임베디드 시스템 설계/2022

사물 tracking 3축 짐벌

최종 보고서

2022. 6. 19

인하대학교 정보통신공학부
윤대민, 김태완, 이재영

인하대학교

연구목표

- 제목: **사물 tracking 3축 짐벌**

- 주요내용

- 사용자의 움직임으로 인한 카메라의 움직임을 보상하여, 카메라가 3차원의 특정 지점에서 움직이지 않도록 보조하는 장치를 구현한다.

- 카메라가 물체를 tracking하는 경우, 위 기능을 위한 데이터에, tracking으로 인한 카메라의 움직임을 반영하여, tracking과 짐벌이 동시에 동작할 수 있는 알고리즘을 구현하고 검증한다.

짐벌 시스템 예시

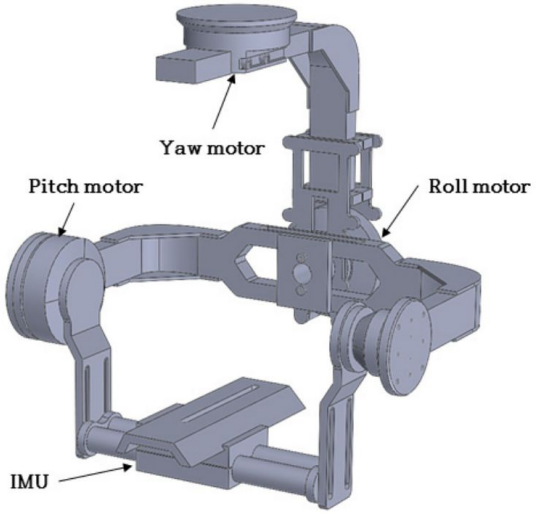
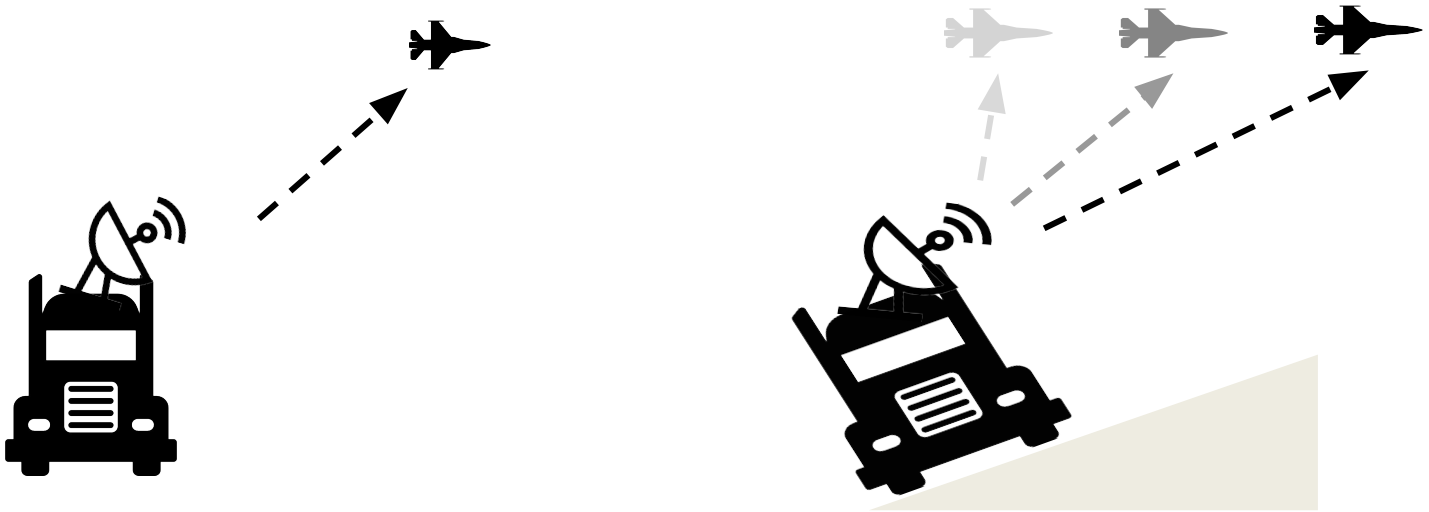


Fig. 2 The Gimbal system Architecture

Fig. 1 The Gimbal system

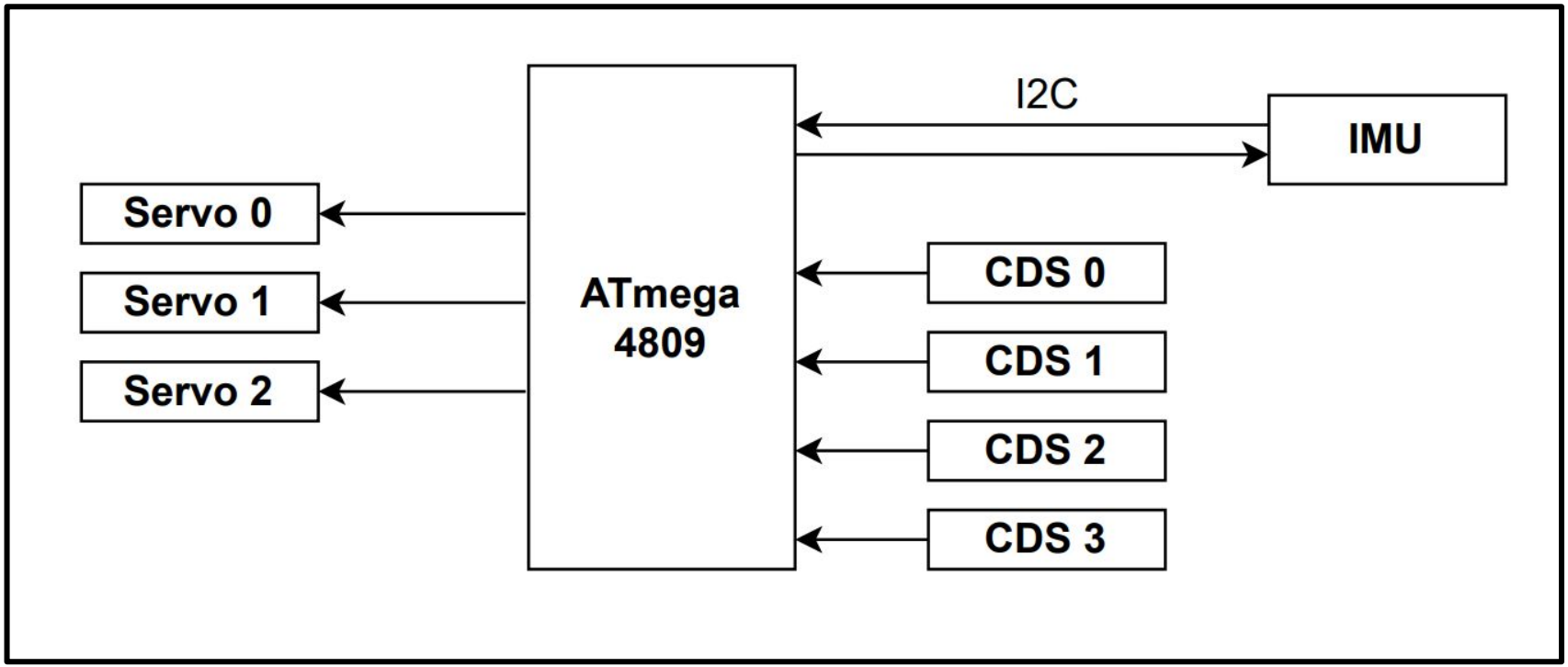
Source: Min Kim, Gi-Sig Byun, Gwan-Hyung Kim, "입체영상 촬영을 위한 드론용 짐벌시스템 제어", *J.Korea Inst. Inf. Commun. Eng.* vol.20, No.11, pp. 2107-2112, Nov. 2016

시스템 적용 예시

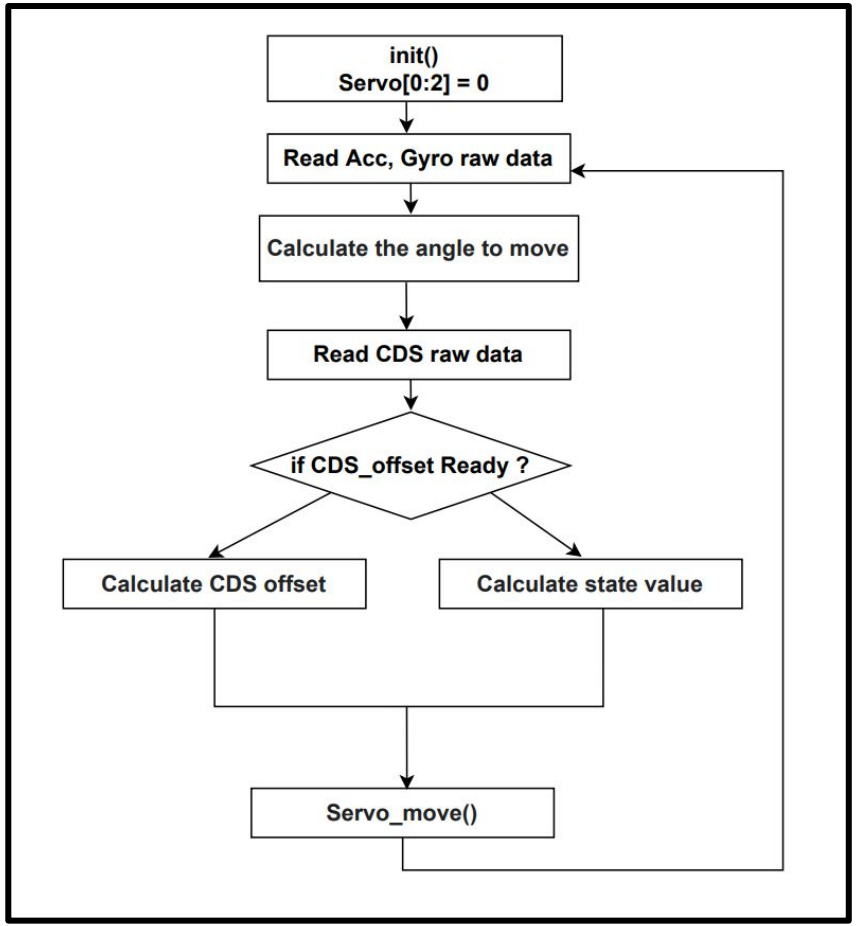


특정 타겟을 조준하고 있는 장비가 이동 중인 경우에도 조준을 유지할 수 있도록 한다.

