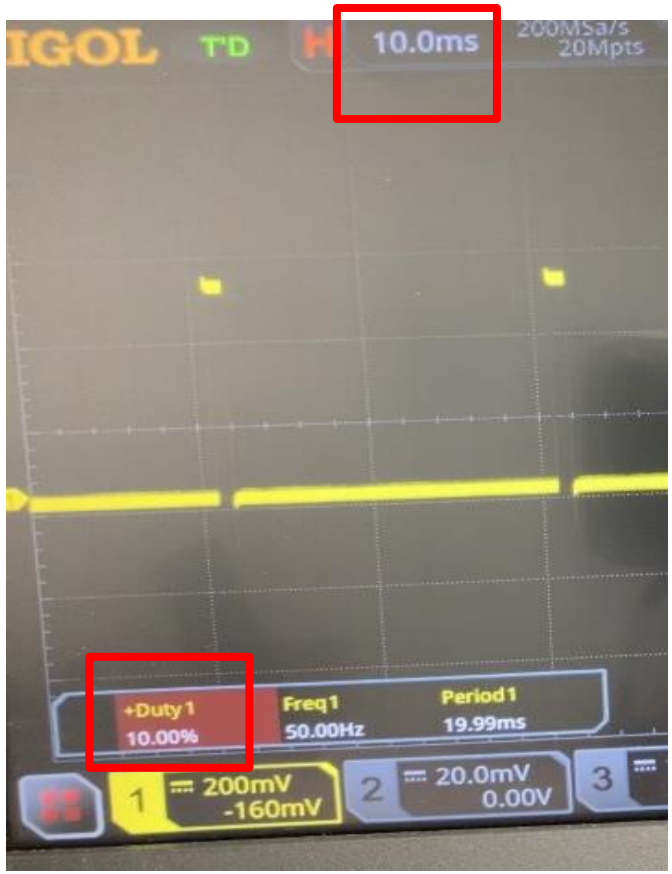
2. 펌웨어 - 3) 모터

- PWM 파형 생성
  - 16비트 타이머/카운터1, 3을 이용
  - Fast PWM모드(ICR -> TOP모드)
  - 주기 20ms, 최소값 1ms, 최대값 2ms
  - OC1A, OC1B, OC3A, OC3B핀 출력

