LG Reskilling 과정 2023년 1차

프로젝트 결과보고서

Cortex M3 기반 AGV 설계

*AGV(Automated Guided Vehicle)

Embedded System Path

리빙솔루션 홍길동

목차구성 CONTENTS COMPOSITION

1. 주제 및 결과 요약

○ Project 주제 선정 배경

- 현업 제품군(리빙솔루션) 고려, 최대한 다양한 외부장치를 제어하는 주제 선정
- 외부 센서 감지값 기반하여, 구동 및 servo 모터를 직접 제어할 수 있는 본 주제로 선정

*LG 세탁기의 핵심부품: HEDD 모터

○ Project 목표

- 다양한 인터페이스(I^2C, ADC, PWM)등을 통한 외부장치 직접 제어
- Sensor류 외부장치에서 전송받은 data를 기반으로, 차량의 동작제어
  1) 정해진 구역 안에서, 장애물을 피해 이동하는 동작 logic 구현(로봇청소기)
  2) 정해진 유도선을 따라 주행하며, 장애물 감지 시 정지 및 추가 입력 대기(AGV)

*자율주행 차의 핵심부품: DC 모터

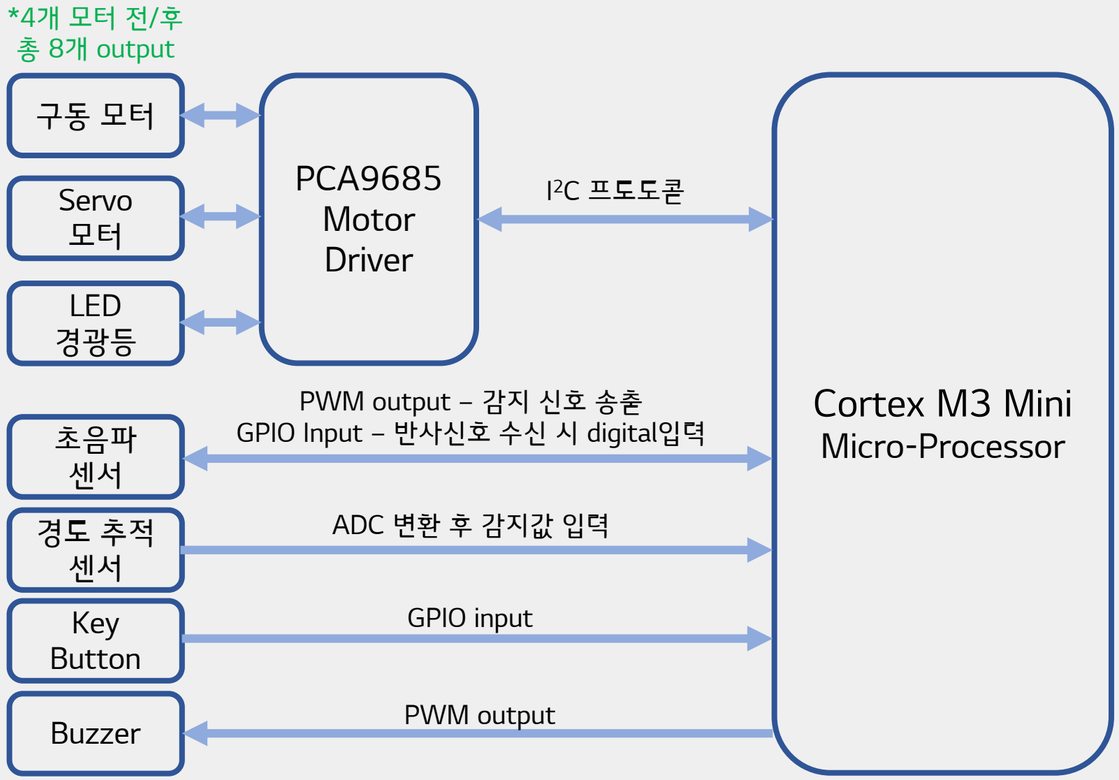
○ Project 진행 결과

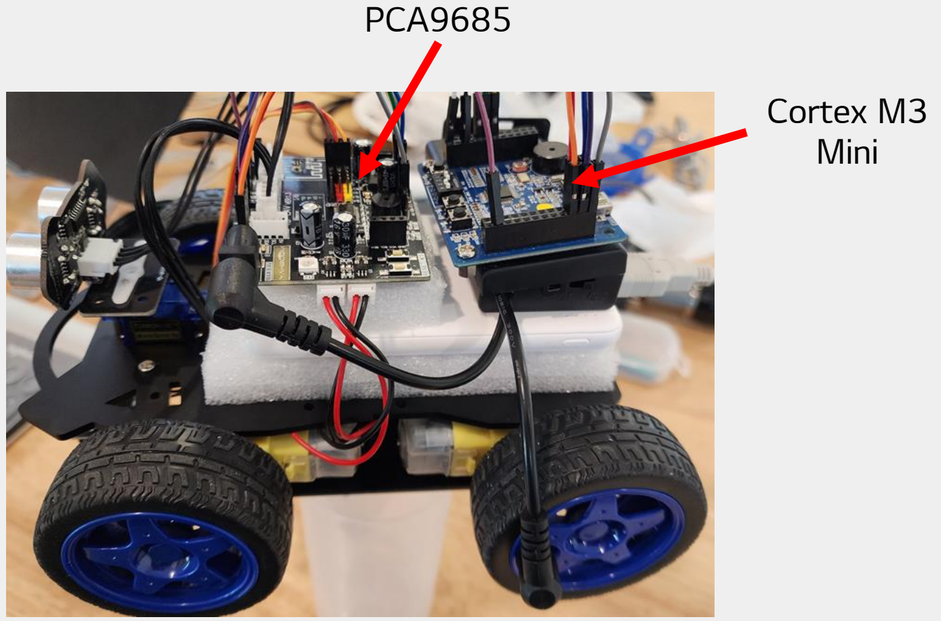
- ADC(Analog to Digital Conversion)을 통한 신호입력 구현
- I^2C(Inter-Integrated Communication) 통신 기반으로 PWM모터 제어 관련 요소 설계
  1) 구역 내 자유 이동 및 장애물 회피 구현 완료
  2) 유도선을 따라 순회 주행 구현 완료 (2종의 경로 구현)

2. 개발 목표 및 개발 결과

○ 개발 목표

- Cortex M3를 기반으로, 외부 전원 및 장치들을 제어하여, 장애물 회피 및 경로 Tracing이 되는 자율 주행 car 제작
- 외부장치 항목 – 총 7종





- 2가지 mode 지원하도록 설계
  1) Boundary 內 장애물 회피(도봇 청소기, etc)
  2) 경로 유도선을 따라 이동 및 장애물 감지 시 정지 후 알람 발생 및 대기(AGV – Automated Guided Vehicle)