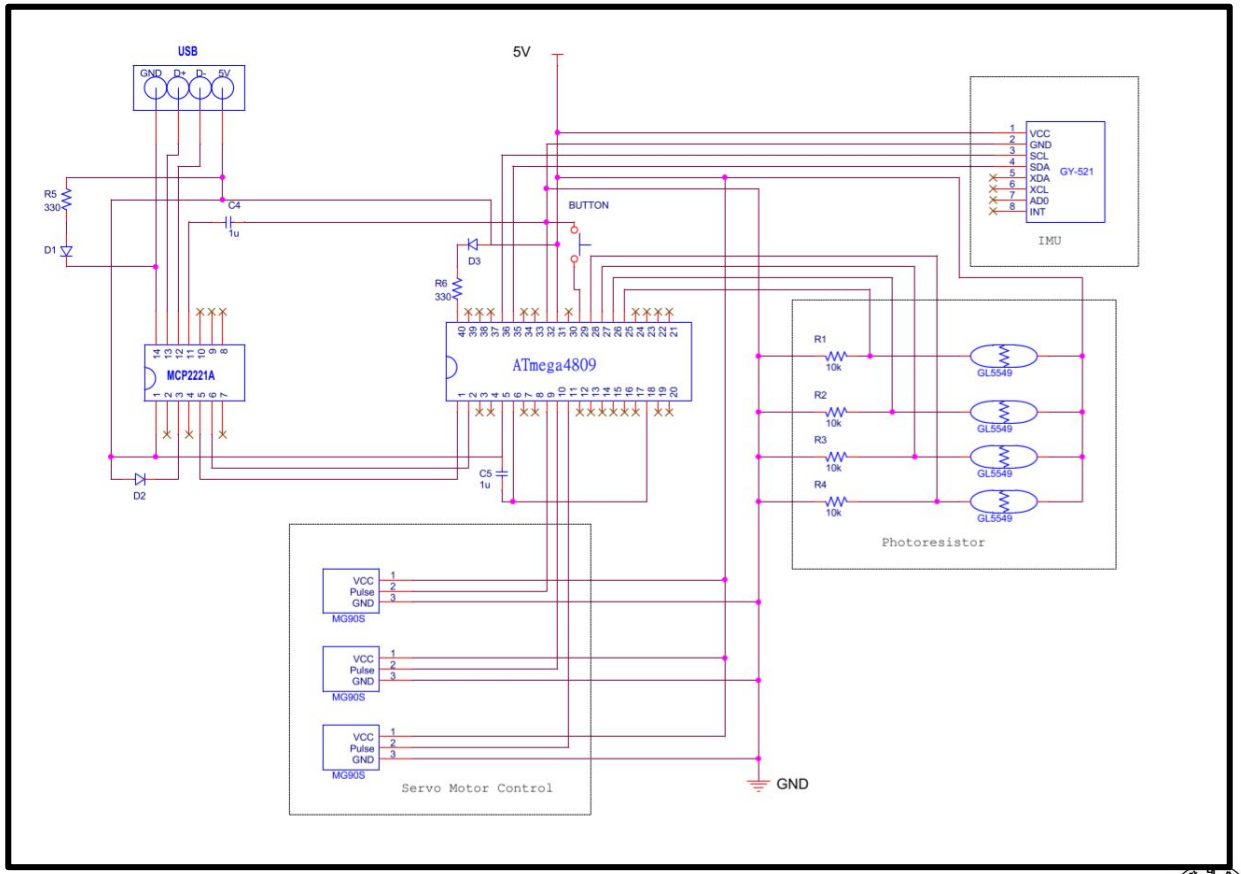
시스템 구성 : Block Diagram



시스템 구성 : Flow-Chart

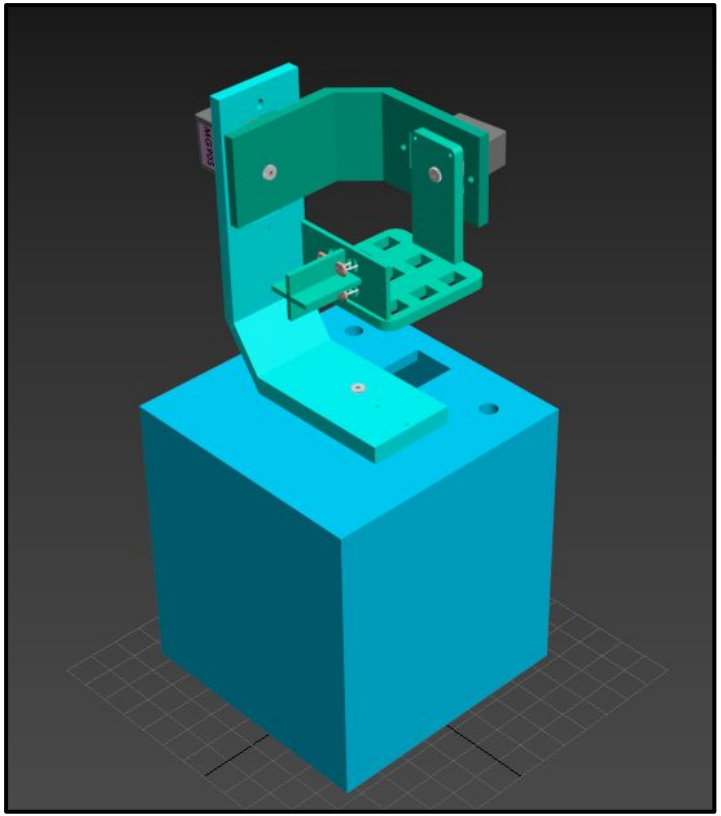


시스템 구성 : 회로도

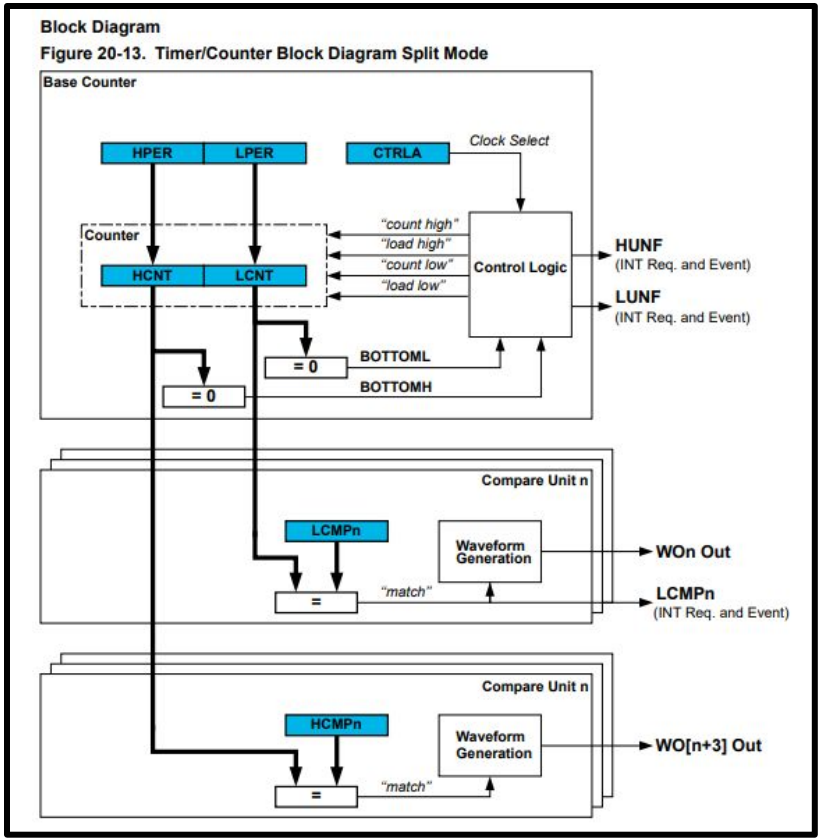

USB
GND
D+
D-
5V
5V
R5
330
D1
C4
1u
D2
MCP2221A
BUTTON
D3
R6
330
ATmega4809
C5
1u
VCC
GND
SCL
SDA
XDA
XCL
AD0
INT
GY-521
IMU
R1
R2
R3
R4
10k
GL5549
Photoresistor
VCC
Pulse
GND
MG90S
Servo Motor Control
GND