- OCR 최솟값 : 2000
- OCR 최댓값 : 4000



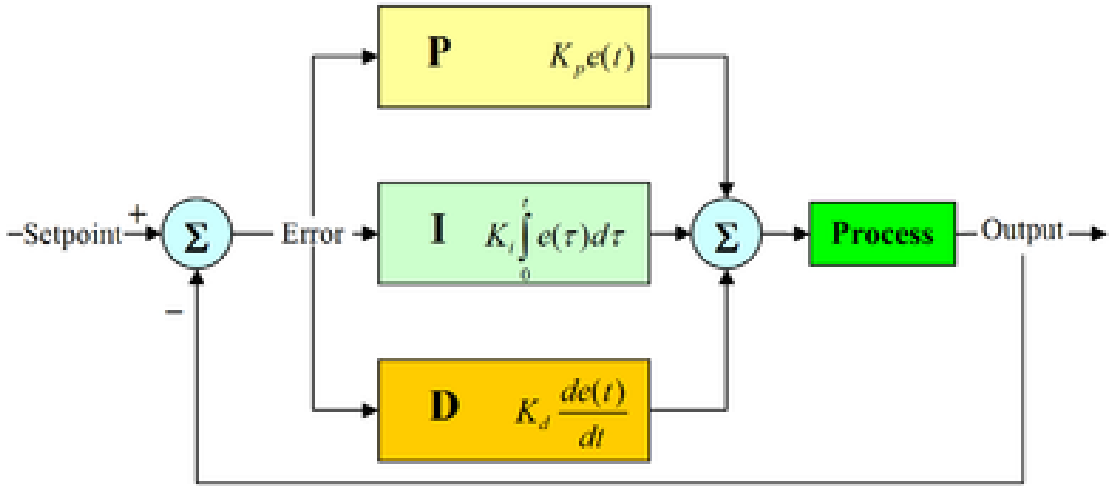
<OC1A핀 MIN>

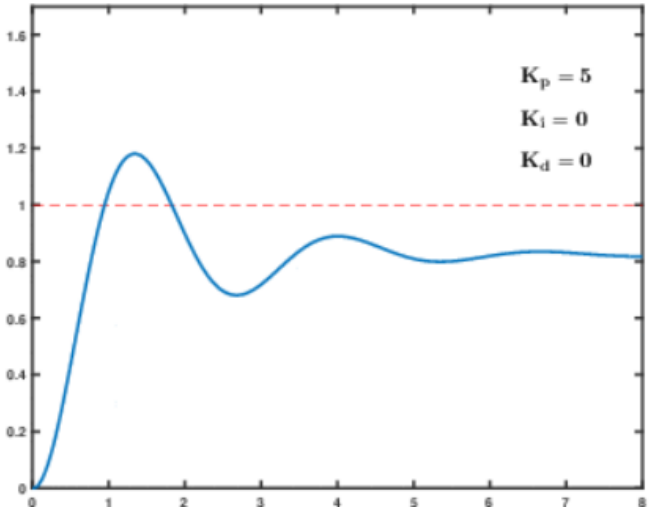


<OC1A핀 MAX>

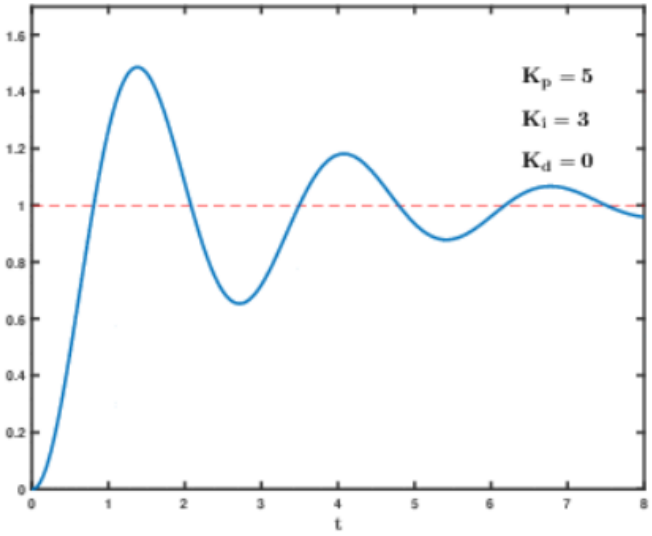
2. 펌웨어 - 4) PID

$$\mathrm{MV}(t) = K_p e(t) + K_i \int_0^t e(t)\,dt + K_d \frac{de}{dt}$$

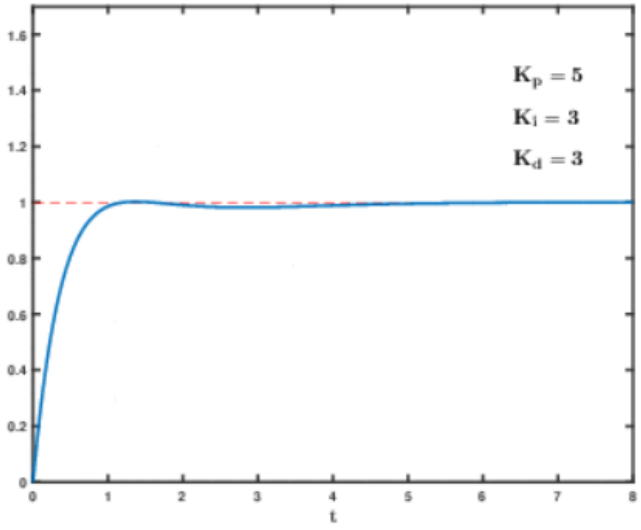




D gain 증가



I gain 증가



D gain 증가

2. 펌웨어 - 4) PID

- PID 코드

```
double PID_control_roll(angle* current_value, angle* target_value)
{
    static double PID;

    error_roll = current_value->roll - target_value->roll;

    Py_roll = error_roll;
    Iy_roll = Iy_roll + error_roll * const_dt;
    Dy_roll =  (error_roll - pre_error_roll) / const_dt;

    PID = (Pgain_roll * Py_roll) + (Igain_roll * Iy_roll) + (Dgain_roll * Dy_roll);

    pre_error_roll = error_roll;

    return PID;
}
```