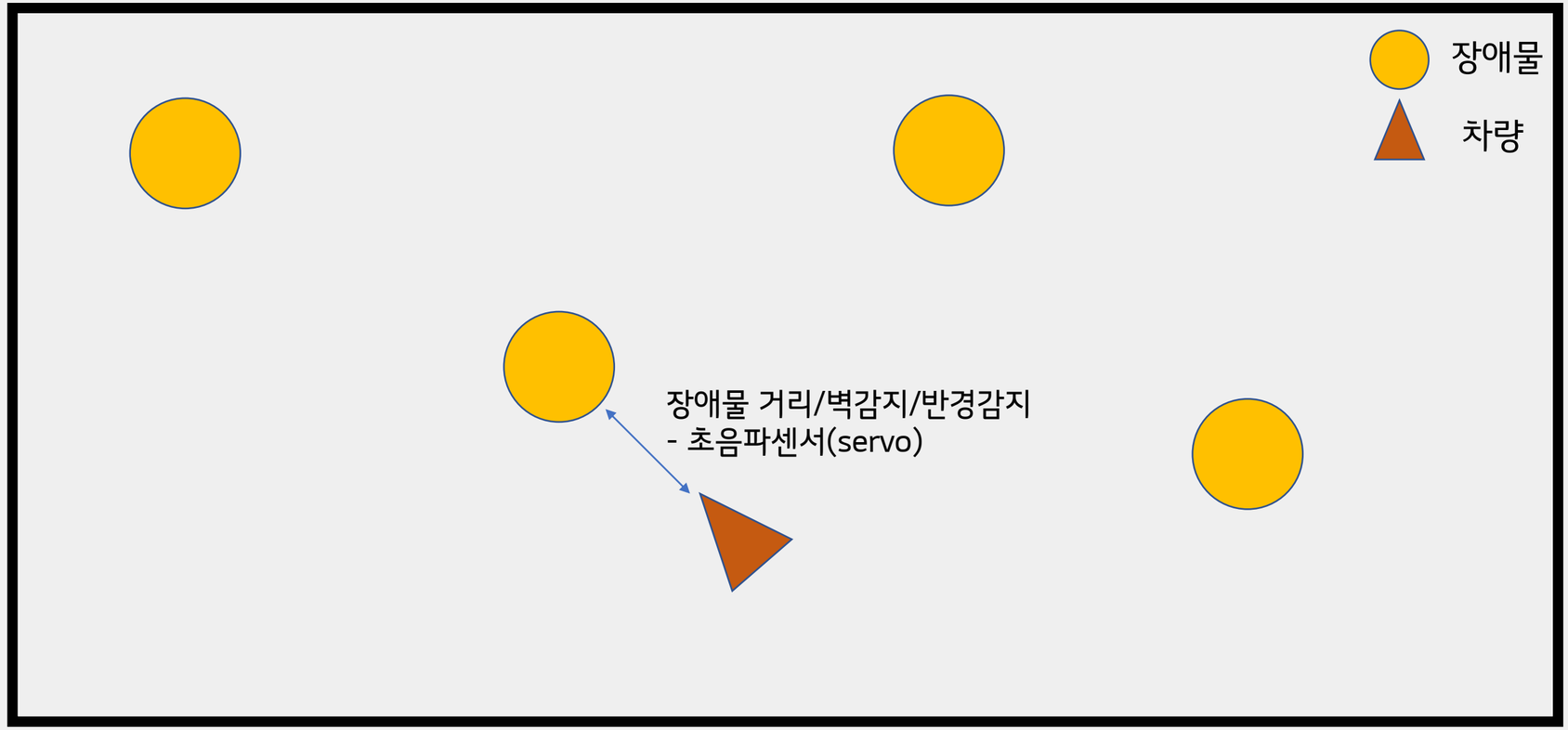
2. 개발 목표 및 개발 결과_로봇청소기 Mode

○기본 동작 목표

- 차체 전방 하부의 IR sensor 를 통해 구역 안/밖을 구분하여 동작 영역 제한
- 차체 전면부의 초음파센서를 통하여 전면부 장애물 감지 후 우회 동작
- 우회경로 미존재 시, 정지하여 RGB색상 및 Buzzer음 발생하여 동작 정지 알림