주요 부분 : 몸체

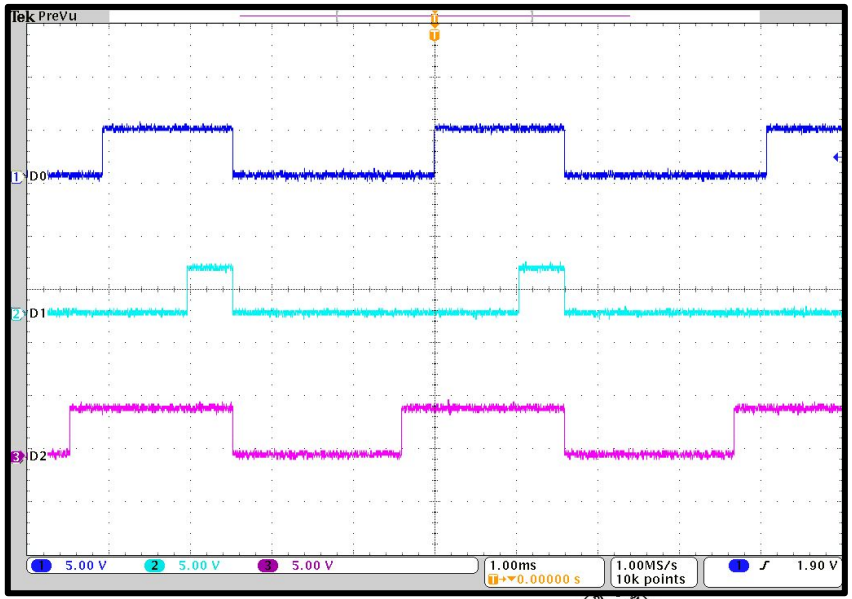
- **센서 측정값 수신**
  - IMU: 가속도, 각속도 측정값을 수신.
    인터페이스: I2C
  - 조도센서: 상하좌우 4방향에 해당하는 조도를 측정
- **데이터 취합 및 계산**
  - IMU의 측정값으로 Roll, Pitch, Yaw 정도를 계산
  - CDS의 상/하, 좌/우 중 조도가 높은 방향(tracking 방향) 결정
  - 기울기 보상을 위한 데이터와, tracking 을 위한 데이터를 계산하여 Servo의 이동 각도 산출
- **서보모터 제어**
  - 8-bit PWM



주요 기능 구현 : PWM



TCA
SPLIT MODE
Single-Slope PWM



source : Microchip Technology Inc. Atmega4808/4809 Data Sheet

주요 기능 구현 : PWM

20.6 Register Summary - TCAn in Split Mode

Offset	Name	Bit Pos.								
0x00	CTRLA	7:0					CLKSEL[2:0]			ENABLE
0x01	CTRLB	7:0		HCMP2EN	HCMP1EN	HCMP0EN		LCMP2EN	LCMP1EN	LCMP0EN
0x02	CTRLC	7:0		HCMP2OV	HCMP1OV	HCMP0OV		LCMP2OV	LCMP1OV	LCMP0OV
0x03	CTRLD	7:0								SPLITM
0x04	CTRLECLR	7:0					CMD[1:0]		CMDEN[1:0]	
0x05	CTRLESET	7:0					CMD[1:0]		CMDEN[1:0]	
0x06 ... 0x09	Reserved									
0x0A	INTCTRL	7:0		LCMP2	LCMP1	LCMP0			HUNF	LUNF
0x0B	INTFLAGS	7:0		LCMP2	LCMP1	LCMP0			HUNF	LUNF
0x0C ... 0x0D	Reserved									
0x0E	DBGCTRL	7:0								DBGRUN
0x0F ... 0x1F	Reserved									
0x20	LCNT	7:0	LCNT[7:0]							
0x21	HCNT	7:0	HCNT[7:0]							
0x22 ... 0x25	Reserved									
0x26	LPER	7:0	LPER[7:0]							
0x27	HPER	7:0	HPER[7:0]							
0x28	LCMP0	7:0	LCMP[7:0]							
0x29	HCMP0	7:0	HCMP[7:0]							
0x2A	LCMP1	7:0	LCMP[7:0]							
0x2B	HCMP1	7:0	HCMP[7:0]							
0x2C	LCMP2	7:0	LCMP[7:0]							
0x2D	HCMP2	7:0	HCMP[7:0]							

```
void TCA_reset() {
  *TCA_CTRLA = 0;
  *TCA_CTRLESET = B00001100;
  *TCA_LCNT = 0;
  *TCA_LCMP0 = 0;
  *TCA_LCMP1 = 0;
  *TCA_LCMP2 = 0;
}
void TCA_init() {
  *PORTMUX_TCA = 0x3;
  *TCA_LPER = 0xFF;
  *TCA_CTRLA = B00001101;
  *TCA_CTRLB = B01110111;
  *TCA_CTRLD = B00000001;
}
```

```
int calc_angle(int angle) {
  if (angle < -90) angle = -90;
  else if (angle > 90) angle = 90;

  int mov = map(angle, -90, 90, 35, 125);
  return mov;
}
```

source : Microchip Technology Inc. Atmega4808/4809 Data Sheet

주요 기능 : Tracking

- 조도 센서를 이용해서 광원의 위치를 파악한다.
  –상하좌우 4개의 조도센서를 사용하며, 각 조도센서로부터의 input이 동일해지도록 모듈을 제어한다.

- 광원 방향으로 이동하기 위해 서보모터에 전달해야 하는 값을 산출한다.

- 자세 제어를 위해 계산한 데이터에 광원 트래킹을 위한 데이터를 더해줌으로써 두 종류의 기능이 동시에 동작하도록 한다.

주요 기능 구현 : ADC

```
void ADC_Read_Value() {
  int addr = 0x0C;
  int16_t tmp[4];

  for (int i = 0; i < 4; i++) {
    ADC0_MUXPOS = addr + i;
    delayMicroseconds(400);
    while (!(ADC0_INTFLAGS & PIN0));
    delayMicroseconds(400);
    tmp[i] = ADC0_RESL;
  }
  delay(100);


  L_top = tmp[0];
  R_top = tmp[1];
  L_bottom = tmp[2];
  R_bottom = tmp[3];
}
```

```
void ADC_init(void) {
  PORTF_PIN2CTRL = 0X04;
  PORTF_PIN3CTRL = 0X04;
  PORTF_PIN4CTRL = 0X04;
  PORTF_PIN5CTRL = 0X04;

  ADC0_CTRLA |=  ADC_RESSEL_8BIT | ADC_ENABLE | ADC_FREERUN;
  ADC0_CTRLC |= ADC_PRESC_DIV16_gc;
  ADC0_COMMAND |= ADC_STCONV;
}
```

ADC의 MUXPOS Register를 이용하여 ADC 4개 채널 활용

- Atemga4809의 AIN12 ~ AIN15를 사용
  => 4개 CDS (GL5537)의 raw data 확인

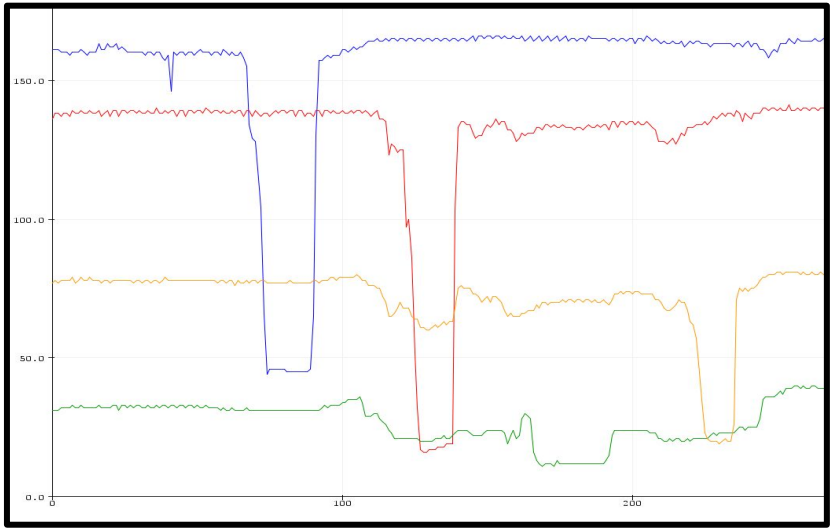
- MUXPOS로 채널이 바뀌면서 settling time 필요
  => 400us의 delay 부여

