Aurora Design Lab

하이브리드 모터제어 컨트롤러

개발 보고서

V2.0

AuroraDesignLab., Co., Ltd.

전화 번호 0505-552-0005 서울시 금천구 두산로 70 B 동 803 호

팩스 02-866-7506

목차

내용

- 2021.05.10 초안 작성
- 2021.05.18 두 번째 수정안
- 2021.06.18 두 번째 수정안

개요

특징

스텝 및 서보 모터와 LED 조명 제어를 동시에 지원하는 하이브리드 컨트롤러이며, 하이파워 고출력 스텝모터 드라이버 3 개를 내장하고 있으며 2 개의 PWM 기반 5V 서보모터 제어와 리미트 스위치 4 개를 이용해 다양한 모터제어 기능을 제공하며 2 개 채널의 SK6812RGBW LED strip 을 제어합니다.

기능

- USB-B 시리얼 제어포트 지원
- 스텝모터 3port 와 서보 모터 2 포트 제어 기능 지원
- 리미트 스위치 4 개 지원
- 2 개 채널 SK6812RGBW 신호 출력 기능

주의 사항

이 제품은 하이파워 스텝모터 드라이버를 사용하고 있으며 모터 제어 시 탈조 및 과열에 의한 파손이 될 수 있으니 주의 바랍니다.

상세 사항

구분	사양 명세	비고
MCU	• 32bit RISC processor • ATSAMD21G18 적용	
SPEC	• Pololu High-Power Stepper Motor Driver 36v4 채용 • 5V 서보 모터 제어 기능 • SK6812RGBW 제어 기능	* 스텝모터 사용 전류량에 따라 드라이버 보드에 직접 전원을 연결할 수 있음.
DC SPEC	• Input: Max DC 55V • Operation Temp: -20~70℃	
Others	• FT232RL USB 시리얼 드라이버 적용	

명령어 구조

기능

1. Step 모터 구동 방향
2. Step 모터 Step 제어 Mode 변경
3. Step 모터 Step 제어
4. Driver 셋팅 모드 확인/설정

명령어 형식 (구분자 #)

STX	CMD Field Length	CMD_Sets	Check Sum	ETX
String (STX)	INT (0~254)	Length_CMD_Sets*CMD	INT [%256]	String (ETX)

예시: LED 1 번을 RGBW 200 120 100 20 색상으로 제어할 때

[STX#6#LED#1#200#120#100#20#185#ETX]

(1) STX (Start Code)

프로토콜 전송 시작 코드로 '**STX**' 로 시작한다. (String)

(2) CMD Field Length (필드 개수)

STX, Length, CheckSum, ETX 를 제외한 명령어 필드의 길이를 의미한다.

예시)

- STM 의 경우 [#STM, #모터 ID, #방향, #회전각도 단위, #모터 스텝 수, #스텝 당 딜레이]로 총 6 의 CMD Field Length 를 갖는다.
- SVM 의 경우 [#SVM, #모터 ID, #절대위치(각도)]로 총 3 의 CMD Field Length 를 갖는다.
- LED 의 경우 [#LED, #조명 ID, #R, #G, #B, #W]로 총 6 의 CMD Field Length 를 갖는다.

(3) CMD_Sets(명령어 구조)

이전 Length 의 길이만큼의 명령어를 가진다. 명령어 구조는 제어하고자 하는 대상의 종류에 따라 아래에 설명을 따른다.

(4) Check Sum

CMD_Sets 의 값을 모두 더하고 256 으로 나눈 나머지 값이다.

예시)

- STX 를 포함하여 [STX#6#LED#1#200#120#100#20#(checksum)#ETX]의 경우

- STX, #LED, #ETX 를 제외하고 #안의 숫자를 모두

  더한 **441(1+200+120+100+20)**를 %256 한 185 값에 #을 더해 아래와 같은 메시지를 전송한다.