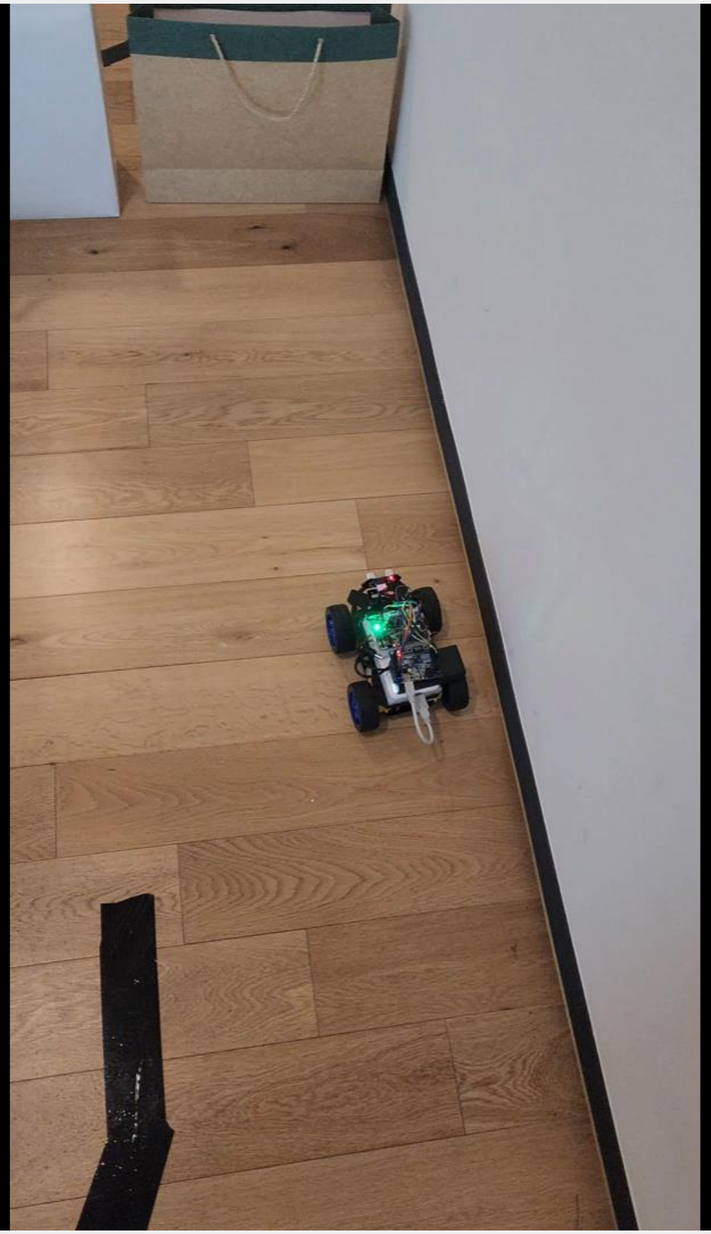
※ 동작 예시

*외곽 기준 선(or 벽) – IR Sensor 판단



2. 개발 목표 및 개발 결과_로봇청소기 Mode

MODE1



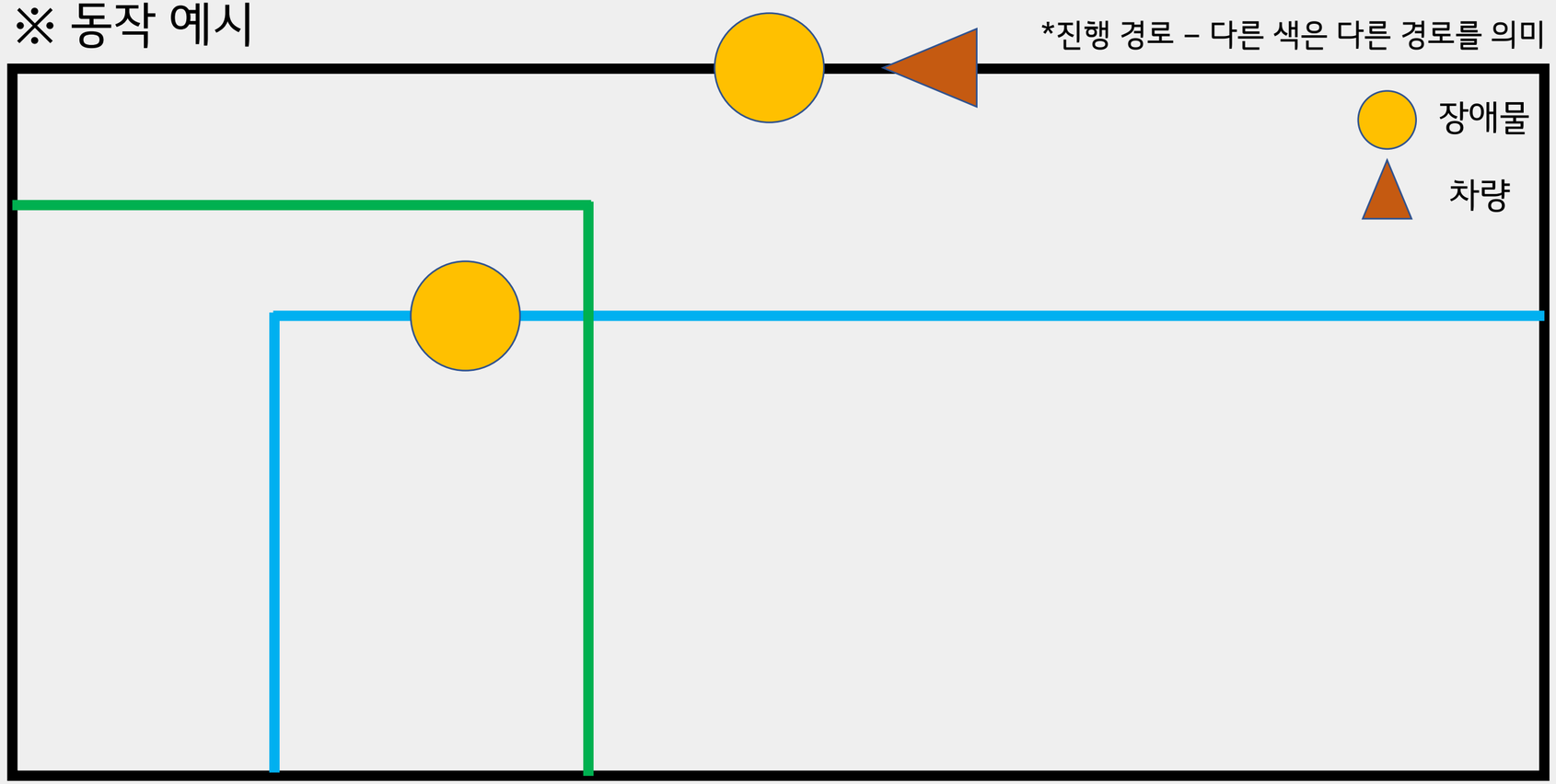
정상주행

2. 개발 목표 및 개발 결과_AGV Mode

○기본 동작 목표

- 생산 현장에서 필수적으로 사용되는 AGV 동작 logic 구현
- 주어진 유도선을 따라 주행 및, 장애물 감지 시 정지 및 buzzer 알람음 발생
- 필요한 경우, 다른 종류의 유도선 배치하여 경로 노선 추가

※ 동작 예시



1) 장애물 감지 시 정지 및 Beep음 발생
2) 여러 경로 존재 시, 유도선 type에 따라 해당경로 진행
   - 일부 노선은 동일 경로 공유

2. 개발 목표 및 개발 결과_AGV Mode

MODE2 주 경로 탐색
(흑색 유도선)

2. 개발 목표 및 개발 결과

○ 개발 결과

- 제어 요구 항목 6종에 대한 통신 및 구동 설계
- Sensor 류 Data 획득을 위한 호출 함수 및 header 설계

*adc.c – 하부 적외선 센서

```
void ADC_Init(int channel)
{
    Macro_Set_Bit(RCC->APB2ENR, 2);                 // PA POWER ON
    Macro_Write_Block(GPIOA->CRL, 0xf, 0x0, 4);     // PA1(ADC-IN1) = Analog Input
    Macro_Write_Block(GPIOA->CRL, 0xf, 0x0, 12);    // PA3(ADC-IN3) = Analog Input
    Macro_Write_Block(GPIOA->CRL, 0xf, 0x0, 20);    // PA5(ADC-IN5) = Analog Input

    Macro_Set_Bit(RCC->APB2ENR, 9);                 // ADC1 POWER ON (1/3/5 are all in ADC1)
    Macro_Write_Block(RCC->CFGR, 0x3, 0x2, 14);     // ADC1 CLOCK = 12MHz(PCLK2/6)