- 8bit resolution으로 0 ~ 255 값 출력

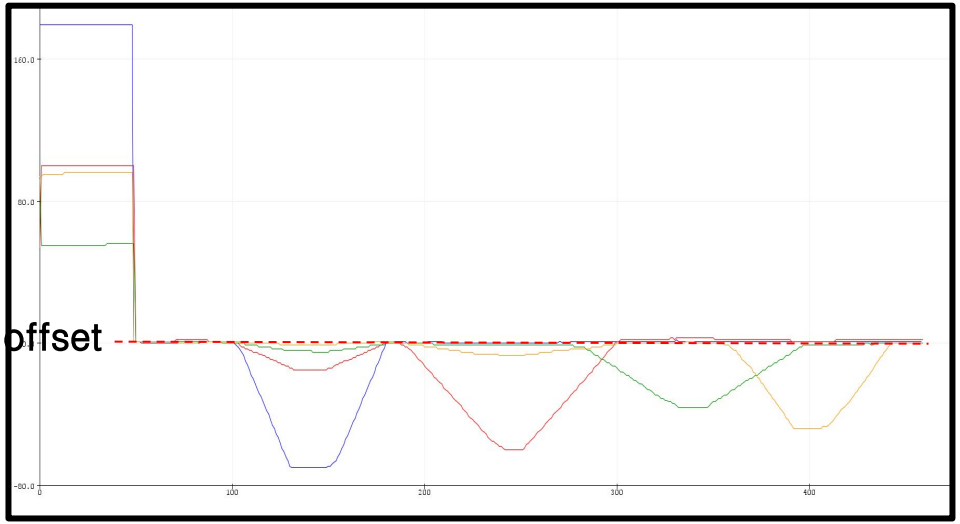
- ADC의 최대 분해능을 위해 CLK_PER를 16분주
  (16MHz / 16 = 1MHz의 ADC Clock)

- Free Run Mode를 이용
  => conversion이 끝나자마자 새로운 conversion 시작

주요 기능 구현 : ADC



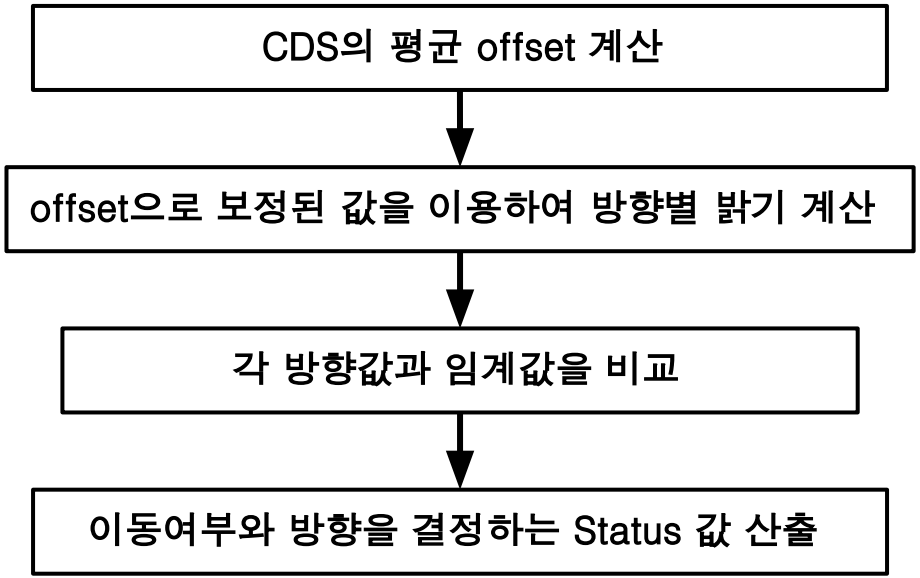
CDS Raw data



offset 보정하고 시험한 결과

주요 기능 구현 : ADC

환경적 요인을 고려한 raw data 보정



주요 기능 구현 : MPU6050

1. **TWI_init**

   MCTRLA: twi_enable

   MSTATUS: make bus idle

Bit	7	6	5	4	3	2	1	0
	RIEN	WIEN		QCEN	TIMEOUT[1:0]		SMEN	ENABLE
Access	R/W	R/W		R/W	R/W	R/W	R/W	R/W
Reset	0	0		0	0	0	0	0

```
void twi_init () {

  uint32_t baud = ((F_CPU/FREQUENCY) - (((F_CPU*T_RISE)/1000)/1000)/1000 - 10)/2;
  *TWI0_MBAUD = (uint8_t)baud;
  //*TWI0_MCTRLA = TWI_ENABLE_bm | TWI_RIEN_bm | TWI_WIEN_bm;
  *TWI0_MCTRLA = TWI_ENABLE_bm;// Enable as master, no interrupts
  *TWI0_MSTATUS = TWI_BUSSTATE_IDLE_gc;
}
```

2. **TWI_start**

   MADDR에 address <<1 | (RW') 를 입력하면, MDATA를 거쳐 bus에 할당

   polling 방식으로 WIF, RIF flag set을 기다림.

   슬레이브로부터 ACK 수신을 감지하면 통신 종료

주요 기능 구현 : MPU6050

3. TWI_Write

start와 동일하게 전송하고자 하는 데이터를 mdata에 저장

MCTRLB에 통신 지속을 위해 TWI_MCMD_RECVTRANS 저장

4. TWI_Read

Read(0)은 데이터 read후 ACK를 전송. Read(1)은 데이터 read 후 NACK를 전송.

ACK 전송시 MCTRLB에 TWI_MCMD_RECVTRANS_gc 저장.

NACK 전송시 통신 종료를 위해 MCTRLB에 TWI_MCMD_STOP_gc 저장.

주요 기능 구현 : MPU6050

5. Sensor setting

*0x1A = 0x06 : 내장 LPF 사용

*0x1B = 0x08 : Scale range of gyro (±500°/s)

*0x1C = 0x00 : Scale range of accelerometer (±2g)

```
Acc_x = ((float)raw_Acc_x) / 16384.0;
Acc_y = ((float)raw_Acc_y) / 16384.0;
Acc_z = ((float)raw_Acc_z) / 16384.0;
```

Output in two's complement format		16
AFS_SEL=0		16,384

```
Gyro_x = ((float)raw_Gyro_x- (Gyro_x_offset)) / 65.5;
Gyro_y = ((float)raw_Gyro_y - (Gyro_y_offset)) / 65.5;
Gyro_z = ((float)raw_Gyro_z - (Gyro_z_offset)) / 65.5;
```

FS_SEL=1		65.5

source : InvenSense Inc. MPU-6000/MPU-6050 Product Specification

주요 기능 구현 : MPU6050

6. 가속도 및 각속도 센서 값 수신

Burst read Sequence

해당 기능을 사용하여 0x3B로부터 Acx, Acy, Acz, temp, Gyx, Gyy, Gyz

총 14 바이트를 연속해서 Read할 수 있도록 작성

Burst Read Sequence

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

```
I2C_0_start(0x68, I2C_DIRECTION_BIT_WRITE);
I2C_0_writingPacket(0x3B);
I2C_0_start(0x68, I2C_DIRECTION_BIT_READ);
raw_Acc_x = (I2C_0_receivingPacket(0) << 8) | I2C_0_receivingPacket(0);
raw_Acc_y = (I2C_0_receivingPacket(0) << 8) | I2C_0_receivingPacket(0);
raw_Acc_z = (I2C_0_receivingPacket(0) << 8) | I2C_0_receivingPacket(0);
raw_temp = (I2C_0_receivingPacket(0) << 8) | I2C_0_receivingPacket(0);
raw_Gyro_x = (I2C_0_receivingPacket(0) << 8) | I2C_0_receivingPacket(0);
raw_Gyro_y = (I2C_0_receivingPacket(0) << 8) | I2C_0_receivingPacket(0);
raw_Gyro_z = (I2C_0_receivingPacket(0) << 8) | I2C_0_receivingPacket(1);
```

source : InvenSense Inc. MPU-6000/MPU-6050 Product Specification

주요 기능 구현 : MPU6050

7. Offset 계산

센서의 초기값에 오차가 있는 경우를 보정하기 위해 초기 100번의 평균값을 offset으로 설정

8. 각도 계산

가속도 센서: 중력 가속도 방향을 기준으로 Roll, Pitch 값을 얻을 수 있음

자이로 센서: 각 축을 기준으로 각속도를 적분하여 Roll, Pitch, Yaw값을 얻을 수 있음.

상보필터: 자이로센서와 가속도센서 계산 값을 상보필터에 적용하여, 정확성을 높일 수 있음.

장치의 움직임이 크지 않을 것이므로 각도 계산에는 가속도 센서값의 비중이 높은 상보필터를 사용

GyroCoef = 0.02

```
angle_x = (GyroCoef * (angle_x + angle_Gyro_x)) + ((1 - GyroCoef) * angle_Acc_x);
angle_y = (GyroCoef * (angle_y + angle_Gyro_y)) + ((1 - GyroCoef) * angle_Acc_y);
angle_z = angle_Gyro_z;
```

주요 기능 구현 : MPU6050

9. Drift Calibration

Yaw 값의 경우, 자이로센서 값만을 parameter로 사용. 따라서 누적되는 오차를 보정해 주는 알고리즘 추가

1. 정지 상태를 감지.

    yaw 데이터 50개를 저장.

    매 측정마다 데이터 변화량 49개를 저장. -> Diff 배열에 저장

    Diff[0]과 Diff[48]의 평균과 전체 Diff의 평균의 특정 값 미만인 경우, 정지 상태로 판단.