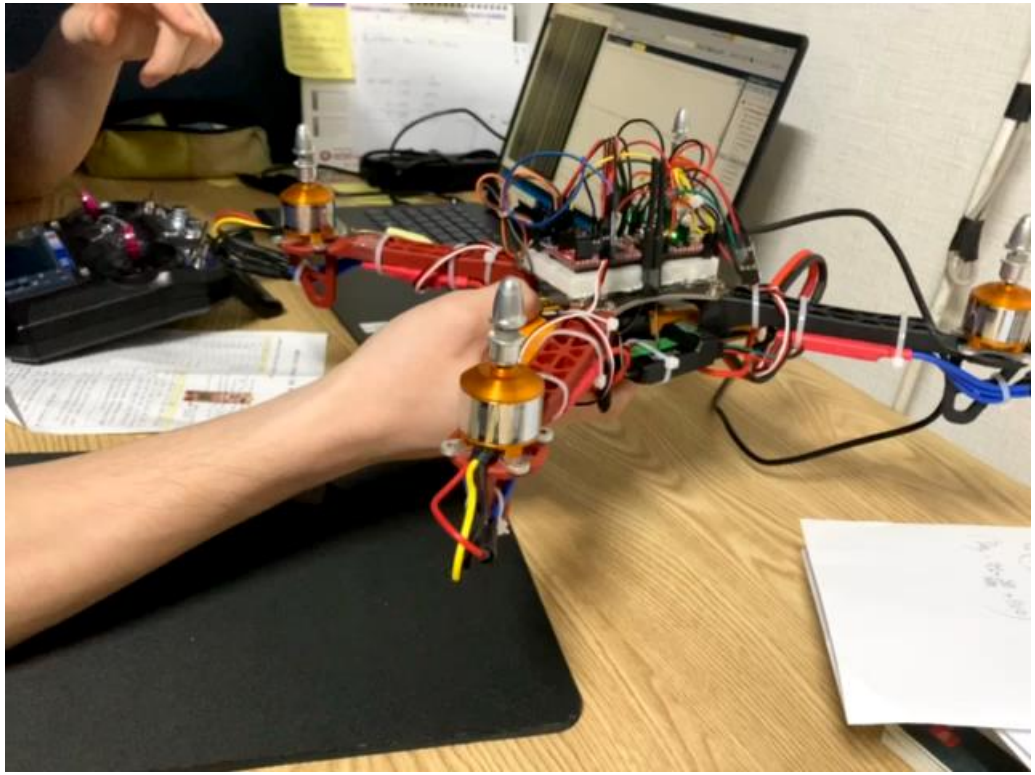
<main.c>

```
target_update(&channel, &target_angle);

PID_roll = PID_control_roll(&Fil_angle, &target_angle, &accel_gyro);
PID_pitch = PID_control_pitch(&Fil_angle, &target_angle, &accel_gyro);
PID_yaw = PID_control_yaw(&Fil_angle, &target_angle);

motor_update(&channel, PID_roll, PID_pitch, PID_yaw);
```

2. 펌웨어 - 4) PID

- PID 값의 활용

```
OCR1A = -PID_roll - PID_pitch - PID_yaw + (ch->ch3 * CH_TO_OCR);

OCR1B = PID_roll - PID_pitch + PID_yaw + (ch->ch3 * CH_TO_OCR);

OCR3A = PID_roll + PID_pitch - PID_yaw + (ch->ch3 * CH_TO_OCR);

OCR3B = -PID_roll + PID_pitch + PID_yaw + (ch->ch3 * CH_TO_OCR);
```



2. 펌웨어 - 4) PID

- P gain 값에 따른 드론 반응성 테스트

<P_gain =2>

<P_gain = 6>

2. 펌웨어 - 4) PID

- 이중 PID제어

```
double PID_control_roll(angle* current_value, angle* target_value, accel_t_gyro* accel_gyro)
{
    static double PID;

    double Gx = (double)(accel_gyro->raw.x_gyro) / FS_SEL3;

    error_roll = current_value->roll - target_value->roll;
    error_rate = (error_roll * Pgain_roll_outer) - Gx;

    Py_roll = error_rate;
    Iy_roll = Iy_roll + error_rate * DT_;
    Dy_roll =  (error_rate - pre_error_roll) / DT_;

    PID = (Pgain_roll * Py_roll) + (Igain_roll * Iy_roll) + (Dgain_roll * Dy_roll);

    pre_error_roll = error_roll;

    return PID;
```