    /*
    Macro_Write_Block(ADC1->SMPR2, 0x7, 0x7, 5); // Clock Configuration of CH1 = 239.5 Cycles
    Macro_Write_Block(ADC1->SMPR2, 0x7, 0x7, 9); // Clock Configuration of CH3 = 239.5 Cycles
    Macro_Write_Block(ADC1->SMPR2, 0x7, 0x7, 15); // Clock Configuration of CH5 = 239.5 Cycles
    */

    Macro_Write_Block(ADC1->SQR1, 0xF, 0x0, 20); // Conversion Sequence No = 1

    if (channel==1) {
        Macro_Write_Block(ADC1->SMPR2, 0x7, 0x7, 3);
        Macro_Write_Block(ADC1->SQR3, 0x1F, 1, 0); // Sequence Channel of No 1 = CH1
    }
    else if (channel==3){
        Macro_Write_Block(ADC1->SMPR2, 0x7, 0x7, 9); // Clock Configuration of CH3 = 239.5 Cycles
        Macro_Write_Block(ADC1->SQR3, 0x1F, 3, 0); // Sequence Channel of No 2 = CH3
    }
    else {
        Macro_Write_Block(ADC1->SMPR2, 0x7, 0x7, 15); // Clock Configuration of CH5 = 239.5 Cycles
        Macro_Write_Block(ADC1->SQR3, 0x1F, 5, 0); // Sequence Channel of No 3 = CH5
    }

    Macro_Write_Block(ADC1->CR2, 0x7, 0x7, 17);     // EXT Trigger = SW Trigger
    Macro_Set_Bit(ADC1->CR2, 0);                    // ADC ON
}
```