2. Diff의 평균을 Drift로 간주

3. 계산한 drift값을 계산값에 적용하여 calibration 진행

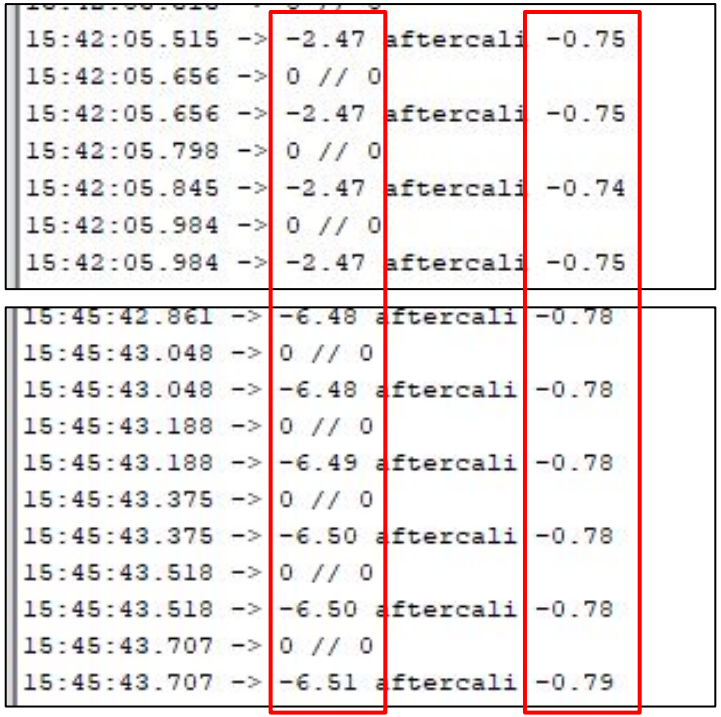
주요 기능 구현 : MPU6050

9. Drift Calibration(상세 코드)

```
//////////////////////drift calibration/////////////////////////////////
  if(driftnum == 0) {
      values[driftnum] = angle_Gyro_z;
      driftnum = 1;
    }
    else if(driftnum >= 1){
      values[driftnum] = angle_Gyro_z;
      diffs[driftnum-1] = values[driftnum] - values[driftnum-1];
      if(driftnum == 50) {
        float sum = 0;
        float avg = 0;
        float avgcal = 0;
        driftnum = 0;
        for(int k = 0; k<49; k++){
          sum = sum + diffs[k];
        }
        avg = sum/49;
        avgcal = (diffs[0]+diffs[49])/2;
        if(((avgcal - avg)*(avgcal - avg))<=0.01){
          mySerial.println("stop moving");
          drifterror = avg;
          mySerial.println(drifterror);
        }
      }
      else driftnum++;
    }
```

주요 기능 구현 : MPU6050

Drift Calibration 검증



3분간 정지 상태 유지

calibration 이 적용되지 않은 경우 drift: −4°

calibration 알고리즘 적용한 경우 drift: −0.04°

최종 기능 구현

1. 자세 제어 (짐벌)

앞서 계산된 각도 변화량만큼 반대로 서보모터를 구동함으로써 움직임을 보상해주도록 구현

최종 기능 구현

2. Tracking

4개의 input을 비교하여 Stay, move left, move right, move top, move bottom 5가지 state를 결정 하고, 계산된 각도에 Status를 반영하여 최종적으로 모터 제어.

최종 기능 구현

```
int16_t top_avg = ((L_top - L_top_offset) + (R_top - R_top_offset)) / 2;
int16_t bottom_avg = ((L_bottom - L_bottom_offset) + (R_bottom - R_bottom_offset)) / 2;
int16_t left_avg = ((L_top - L_top_offset) + (L_bottom - L_bottom_offset)) / 2;
int16_t right_avg = ((R_top - R_top_offset) + (R_bottom - R_bottom_offset)) / 2;
```

```
bool mov_vert_flag = (abs(top_avg - bottom_avg) > threshold) ? true : false;
bool mov_horiz_flag = (abs(left_avg - right_avg) > threshold) ? true : false;
```

평균값의 차이가
threshold미만인 경우
target을 바라본다고 판단

```
if (horiz_status == 1) {
  fin_z = fin_z - 1;
}
else if (horiz_status == -1) {
  fin_z = fin_z + 1;
}
else fin_z = fin_z;
if (vert_status == 1) {
  fin_x = fin_x - 1;
}
else if (vert_status == -1) {
  fin_x = fin_x + 1;
}
else fin_x = fin_x;
```

```
*TCA_LCMP0 = move_Servo(-angle_z + fin_z);
*TCA_LCMP1 = move_Servo(angle_x + fin_x);
*TCA_LCMP2 = move_Servo(-angle_y);
```

자세 변화 각도에, tracking을 위해 이동한 각도를 반영

=> 최종적으로 서보모터의 각도 결정

+

각도 유지를 위해 트래킹으로 인한 움직임을 저장

구현 중 문제점 및 고찰

1. **기기에 과도한 진동 발생**

   전기적 문제를 의심하였으나 전압, pwm signal이 모두 정상임을 확인. 특정 상태에서 무게중심이 문제가 되는 것을 확인하고 조정을 통해 보완

   – 제품을 설계 할 때 전기적인 부분 외에도 다방면으로 고려해야 함을 인식

2. **Tracking 알고리즘의 한계**

   자세 제어를 위해 움직였을 때, 축의 방향이 바뀌는 것을 고려하지 못함.

   – 각도 범위를 지정하여 상태에 따라 축을 변경해주면 개선 가능할 것으로 예상

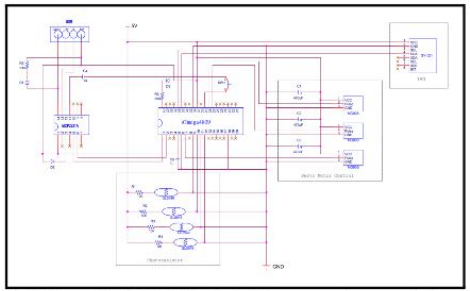
부품 리스트

부품명	규격 (mm)	모델/제조사	수량	가격(₩)	비고
IMU	21 x 11 x 1.5	GY-521/ OEM	1	3500/EA	I2C
조도센서	5.1 x 4.3	GL5549/ OEM	4	300/EA	-
서보모터	32.4 x 32.5 x 12	MG90S/ OEM	3	4200/EA	무게 : 13.4g 토크 : 1.8kgf·cm
MCU	50.8 x 15.24 x 5.06	atmega4809/ MicroChip	1	-	-
배터리	60 x 34 x 50	YJ603450/ YJ Power Group	1	5500/EA	3.7v 1000mAh 리튬폴리머배터리
만능기판	80 x 80	-	1	3000/EA	양면

총 가격: 25800원

연구일지

프로젝트 명(윤대민, 김태완, 이재영): 사물 tracking 3축 짐벌



Orcad를 이용한 회로도 구성은 위와 같다. 실습 때 이용하면 Atmega4809와 MCP2221A의 회로를 바탕으로 내부의 풀업 저항을 이용할 수 있는 GY-521를 MCU와 I2C통신으로 연결하였고, 3개의 Servo Moter를 PWM Output으로 연결하되 안정적으로 사용하기 위해 각각에 전해 축전기를 연결해주었다. 4개의 Photoresistor는 풀업 저항을 이용하여 과전류를 방지한다. 측정되는 빛의 양이 많을수록 측정되는 전압의 크기는 작아진다.

사물 tracking 3축 짐벌 | 윤대민, 김태완, 이재영

Bill Of Materials | May 14 2022

Item	Quantity	Reference	Part	Part name	Package type	가격
1	3	C1,C2,C3	Electrolyte capacitor	470uf, 16V		300
2	1	C4,C5	Capacitor	1uf		-
3	1	D1	Red LED			-
4	1	D2	Green LED			-
5	1	D3	Yellow LED			-
6	1	SW1	Push button (Switch)			-
7	6	R1,R2,R3,R4,R5,R6	Resistor	1kΩ(1), 10kΩ(5)		-
8	1	U2	IMU	GY-521		3,500
9	3	U3,U4,U5	Servo motor	MG90S		12,000
10	1	U6,U8,U9,U10	Photoresistor	GL5579		1,000
11	1	U11	Usb to Serial Converter	MCP2221A	TSSOP	-
12	1	U12	MCU	Atmega4809	40-DIP	-
13	1		Battery	18650		5,500
14	1		3축 짐벌	100x100 사각 3축 짐벌	100mm x 100mm	
15	1		3축 서보		ST, 25T,420P2	15,000
				합계가격		38,120

회로도를 바탕으로 필요한 부품을 BOM으로 정리하였으며 외형 제작에 필요한 단단한 재료를 찾아본 결과 폴리스틸렌을 발포한 것을 고밀도로 압축한 포맥스가 적합해 보였다. 패 가격이 나가지만 단단한 몸체를 유지하기 위해 최적의 재료인 듯하다.

1. ADC

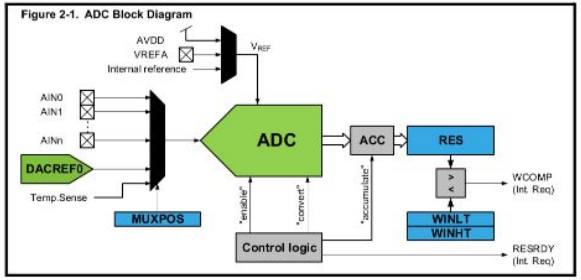
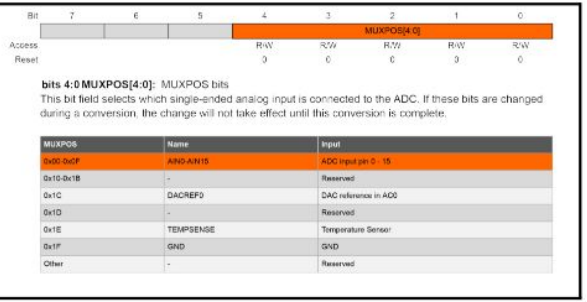


Figure 2-1. ADC Block Diagram



Bit	7	6	5	4	3	2	1	0
	MUXPOS[4:0]							
Access				R/W	R/W	R/W	R/W	R/W
Reset				0	0	0	0	0

bits 4:0 MUXPOS[4:0]: MUXPOS bits
This bit field selects which single-ended analog input is connected to the ADC. If these bits are changed during a conversion, the change will not take effect until this conversion is complete.