- **STX**#6#LED#1#200#120#100#20**#185#ETX**

(5) ETX (End Code)

- 프로토콜 전송 종료 코드로 '**ETX**' 로 끝난다. (String)

Step Motor Control

기능

1. Step 모터 ID 선택
2. Step 모터 구동방향
3. Step 모터 Step 제어
4. 회전 속도

명령어 형식

제어종류	모터 ID	방향	회전각도 단위 (정밀도)	모터 스텝 수	스텝 당 딜레이 (속도제어)
STM [String]	1~3 [int]	0~3 [int]	1,2,4,8...256 (정지 시 0) [int]	0~INT_MAX (정지 시 0) [int]	100~3000 (정지 시 0) [int]

A. 제어종류(구분용) : STM

B. 모터 ID : #ID → 한 자리 숫자 : 1, 2, 3

C. 방향(명령) : #0 == 정지, #1 == 정방향, #2 == 역방향, #3 == 위치초기화

#위치초기화는 리미트 스위치가 가동될 때까지 이동함.

D. 회전각도단위(정밀도) : #0 == 정지, 최소구동각도인 1.8 도를 나누어 정밀하게 제어시 활용, 2 의 배수로 입력 가능

예)#128 : 모터에서 제공하는 (기준 step 이 1.8 일 경우) / 128 만큼 0.014 도로 스텝 제어함. 값이 커지면 미세하게 조절하고, 1 이면 기본 회전각도인 1.8 도로 제어함.

E. 모터 스텝수 : #0 == 무한, #1 == 1 스텝, #2 == 2 스텝, ...

- 무한의 경우 해당 동작 와중엔 정지와 모터 제어를 제외한 명령어 입력은 모두 무시된다.

- 동작 중에 입력한 명령어 중 모터 속도는 이미 동작하고 있는 모터의 모터 속도를 따른다.

F. 스텝 당 딜레이 (속도제어): 모터 스텝 이동 후 설정 값만큼 us 지연함.

- #100 == 한 스텝 이동 후 100us delay

예시)

스텝모터 1 번을 정방향으로 (기준 step)/128 으로 1,000 번 회전하며 스텝당 500us 만큼 delay 하는 방법

=> [#STM#1#1#128#1000#500]

스텝모터 1 번을 정지하는 방법

=> [#STM#1#0#0#0#0]

Servo Motor Control

기능

1. 모터 구동 각도 제어

명령어 형식

제어종류	모터 ID	절대위치 (각도)
SVM [String]	1~2 [int]	10~170 [int]

A. 제어종류(구분용) : SVM

B. 모터 ID : #ID → 한 자리 숫자 : 1(Pan), 2(Tilt)

C. 절대위치(각도) : #10 ~ #170 (#90 은 가운데 위치)

예시)

1 번 서보 모터를 135 도의 위치로 이동시킬 경우

=> [#SVM#1#135]

LED Bar Control

기능

1. LED 바 색상 변경

명령어 형식

제어종류	조명 ID	R	G	B	W
LED [String]	1~2 [int]	0~255 [int]	0~255 [int]	0~255 [int]	0~255 [int]

A. 제어종류(구분용) : LED

B. 조명 ID : #ID → 한 자리 숫자 : 1, 2

C. RGBW 색상조절 : #0~255, #0~255, #0~255, #0~255

예시)

1 번 LED 조명의 색상을 R200, G120, B100, W20 으로 제어할 경우

=> [#LED#1#200#120#100#20]

명령어 예시

1. 세 가지 스텝 모터를 동시에 돌리는 코드

=> STX#6#STM#1#2#8#0#100#111#ETX

STX#6#STM#2#2#16#0#100#120#ETX

STX#6#STM#3#1#256#0#100#104#ETX

2. 모터 별 가장 빠른 동작

=> 모터 1: STX#6#STM#1#1#4#0#100#106#ETX

STX#6#STM#1#2#4#0#100#107#ETX

=> 모터 2: STX#6#STM#2#1#16#0#100#119#ETX

STX#6#STM#2#2#16#0#100#120#ETX

=> 모터 3: STX#6#STM#3#1#8#0#20#32#ETX

3. 모터 별 정지

=> 모터 1: STX#6#STM#1#0#0#0#0#1#ETX

=> 모터 2: STX#6#STM#2#0#0#0#0#2#ETX

=> 모터 3: STX#6#STM#3#0#0#0#0#3#ETX

명령어 예시

4. 초기 장소로 이동

=> STX#6#STM#1#3#0#0#0#4#ETX

5. 한 스텝(1.8) 한번 이동(모터 3)