*센서 데이터 기반 경계 판단 함수

```
int boundary(void){
    int i=0;
    int cnt_boundary=0;
    int channels[3]={5, 3 ,1};
    char* position[5]={"left", "center", "right"};
    for(i=0; i<3; i++){
        ADC_Init(channels[i]);
        ADC_Start();
        while(!Adc_Get_Status());
        unsigned int val=Adc_Get_Data();
        if(val>1500) cnt_boundary++;
        Uart1_Printf("%-8s %4d   ",position[i], val);
    }
    Uart_Printf("%d \n", cnt_boundary);
    if(cnt_boundary>2) return 1;
    else return 0;
}
```

*복수의 경로 판단 시에는 해당 threshold값이 추가로 분기

2. 개발 목표 및 개발 결과

○ 개발 결과

- I2C 통신 프로토콜 구축

*i2c.c - I²C 동신 기본 setup 및 data전달 함수 구현

```
#define PCA9685_I2CADDR                                 0x40
#define PCA9685_I2CADDR_WR                              (PCA9685_I2CADDR|0x0)
#define PCA9685_I2CADDR_RD                              (PCA9685_I2CADDR|0x1)

void I2C_PCA9685_Init(unsigned int freq)
{
    unsigned int r;
    volatile int i;

    Macro_Set_Bit(RCC->APB1ENR, 21);

    Macro_Clear_Bit(RCC->APB1RSTR, 21);
    for(i=0; i<1000; i++);
    Macro_Set_Bit(RCC->APB1RSTR, 21);
    for(i=0; i<1000; i++);
    Macro_Clear_Bit(RCC->APB1RSTR, 21);

    Macro_Set_Bit(RCC->APB2ENR, 3);
    Macro_Clear_Bit(AFIO->MAPR, 1);
    Macro_Write_Block(GPIOB->CRL, 0xff, 0xFF, 24);   //PB6&7as output, alternative <-SDA/SCL

    Macro_Write_Block(I2C1->CR2, 0x3f, PCLK1/1000000, 0);  //Gives Frequency
    Macro_Clear_Bit(I2C1->CR1, 0);  //Peripheral disable
    I2C1->TRISE = PCLK1/1000000+1;
    r = PCLK1/(freq*2);
    I2C1->CCR = ((r<4)?4:r);

    Macro_Clear_Bit(I2C1->CR1, 1);
    Macro_Set_Bit(I2C1->CR1, 0);
    Macro_Set_Bit(I2C1->CR1, 10);
    Uart_Printf("I2C Init finished\n");
}
```

*Motor Driver에 data 전송하는 주 함수

```
void I2C_PCA9685_Write_Reg(unsigned int addr, unsigned int data)
{
    //volatile int i=0;
    while(Macro_Check_Bit_Set(I2C1->SR2, 1));                    // Idle OK
#if debug
    Uart_Printf("here 1\n");
#endif
    Macro_Set_Bit(I2C1->CR1, 8);                                 // Start
    while(Macro_Check_Bit_Clear(I2C1->SR2, 0));                  // Check Start
    //SR2[0]- 1:Master Mode, 0:Slave Mode
#if debug
    Uart_Printf("here 2\n");
#endif
    I2C1->DR = (PCA9685_I2CADDR_WR<<1)&0xFE;                 // Send WR Address
    while(Macro_Check_Bit_Clear(I2C1->SR1, 1));
    //SR1[1], Master - 1: End of address transmission, 0: No end of address transmission
    //SR1[1], Slave - 1: Received address matched, 0:Address mismatched or not received
#if debug
    Uart_Printf("here 3\n");
#endif
    (void)I2C1->SR2;                                             // Clear ADDR flag by reading SR2
    I2C1->DR = addr;                        // Send Register Address
    while(Macro_Check_Bit_Clear(I2C1->SR1, 2));                  // Check Byte Transfer Finished
    //SR1[2]- 1: Data byte transfer succeeded, 0: Data byte transfer not done
#if debug
    Uart_Printf("here 4\n");
#endif
    I2C1->DR = data;
    //Uart_Printf("Data Written %d\n", data);                                  // Send Data
    while(Macro_Check_Bit_Clear(I2C1->SR1, 2));                  // Check Byte Transfer Finished
#if debug
    Uart_Printf("here 5\n");
#endif
    Macro_Set_Bit(I2C1->CR1, 9);                                 // Stop
    while(Macro_Check_Bit_Set(I2C1->CR1, 9));                    // Check Stop(Auto Cleared)
```