MUXPOS	Name	Input
0x00-0x0F	AIN0-AIN15	ADC input pin 0 - 15
0x10-0x1B	-	Reserved
0x1C	DACREF0	DAC reference in AC0
0x1D	-	Reserved
0x1E	TEMPSENSE	Temperature Sensor
0x1F	GND	GND
Other	-	Reserved

Atmega4809 ADC datasheet를 통해 MUXPOS Register로 해당하는 Analog input port와 연결할 수 있는 것을 확인하였고 이를 이용하여 4개의 CDS로 얻어 Result Register에 저장된 값을 시리얼 모니터로 확인해보고자 하였다.

```
void Temp() {
  if( ADC0_INTFLAGS & PIN0 ) // Read Ready Interrupt Enable
  {
    Ain0_result = 0x0a;
    USART1_string_send("A0 : ");
    USART1_Transmit(ADC0_RES); //ADC0_RES 7;
    delaymicroseconds(100); // 데이터 출력을 안정하기 위한
    ADC0_MUXPOS = 0x0a0;
    USART1_string_send(" A1 : ");
    USART1_Transmit(ADC0_RES);
    delayMicroseconds(100);
    ADC0_MUXPOS = 0x0b;
    USART1_string_send(" A2 : ");
    USART1_Transmit(ADC0_RES);
    delaymicroseconds(100);
    ADC0_MUXPOS = 0x0c;
    USART1_string_send(" A3 : ");
    USART1_Transmit(ADC0_RES);
    USART1_String_send("\n");
    delay(500);
  }
}
```

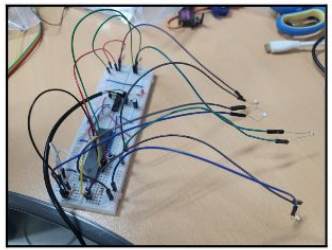
```
void loop() {
  if( ADC0_INTFLAGS & PIN0 )
  {
    ADC0_MUXPOS = 0x0a;
    mySerial.print("A0 : ");
    mySerial.print(ADC0_RES);
    delaymicroseconds(100);
    ADC0_MUXPOS = 0x0b;
    mySerial.print(" A1 : ");
    mySerial.print(ADC0_RES);
    delayMicroseconds(100);
    ADC0_MUXPOS = 0x0c;
    mySerial.print(" A2 : ");
    mySerial.print(ADC0_RES);
    delayMicroseconds(100);
    ADC0_MUXPOS = 0x0d;
    mySerial.print(" A3 : ");
    mySerial.println(ADC0_RES);
    Delay(2000);
  }
}
```

```
ADC0_CTRLA |= ADC_RESSEL_8BIT|ADC_ENABLE| ADC_FREERUN;
```

```
#define ADC_RESSEL_8BIT (0x01<<2)
#define ADC_ENABLE (0x01)
#define ADC_STCONV (0x01)
#define ADC_FREERUN (0x02)

#define PIN0 (1<<0)
#define PIN1 (1<<1)
```

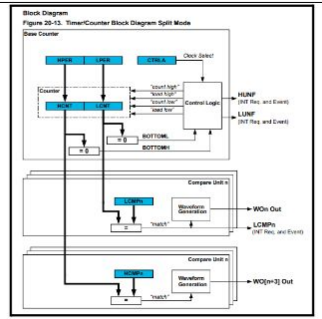
PF4 ~ PF7 핀을 이용하여 MUXPOS를 변경해가며 각각의 CDS 값을 확인할 수 있었다. 8bit 분해능과 Free Running Mode를 이용하였으며 상단 좌측의 코드는 USART를 레지스터 코딩으로 구현하여 시리얼 모니터로 확인해보았는데 char 타입의 Extended ASCII 값으로 출력되는 것을 알 수 있었다. 우측의 softwareserial 라이브러리를 이용한 코드는 정수 타입으로 값을 확인할 수 있었다. 출력하는 PIN을 변경할 때 데이터가 밀리는 것을 방지하기 위해 100us의 delay를 넣어주었다.

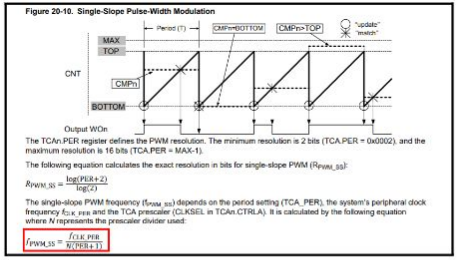


100us는 작은 delay여서 4개의 CDS로 얻는 값을 거의 동시에 확인할 수 있었으며 회로는 위의 사진과 같이 구현해보았다. 10k옴의 큰다은 저항을 추가하여 빛이 너무 밝아 CDS의 저항이 0이 되어도 과전류가 흐르는 것을 방지하였다.
4개의 CDS를 통해 값을 얻는 것을 확인하였고 이를 이용해 servo motor를 통해 tracking 기능을 구현해 볼 예정이다.

2. PWM

TCA Split Mode를 사용하면, Data sheet 상에서 확인하였을 때, 2개의 8-bit Timer/Counter 를 통해 6개의 PWM을 구현할 수 있는 것으로 확인하였고, TCA Split Mode로 아래와 같이 구현하였다.





The TCAn.PER register defines the PWM resolution. The minimum resolution is 2 bits (TCA.PER = 0x0002), and the maximum resolution is 16 bits (TCA.PER = MAX-1).

The following equation calculates the exact resolution in bits for single-slope PWM (R_{PWM_SS}):

$$R_{PWM\_SS} = \frac{\log(PER+2)}{\log(2)}$$

The single-slope PWM frequency (f_{PWM_SS}) depends on the period setting (TCA_PER), the system's peripheral clock frequency f_{CLK_PER} and the TCA prescaler (CLKSEL in TCAn.CTRLA). It is calculated by the following equation where *N* represents the prescaler divider used:

$$f_{PWM\_SS} = \frac{f_{CLK\_PER}}{N(PER+1)}$$

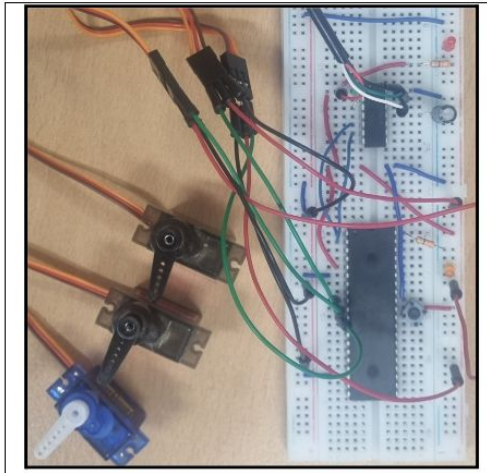
Operation은 Single-Slope PWM 을 사용하였다. 20ms 주기로 동작하게 하기 위해 위 식을 통하여 PER을 312로 계산하고 사용하였는데, 8 bit Register 이기에 Overflow로 인해 56 이 PER 값으로 들어가게 되었는데, 처음 구현에서는 이 부분을 놓치고 구현하였다.

20.6 Register Summary - TCAn in Split Mode

Offset	Name	Bit Pos.								
0x00	CTRLA	7:0					CLKSEL[2:0]			ENABLE
0x01	CTRLB	7:0		HCMP2EN	HCMP1EN	HCMP0EN		LCMP2EN	LCMP1EN	LCMP0EN
0x02	CTRLC	7:0		HCMP2OV	HCMP1OV	HCMP0OV		LCMP2OV	LCMP1OV	LCMP0OV
0x03	CTRLD	7:0								SPLITM
0x04	CTRLECLR	7:0					CMD[1:0]		CMDEN[1:0]	
0x05	CTRLESET	7:0					CMD[1:0]		CMDEN[1:0]	
0x06 ... 0x09	Reserved									
0x0A	INTCTRL	7:0		LCMP2	LCMP1	LCMP0			HUNF	LUNF
0x0B	INTFLAGS	7:0		LCMP2	LCMP1	LCMP0			HUNF	LUNF
0x0C ... 0x0D	Reserved									
0x0E	DBGCTRL	7:0								DBGRUN
0x0F ... 0x1F	Reserved									
0x20	LCNT	7:0	LCNT[7:0]							
0x21	HCNT	7:0	HCNT[7:0]							
0x22 ... 0x25	Reserved									
0x26	LPER	7:0	LPER[7:0]							
0x27	HPER	7:0	HPER[7:0]							
0x28	LCMP0	7:0	LCMP[7:0]							
0x29	HCMP0	7:0	HCMP[7:0]							
0x2A	LCMP1	7:0	LCMP[7:0]							
0x2B	HCMP1	7:0	HCMP[7:0]							
0x2C	LCMP2	7:0	LCMP[7:0]							
0x2D	HCMP2	7:0	HCMP[7:0]							

```
void TCA_init(){
  *PORTMUX_TCA = 0x3;        // PWM Output : PD[0:5]
  *TCA_LPER  = B00111000;     // Low Byte PER
  *TCA_CTRLA = B00001111;    // 0 | 0 | 0 | 0 | DIV1024 | EN
  *TCA_CTRLB = B00000111;    // 0 | HCMP[0:2]EN | 0 | LCMP[0:2]EN
  *TCA_CTRLD = B00000001;    // 0 | 0 | 0 | 0 | 0 | 0 | 0 | SPLITMODE EN
}
```

처음 구현에서 Overflow가 발생하여도 정상적으로 동작하여, PER 값을 56으로 설정하여 이와 같이 구현하였는데, 정상적으로 동작함을 확인하였다.