(현재 진행중~)

2. 펌웨어 – 5) 트러블 슈팅

① 2개의 인터럽트 서비스 루틴의 충돌

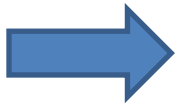
- 수신기 인터럽트와 센서 수신 인터럽트를 동시에 실행 할 때 수신기 데이터 값이 불안정하게 튀는 현상이 발생

- 해결하기 위한 시도
  a. 외부전원 : 컴퓨터의 USB를 통해 MCU, MPU6050, 수신기에 전원을 동시에 연결했기 때문에 전원 문제일 것이라고 판단하고 외부전원을 인가했지만 해결되지 않음.
  b. 센싱 주기 변경 : PPM파형의 인터럽트 발생 주기가 0.4ms~1.6ms이고, MPU6050은 2ms 인터럽트를 이용하기 때문에 데이터를 받는 속도가 너무 빠르기 때문이라고 판단하고 PWM파형을 수신했고 MPU6050 주기를 10ms로 바꿨지만 튀는 현상이 조금 개선되었을 뿐 해결되진 않음
  c. ATmega328p로 테스트 : Input Capture기능을 사용하여 수신기 데이터를 받으니까 MPU6050과 같이 사용해도 안정적으로 나옴. 따라서 수신기 데이터를 받을 때 사용하는 외부 인터럽트의 문제라고 판단됨
  d. MPU9250 SPI통신 사용 : SPI통신 하면서 외부인터럽트로 수신기 데이터 파싱 받아도 값이 튀는 문제는 해결되지 않음
  e. 인터럽트 안에서 모든 연산을 처리하지 말고 플래그만 활용하고 main문에서 연산함으로써 해결함(bottom half)

2. 펌웨어 – 5) 트러블 슈팅

- 기존 코드 : ISR안에 센서 값을 받아오고 연산하는 모든 처리를 수행함

```
ISR (TIMER0_COMP_vect)
{
    get_accel_gyro_raw(&accel_gyro);
    mpu6050_run(&accel_gyro, &Fil_angle);
}
```



- 수정한 코드 : ISR에는 flag만 설정하고 센서 값을 받아서 연산하는 과정은 main문 안에서 수행함

```
ISR (TIMER0_COMP_vect)
{
    MPU_flag=1;
}

while (1)
{
    if(MPU_flag == 1)
    {
        cbi(TIMSK,OCIE0);

        get_accel_gyro_raw(&accel_gyro);
        mpu6050_run(&accel_gyro, &Fil_angle);

        sbi(TIMSK,OCIE0);

        MPU_flag = 0;
    }
```

- **이유** : 수신기 데이터를 인터럽트로 처리하는 상황이었고 MPU를 위한 인터럽트 안에 너무 많은 연산이 들어 있기 때문에 인터럽트가 중첩되어 값을 보장할 수 없는 상황이었다고 판단됨.

2. 펌웨어 – 5) 트러블 슈팅

② 수신 받은 데이터를 채널에 할당 할 때 값이 밀리는 현상 발생

- 해결방법
  - 9번째 채널 값의 범위는 4600~10500us 범위를 갖는다. 이 채널의 Duty값이 4000이상인 값이 나왔을 때 이 값은 무조건 9번 채널에 할당시키고 그 뒤의 채널들을 차례대로 파싱하는 코드를 추가했다.

```
//채널이 밀리지 않고 파싱되게끔 해주는 코드
if(duty > 4000) //마지막 잉여 채널은 4600~10500us 범위를 가짐
{
    rising_cnt = 9;
    falling_cnt = 9;
}
```