2. 개발 목표 및 개발 결과

○ 개발 결과

- Motor류 구동을 위한 함수 구현
- 각 동작 logic별 동작 pin 배열화하여, 추가 동작 구현 시 용이하도록 설계

*motor.c – 바퀴 진행 방향 별 배열 지정 및 duty 공급함수 제작

```c
#include "device_driver.h"
/* LED 8-15
 * LEFT Front Forward : 13 LEFT Front Reverse : 12
 * LEFT Rear Forward : 15 LEFT Rear Reverse : 14
 * Right Front Forward : 10 Right Front Reverse : 11
 * Right Rear Forward : 8 Right Rear Reverse : 9
 */
//speed is in terms of duty rate (0-100%)
int pinout=0;
int i;
int delay_opt=100;
int delay_count;
int forwards[4]={8,15,13,10};
int backwards[4]={9,14,12,11};
int stops[8]={8,15,13,10,9,14,12,11};
int flefts[2]={15,13};
int blefts[2]={14,12};
int frights[2]={8,10};
int brights[2]={9,11};



void set_pwm_duty(unsigned int pinout, unsigned int duty){
    unsigned int pwm_val = (unsigned int)(duty * 40.95);  // 40.95 = 4096 / 100
    //I2C_PCA9685_Write_Reg(0x6 + pinout * 4, 0);
    //I2C_PCA9685_Write_Reg(0x7 + pinout * 4, 0);
    I2C_PCA9685_Write_Reg(0x8 + pinout * 4, pwm_val & 0xFF);
    I2C_PCA9685_Write_Reg(0x9 + pinout * 4, pwm_val >> 8);

}
```

동작 logic별 함수 구현

```c
void forward(int speed){
    for(i=0; i<4; i++){
        set_pwm_duty(forwards[i], speed);
    }
}

void backward(int speed){
    for(i=0; i<4; i++){
        set_pwm_duty(backwards[i], speed);
    }
}

void spin_left(int speed){
    for(i=0; i<2; i++){
        set_pwm_duty(blefts[i], speed);
        set_pwm_duty(frights[i], speed);
    }
}

void spin_right(int speed){
    for(i=0; i<2; i++){
        set_pwm_duty(flefts[i], speed);
        set_pwm_duty(brights[i], speed);
    }
}


void turn_car(int speed1, int speed2){
    for(i=0; i<2; i++){
        set_pwm_duty(flefts[i], speed1);
        set_pwm_duty(frights[i], speed2);
    }
}
```

*forward: 직진 주행

*backward: 후진

*spin_left: 제자리 좌회전

*spin_right: 제자리 우회전

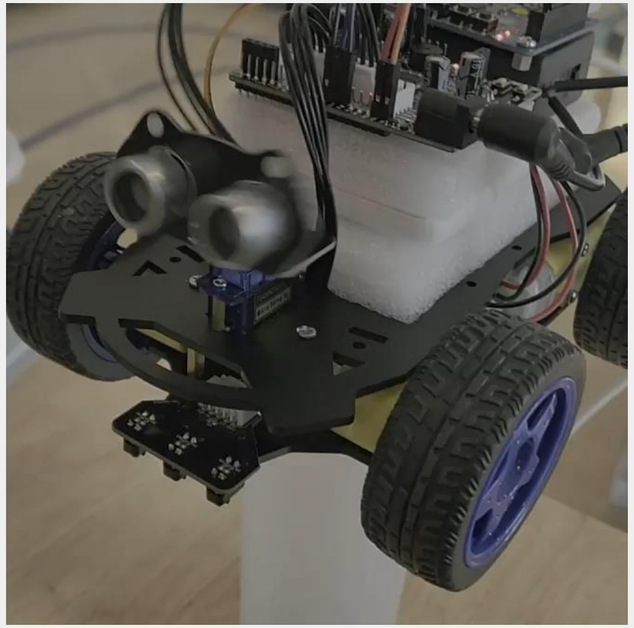
*turn_car: 좌/우 모터 duty 지정하여 원하는 방향으로의 회전 기동