실제로 구현한 이후, PER 값을 잘못 계산하였기에, 주기는 우리가 원하는 20ms 로 설정되지 않았다. 오실로스코프에서 측정한 결과 한 주기는 약 3.6us 정도로 측정되었는데, 이는 Data sheet 상에서 확인한 결과로는 동작이 되지 않아야 했으나, 정상적으로 동작되어 예상하지 못한 결과였다.

구현상에서 발견한 다른 문제점들은

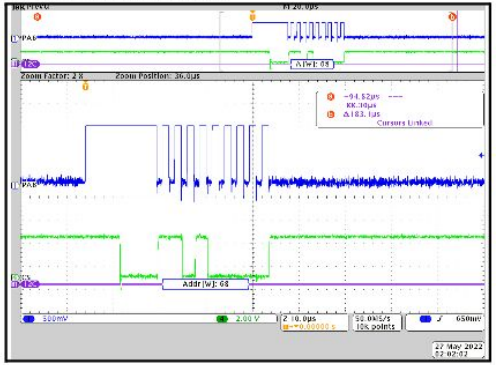
1. 3개의 Servo motor 를 연결하였을 때, 전원이 불안정한 경우 motor 가 동작하지 않는 경우가 발생한다.
2. 같은 MG-90S가 같은 Input에서 다른 동작을 수행하는 경우가 발생한다. 제조사에 따라 다른 동작을 수행하는 경우가 발생하거나, 해당 Servo motor 자체의 고장으로 예상된다.

3. I2C

i2c 프로토콜의 구성은 다음과 같다.

Read operation의 경우 Start+send slave address+write -> send register address -> restart + send slave address+read -> read data 순으로 동작한다.

오실로스코프로 확인한 동작은 다음과 같다.



start signal 후에 slave address를 버스로 잘 전송하는 것을 확인할 수 있다.

i2c 프로토콜은 크게 5개의 함수를 사용한다.

I2C_0_init()

```
void I2C_0_init(){

    TWI0_MBAUD = 30;
    TWI0_MCTRLA = B00000001;
    //TWI0_MSTATUS = B11000001;
    TWI0.MCTRLB |= TWI_FLUSH_bm;
    TWI0.MSTATUS |= (TWI_RIF_bm | TWI_WIF_bm);
    TWI0_MSTATUS |= TWI_BUSSTATE_IDLE_gc;
}
```

통신을 시작하기 전에 MBAUD 레지스터에 값을 할당해주어야 한다. MCTRLA 레지스터로 enable을 해준다. MSTATUS 레지스터 값을 지정하여, flag enable을 하고, bus state를 idle로 설정해 준다.

I2C_0_start

```
uint8_t I2C_0_start(uint8_t baseAddres, uint8_t directionBit){
    TWI0_MADDR = (baseAddres << 1) + directionBit;
    while (!(TWI0_MSTATUS & (TWI_WIF_bm | TWI_RIF_bm)));     //w
    if ((TWI0_MSTATUS & TWI_ARBLOST_bm)) return 0 ;           //s
    return !(TWI0_MSTATUS & TWI_RXACK_bm);                    //s
}
```

MADDR에 값을 넣어주면 MDATA를 통해 bus로 전달된다. start를 위해서는 slave address와 write 를 start로 전송해주고, slave로부터 ack를 전달받은 경우에 while을 빠져나온다.

I2C_0_writingPacket

```
uint8_t I2C_0_writingPacket(uint8_t data)
{
    while (!(TWI0_MSTATUS & TWI_WIF_bm));
    TWI0_MDATA = data;
    TWI0_MCTRLB = TWI_MCMD_RECVTRANS_gc;
    return (!(TWI0_MSTATUS & TWI_RXACK_bm));
}
```

write 명령은 write 동작이 실행되면 mdata에 값을 넣어줌으로써 bus로 전달된다.

I2C_0_receivingPacket

```
uint8_t I2C_0_receivingPacket(uint8_t acknack)       // 0 -> ack, else nack
{
    while (!(TWI0_MSTATUS & TWI_RIF_bm));                //wait for read interrupt flag
    uint8_t data = TWI0_MDATA;
    if (acknack == 0) {TWI0_MCTRLB = (TWI_ACKACT_ACK_gc  | TWI_MCMD_RECVTRANS_gc); }
    else              {TWI0_MCTRLB = (TWI_ACKACT_NACK_gc | TWI_MCMD_RECVTRANS_gc); }
    return data;
}
```

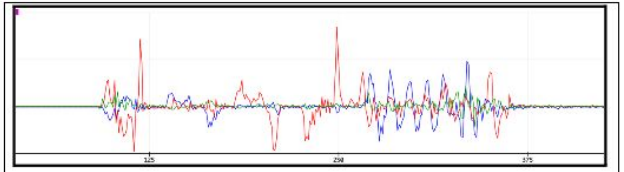
기능은 다음과 같다. MDATA 레지스터에 들어있는 값을 내부 변수 data에 입력한다. 이때 이 함수의 입력이 0인 경우에는 수신 후 버스에 ack를 전달한다. 이 함수의 입력이 1인 경우에 수신 후 버스에 nack를 전달한다.

Burst Read Sequence

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

위 사진은 accx, accy, accz, temp, gyrox, gyroy, gyroz 를 읽어오기 위한 i2c burst read sequence 프로토콜이다. 연달아 값을 읽어오기 위해 읽어온 후 master가 ack를 전송하고, 마지막 수신일 경우에만 nack를 전송한다. 해당 프로토콜에 맞춰 코드를 작성하였고, 값이 출력되는 것을 확인하였다.

```
I2C_0_start(0x68, I2C_DIRECTION_BIT_WRITE);
I2C_0_writingPacket(0x3B);
I2C_0_start(0x68, I2C_DIRECTION_BIT_READ);
raw_Acc_x = (I2C_0_receivingPacket(0) << 8) | I2C_0_receivingPacket(0);
raw_Acc_y = (I2C_0_receivingPacket(0) << 8) | I2C_0_receivingPacket(0);
raw_Acc_z = (I2C_0_receivingPacket(0) << 8) | I2C_0_receivingPacket(0);
raw_temp = (I2C_0_receivingPacket(0) << 8) | I2C_0_receivingPacket(0);
raw_Gyro_x = (I2C_0_receivingPacket(0) << 8) | I2C_0_receivingPacket(0);
raw_Gyro_y = (I2C_0_receivingPacket(0) << 8) | I2C_0_receivingPacket(0);
raw_Gyro_z = (I2C_0_receivingPacket(0) << 8) | I2C_0_receivingPacket(1);
```



다음은 각각의 raw 데이터를 전송받아서 그래프화 시킨 예시이다. 해당 그래프는 자이로센서의 raw data이다. mpu6050이 탑재되어 있는 보드를 움직였을 때, 해당 동작에 맞춰서 변화하는 것을 확인하였다.

4. Raw data 를 이용한 각도 산출

accelerometer의 데이터로 각도를 산출하는 방법은 다음과 같다.
중력방향을 기준으로 일정한 중력 가속도가 측정이 되고 있기 때문에, x, y축으로 이 중력가속도가 분배되는 정도를 계산하여, 보드가 기울어진 각도를 산출할 수 있다.

gyroscensor의 데이터로 각도를 산출하는 방법은 다음과 같다.
해당 축을 기준으로 회전한 정도에 대한 각속도가 측정되기 때문에, 이를 바탕으로 각 축을 기준으로 회전한 정도를 계산할 수 있다.

우리가 사용하는 장치의 경우, 장치의 움직임이 일정하므로, 가속도의 변화가 크지 않을 것으로 가정하고 상보필터의 비율을 정할 때 가속도센서로부터 측정한 각도를 주로 반영하도록 하였다. 이는 자이로센서로부터 각도를 산출하는 경우 가속도센서의 드리프트가 누적되기 때문에, 오차를 최소화하기 위함이다.

```
angle_x = (GyroCoef * (angle_x + angle_Gyro_x)) + ((1 - GyroCoef) * angle_Acc_x);
angle_y = (GyroCoef * (angle_y + angle_Gyro_y)) + ((1 - GyroCoef) * angle_Acc_y);
angle_z = angle_Gyro_z;
```

이 경우 gyrocoef는 0.02로 거의 반영하지 않았다.

다음으로는 자이로센서의 드리프트를 보정하는 알고리즘을 작성하였다.
Yaw 각도의 경우 자이로센서로부터 계산할 수밖에 없는데, 이는 yaw데이터의 drift가 발생할 수밖에 없음을 의미한다. 따라서 기기가 정지할 때마다, 드리프트를 보정해주는 알고리즘을 작성하였다.