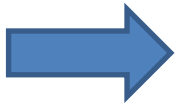
```
BLDCmotor init
Rceiver init
CH1 : 1100  CH2 : 1100  CH3 : 0  CH4 : 0
CH5 : 600  CH6 : 600  CH7 : 1100  CH8 : 1100  CH9 : 9099
CH1 : 1103  CH2 : 1100  CH3 : 603  CH4 : 1095
CH5 : 600  CH6 : 600  CH7 : 1100  CH8 : 1100  CH9 : 9099
UART init
I2C init
BLDCmotor init
Rceiver init
CH1 : 1103  CH2 : 0  CH3 : 0  CH4 : 0
CH5 : 600  CH6 : 600  CH7 : 1100  CH8 : 1100  CH9 : 9099
CH1 : 1103  CH2 : 1100  CH3 : 603  CH4 : 1095
CH5 : 600  CH6 : 600  CH7 : 1100  CH8 : 1100  CH9 : 9099
UART init
I2C init
BLDCmotor init
Rceiver init
CH1 : 600  CH2 : 600  CH3 : 1100  CH4 : 1099
CH5 : 600  CH6 : 600  CH7 : 1100  CH8 : 1099  CH9 : 9099
CH1 : 1103  CH2 : 1100  CH3 : 603  CH4 : 1095
CH5 : 600  CH6 : 600  CH7 : 1100  CH8 : 1099  CH9 : 9099
```

2. 펌웨어 – 5) 트러블 슈팅

③ I2C 통신이 중간에 끊기는 현상

- 해결방법 : 통신 에러 발생 시 재귀함수 호출을 통해 통신을 계속 시도하도록 코드를 수정함.

```
uint8_t i2c_SLA_W(uint8_t address)
{
    //SLA+W
    TWDR = (address<<1) | WRITE;
    TWCR = (1<<TWINT) | (1<<TWEN);
    while(!(TWCR & (1<<TWINT)));

    if((TWSR & 0xF8) != MT_SLA_W_ACK)
    {
        printf("%d\n", (TWSR & 0xF8));
        printf("MT_SLA_W_ACK error\n");
        return 1;
    }
    return 0;
}
```



```
uint8_t i2c_SLA_W(uint8_t address)
{
    //SLA+W
    TWDR = (address<<1) | WRITE;
    TWCR = (1<<TWINT) | (1<<TWEN);
    while(!(TWCR & (1<<TWINT)));

    if((TWSR & 0xF8) != MT_SLA_W_ACK)
    {
        printf("%d\n", (TWSR & 0xF8));
        printf("MT_SLA_W_ACK error\n");
        i2c_SLA_W(address);
    }
    return 0;
}
```

2. 펌웨어 – 5) 트러블 슈팅

④ Atmega328p -> Atmega128 MCU 변경

- 기존의 MCU는 16비트 T/C 채널이 2개이다. 8비트 T/C로 모터를 제어하기에는 분해능이 떨어지기 때문에 6개 채널의 16비트 T/C가 있는 Atmega128로 MCU를 교체했다

⑤ 타이머/카운터 인터럽트를 잘못 사용하고 있었음.

- 기존에 Normal모드로 설정하고 TCNT값을 조절해 2ms 인터럽트를 만들었는데 이렇게 하면 ISR에서 TCNT값을 계속 갱신해주어야 한다는 걸 모르고 사용해왔음

```
TCCR0 = 0x06; //Normal Mode, 분주비 256 -> 1계수 당 16us
sbi(TIMSK, TOIE0); //Overflow Interrupt Enable
TCNT0 = 131; //256 - 125 = 131 -> 125번 계수하면 16us*125 = 2ms
```

- 따라서 CTC모드로 설정한 뒤 OCR값을 조절해 2ms를 만들어주는 방식으로 바꿈.

```
TCCR0 |= (1<<WGM01)|(1<<CS01)|(1<<CS02);//CTC모드, 분주비 256 -> 1계수 당 16us
OCR0 = 124  ;   //16us * 125 = 2ms
TIMSK |= 0x02; //Output Compare Interrupt Enable
```

OCR도달 시 toggle하는 PWM을 설정한 다음 오실로스코프로 2ms마다 동작하는지 확인해 봄

앞으로 할 일

1. 이중 루프 PID를 이용한 자세 제어
2. Yaw 제어 추가
3. 32bit MCU 이용
4. 고도 센서를 이용한 호버링
5. RC카 프로젝트

마무리

그동안 감사했습니다. 많이 배웠습니다!