3. 핵심 기술

○ 거리 감지 센서에 서보 탑재를 통한 최적 경로 탐색

- 좌/우 각 10회씩 거리 감지 후, 여유거리가 더 큰 곳으로 이동
- 좌/우 모두 15cm 이하일 시, 180도 선회

<거리 감지 방향 제어>



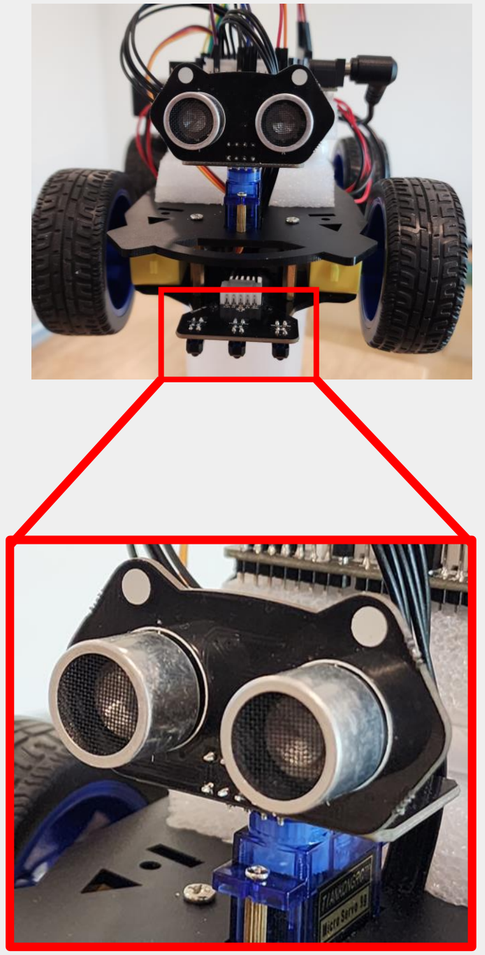
<동작 함수>

```
int avoid_obstacle(void){
    unsigned int dist_left=0, dist_right=0, dist_center=0;
    int tmp=0, cnt=0, dist_sum=0;

    Servo_180(0, 60);
    TIM2_Delay(750);
    for (;;){
        tmp=Distance();
            dist_left+=tmp;
            cnt++;
            if(cnt>=10) break;
    }

    cnt=0;
    Servo_180(0, 160);
    TIM2_Delay(750);
    for (;;){
        tmp=Distance();
            dist_right+=tmp;
            cnt++;
            if(cnt>=10) break;

    }
    Servo_180(0, 120);
    if(dist_left<=150 && dist_right<=150) return 0;

    if(dist_left>dist_right) return 1;
    else return 2;
}
```

<Data 출력>

```
LEFT
time:  640 distance:  11
time:  640 distance:  11
time:  640 distance:  11
time:  640 distance:  11
time:  640 distance:  11
time:  640 distance:  11
time:  640 distance:  11
time:  640 distance:  11
time:  640 distance:  11
time:  640 distance:  11
Right
time:  2520 distance:  43
time:  2520 distance:  43
time:  2520 distance:  43
time:  2520 distance:  43
time:  2520 distance:  43
time:  2520 distance:  43
time:  2520 distance:  43
time:  2520 distance:  43
time:  2520 distance:  43
time:  2520 distance:  43
```

※거리 감지 sensor 위치



3. 핵심 기술

○ 3개 sensor 감지값 기반하여, 감지 case별 동작 세분화

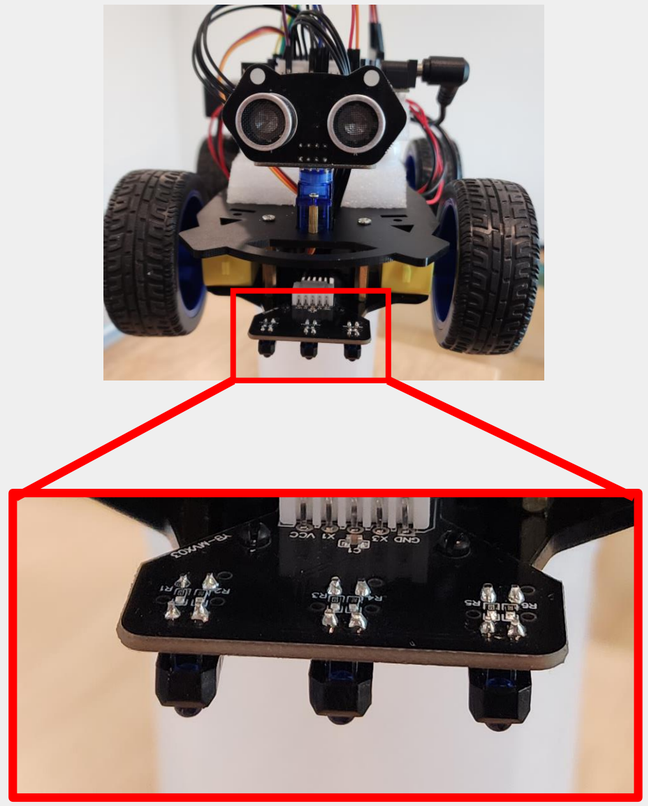
- 총 8가지 case 판단 가능
- if분기에서 속도 최적화를 위해 bit연산사용