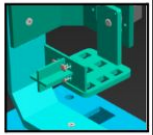
```
if(driftnum == 0) {
    values[driftnum] = angle_Gyro_z;
    driftnum = 1;
  }
  else if(driftnum >= 1){
    values[driftnum] = angle_Gyro_z;
    diffs[driftnum-1] = values[driftnum] - values[driftnum-1];
    if(driftnum == 50) {
      float sum = 0;
      float avg = 0;
      float avgcal = 0;
      driftnum = 0;
      for(int k = 0; k<49; k++){
        sum = sum + diffs[k];
      }
      avg = sum/49;
      avgcal = (diffs[0]+diffs[49])/2;
      if(((avgcal - avg)*(avgcal - avg))<=0.01){
        mySerial.println("stop moving");
        driftcror = avg;
        mySerial.println(driftcrror);
      }
    }
    else driftnum++;
  }
```

동작은 다음과 같다. 매 50 개의 데이터를 비교하고 해당 데이터의 변화량을 저장한다. 변화량이 일정하다는 것은 기기의 움직임은 없고, 드리프트에 의한 증감만 일어나고 있음을 의미한다. 따라서 정지상태에서 일정한 변화량을 드리프트로 간주하고, 해당 드리프트를 보정해줄 수 있도록 하였다.

```
15:42:05.515 -> -2.47 aftercali -0.75
15:42:05.656 -> 0 // 0
15:42:05.656 -> -2.47 aftercali -0.75
15:42:05.798 -> 0 // 0
15:42:05.845 -> -2.47 aftercali -0.74
15:42:05.984 -> 0 // 0
15:42:05.984 -> -2.47 aftercali -0.75
```

```
15:45:42.861 -> -6.48 aftercali -0.78
15:45:43.048 -> 0 // 0
15:45:43.048 -> -6.48 aftercali -0.78
15:45:43.188 -> 0 // 0
15:45:43.188 -> -6.49 aftercali -0.78
15:45:43.375 -> 0 // 0
15:45:43.375 -> -6.50 aftercali -0.78
15:45:43.518 -> 0 // 0
15:45:43.518 -> -6.50 aftercali -0.78
15:45:43.707 -> 0 // 0
15:45:43.707 -> -6.51 aftercali -0.79
```

위 스크린샷은 3분동안 정지상태에서 드리프트를 측정하고, 알고리즘 적용 여부에 따른 드리프트 양을 비교한 사진이다. 3분동안 알고리즘이 적용된 경우는 0.03의 드리프트가 측정되었고, 알고리즘을 적용하지 않은 경우 4가량의 드리프트가 측정되었다. 드리프트의 경우, 센서 측정값이 크고, 동작이 많을 수록 커지는 경향을 보이므로, 실제 동작시에는 보정 여부가 결과에 큰 영향을 끼칠 것이라고 생각한다.

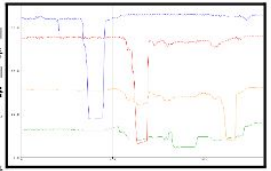


5. CDS의 Offset 조정

4개의 CDS가 하늘을 바라보는 것이 아닌 측면을 바라보도록 배치하였으며, 광원의 위치를 확실하게 인식하게 위해 2개의 가림막을 이용해 각 센서를 독립시키는 형태로 모델을 구현하였다.

광원에 특정 위치에 있다면 광원에 제일 근접한 CDS는 가장 큰 데이터를 얻을 것이고 나머지 CDS들은 가림막에 의해 그림자가 져서 낮은 데이터를 얻게 됨으로 데이터의 차이를 이용하여 광원을 따라 Tracking 하는 방식을 고민했다.

우측의 그림과 같은 배치를 하게 되어서 발생한 문제점이 설정한 광원이 없음에도 불구하고 형광등이나 자연광 등 환경적인 요인에 상단과 하단의 CDS들이 큰 차이를 유지하게 되는 것이었다. 우측의 시리얼 플로터로 CDS들의 raw data를 확인해본 결과이다. 파란색과 빨간색이 상단의 CDS들, 노란색과 초록색이 하단의 CDS들의 데이터를 보여준다.

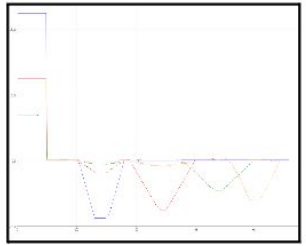


이러한 문제점을 해결하기 위한 방법으로 생각한 것이 각 CDS로 부터 초기 값들을 저장하여 이 값들의 평균을 내어 offset을 구하는 것이다. offset을 구한 이후에 받게 되는 raw data에서 offset을 빼서 어느 정도 data를 보정하는 것이다.

```
if (count < arrsize) {
    ADC_Read_Value();

    sum1 += L_top;
    sum2 += R_top;
    sum3 += L_bottom;
    sum4 += R_bottom;

    count++;
  }
  if (count == arrsize) {
    L_top_offset = sum1 / arrsize;
    R_top_offset = sum2 / arrsize;
    L_bottom_offset = sum3 / arrsize;
    R_bottom_offset = sum4 / arrsize;
    offset_completw_flag = 1;
  }
  ADC_Read_Value();
```



offset을 보정하는 데 시간이 조금 걸린다는 단점이 있기는 하지만 설정한 광원이 아닌 주위의 빛에 의한 영향을 없앨 수 있고 이를 플로터로 확인할 수 있었다. 모든 CDS가 일정한 값을 보이는 것을 볼 수 있으며 각각의 CDS가 하나씩 가려보면서 그림자가 졌을 상황을 가정했을 때 데이터의 값도 정상적으로 얻을 수 있는 지 확인해 보았다.

```
if (offset_completw_flag) {
  vert_status = 0;
  horiz_status = 0;
  int16_t top_avg = (L_top - L_top_offset) + (R_top - R_top_offset)) / 2;
  int16_t bottom_avg = ((L_bottom - L_bottom_offset) + (R_bottom - R_bottom_offset)) / 2;
  int16_t left_avg = ((L_top - L_top_offset) + (L_bottom - L_bottom_offset)) / 2;
  int16_t right_avg = ((R_top - R_top_offset) + (R_bottom - R_bottom_offset)) / 2;

  bool vert_start_flag = (abs(top_avg - bottom_avg) > threshold) ? true : false;
  bool hor_start_flag = (abs(left_avg - right_avg) > threshold) ? true : false;

  if (vert_start_flag) {
    if (top_avg > bottom_avg) vert_status = 1;
    else vert_status = -1;
  }
  if (hor_start_flag) {
    if (left_avg > right_avg) horiz_status = 1;
    else horiz_status = -1;
  }
```

정보통신공학부
임베디드시스템 설계

Offset 보정이 완료되면 flag가 생성되어 4개의 CDS를 상하좌우 방향에 따라 2개씩 묶어 각 방향에 대한 평균값을 산출하였다. 산출 값을 이용하여 각 방향의 차이가 설정한 임계값을 초과하게 되면 수직/수평 방향의 움직임에 대한 flag를 생성하게 된다.

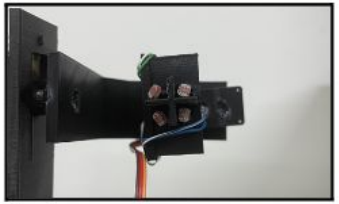
```
if (horiz_status == 1) {
  fin_z = fin_z - 1;
}
else if (horiz_status == -1) {
  fin_z = fin_z + 1;
}
else fin_z = fin_z;
if (vert_status == 1) {
  fin_x = fin_x - 1;
}
else if (vert_status == -1) {
  fin_x = fin_x + 1;
}
else fin_x = fin_x;
```

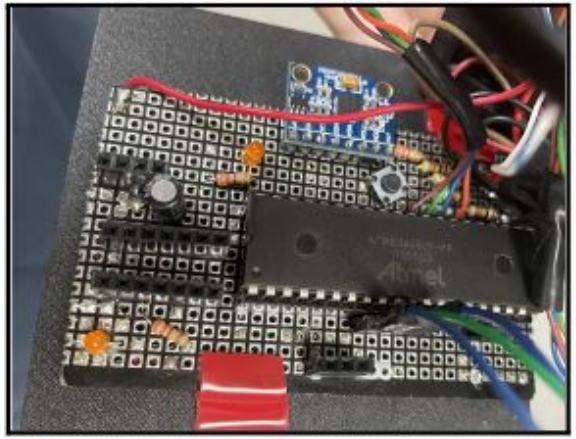
그래서 각 flag에 따라 광원이 있는 방향으로 1도씩 움직일 수 있도록 데이터를 누적시키는 코드를 구현했다.

```
*TCA_LCMP0 = move_Servo(-angle_z + fin_z);
*TCA_LCMP1 = move_Servo(angle_x + fin_x);
*TCA_LCMP2 = move_Servo(-angle_y);
```

이렇게 해서 산출한 tracking에 필요한 각도를 자세 제어를 위한 각도에 반영하여 두 가지 기능을 함께 구현하도록 했다.

6. 회로 제작 및 모델 조립





완성한 장치는 다음과 같다. 헤드 부분에 cds 4개를 부착하였고, 각각의 데이터 판단이 용이하도록 가림막을 통해 확실히 분리해 주었다.

만능기판은 장치 내부에 들어갈 수 있는 사이즈로 제작하였다.

최종적으로 구현 목표를 만족하도록 동작하는 것을 확인하였다.