	좌측	중앙	우측
유도선 이탈 시:	0	0	0
유도선 감지 시:	1	1	1

```
for(i=0; i<3; i++){
    ADC_Init(channels[i]);
    ADC_Start();
    while(!Adc_Get_Status());
    tmp=Adc_Get_Data();
    if(tmp>threshold_1) val+=(1<<i);
    Uart1_Printf("%-8s %4d    ",position[i], tmp);
}
```

0	**0**	**0**	후진 하여 경도 탐색(3회 반복 후 경도 미 확인 시 종료)
0	0	1	중앙부 이탈 → 좌회전(속도 **강**)
0	1	0	좌우 대칭 및 중앙부 경도 상에 있으므도 그대도 진행
0	1	1	좌측만 이탈 → 우회전(속도 약)
1	0	0	중앙부 이탈 → 우회전(속도 **강**)
1	0	1	좌우 대칭으도 중앙부만 이탈하였으므도 그대도 진행
1	1	0	우측만 이탈 → 좌회전(속도 약)
1	1	1	정상감지 중이므도 그대도 진행

※경도 추적 sensor 위치



4. 결과 분석 및 기대 효과

○ 목표 대비 결과 성능

개발 목표	모터 기동	장애물 회피	유도선 추종	유도선 경로 이원화
달성 항목	- Motor Driver를 통한 개별 motor제어 가능	- 충돌 이전 정지가능 - 정지 후 좌우 거리 확인 및 방향 제어/이동 가능	- 주어진 경로에 따라 추종 가능	- Mode변경에 따라 부 경로 감지후 주행 가능
기대 효과	- 원하는 logic을 자유롭게 반영하여 차체 동작 가능 (속력, 방향 등)	- 전방의 장애물 감지 및 회피를 통해 원하는 process추가 가능 ex) 정지 후 경로 변경, mode변경 등	- 자재 공급 시 필요한 동선을 제작하여 원하는 곳으로 이동 및 수송 가능	- 필요한 경로를 이원화하여, A는 target A'에, B는 target B'에 전달 가능
미완성 항목	- 저속(Low Duty)에서 주행 중 멈춤 발생	- 좌우 감지 후 회전 시, 부정확한 회전각도	- 경로이탈 확인 시, 차체 회전각 과다 발생	- 유도선 반사율이 다양하지 못해 추가 경로 설정 어려움
개선책	- Motor 등급 변경 - 위치감지 sensor등으로, low duty에서 멈출 시, high duty 공급	- 가속도 센서 등을 통해, 원하는 각도로 제어 - 지속적으로 회전하면서 거리측정하여 순간순간 최적경로 탐색	- 경로추적 sensor 측정 주기 단축 필요	- 색상 및 반사율이 다양한 소재 및 경로 탐지 logic 개선 필요

5. 향후 연구 과제

○ 추후 개선점

1) 정밀한 회전 각도 및 자세 제어
   → 가속도 센서를 추가하여, 보다 정밀한 자세 제어 및 위치 data 수집
   ex) ADXL375 6축 Gyro Sensor

*ADXL375 sensor

2) 경로 추적시 과다 이탈 방지
   → 현재 센서 모듈의 위치가 고정이므로, 추가로 2개소를 배치하여,
   보다 세분화된 속도 제어(1개 추가시 5ms 추가)
   → 1번에서 수집한 위치 data를 기반으로 동작 경로를 학습하여,
   보다 원활한 경로 추적 및 monitoring

*현재 경로 추적 센서 위치(고정)

3) 센서류 1회 동작 시 delay 과다 (현 수준: 190ms)
   → Interrupt방식의 제어 방식 도입 필요
   (센서류 동작을 위한 init호출시 120ms 정도의 시간 소요됨)

*매 loop 동작 시 동작 제어에 190ms 정도 소요

```
tracing mode: 2
time: 2520 distance: 43
left     4095   center   4095   right    4095
time: 190.14
time: 2520 distance: 43
left     4095   center   4095   right    4095
time: 190.10
```

6. 프로젝트 수행 후기

○ 난관들과 Troubleshooting, 느낀점

1) H/W Wiring의 중요성

- 잘 되던 동작이 다음날 오작동 -> 출력 pin들이 금속에 닿아서 short
- 외부장치가 1개 늘 때마다 기하급수적으로 늘어나고 쉽게 빠지는 jumper 선들...
- 외부장치마다 서로 다른 통신법/동작원리/조작법

→ Embedded는 H/W도 공부를 해야한다

2) I2C 통신 살리기

- Motor Driver의 I2C 통신규약을 Manual 어디에서도 도저히 찾을 수 없었음
- H/W 주소와 전송 data bit연산 수식을 하나하나 수정해가면서,
  5~60여번 시도 끝에 간신히 연결

→ 안 될 거 같아도 개발자를 갈아 넣으면 되기는 한다.

3) 초음파 센서

- Arduino에서 가장 많이 써봤던 모듈이라 당연히 가장 쉬울 것으로 착각
- Micro-Processor의 동작 방식에 대해서 가장 많은 detail을 이해했어야 하는 모듈이었음
- 첫 거리측정 값 얻기 까지 만 1주일을 갈아 넣음

→ **간단한 동작이라고 해도 구현은 절대 쉽지 않다.**

*지난 4주 요약