=> STX#6#STM#3#1#1#1#1000#238#ETX(overshoot, undershoot 발생)

=> STX#6#STM#3#1#256#256#1000#236#ETX

6. 한 바퀴 이동(모터 3)

=> STX#6#STM#1#1#1#200#1300#223#ETX

=> STX#6#STM#2#1#1#200#1300#224#ETX

7. 서보 제어

=> STX#3#SVM#1#10#11#ETX

=> STX#3#SVM#1#110#111#ETX

=> STX#3#SVM#2#20#22#ETX

=> STX#3#SVM#2#90#92#ETX

=> STX#3#SVM#2#30#32#ETX

8. LED 동작

=> STX#6#LED#1#200#120#100#20#185#ETX

=> STX#6#LED#1#255#0#0#0#0#ETX

=> STX#6#LED#1#0#255#0#0#0#ETX

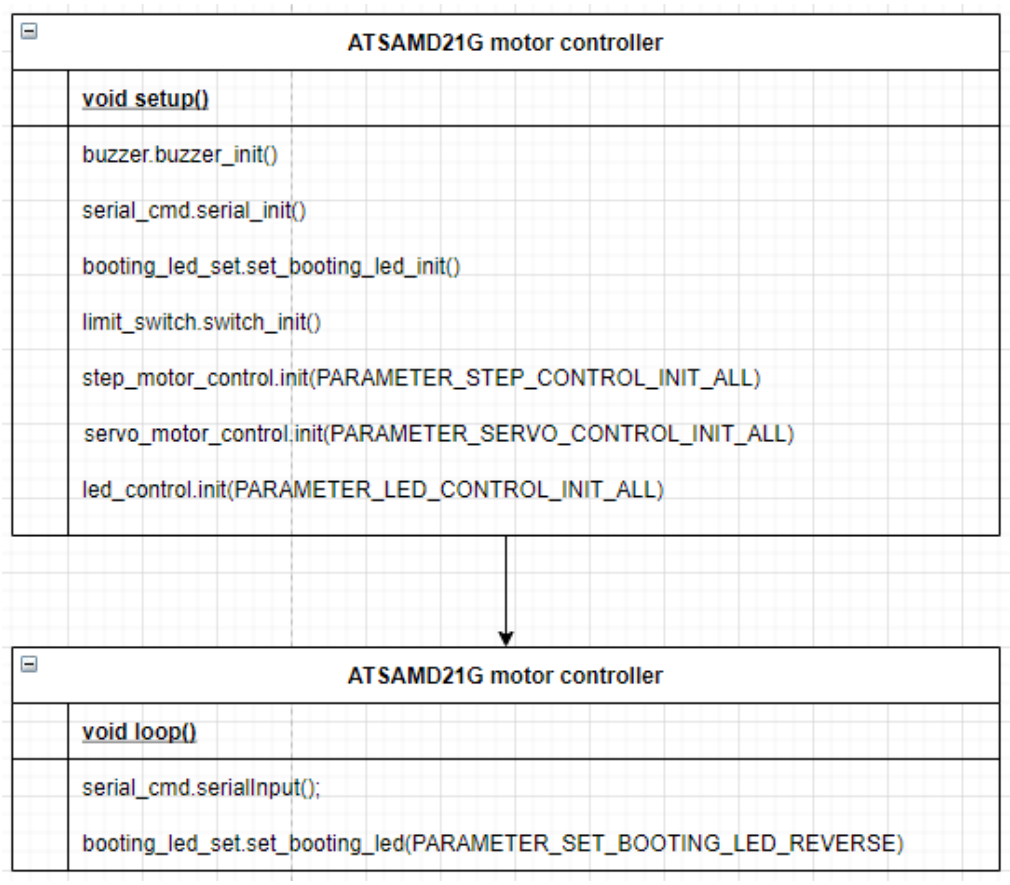
=> STX#6#LED#1#0#0#255#0#0#ETX

=> STX#6#LED#1#0#0#0#255#0#ETX

9. LED 동작 끄기

=> STX#6#LED#1#0#0#0#0#1#ETX

프로그램 구조



- void setup()

1. buzzer.buzzer_init()

 - 모터 동작 중 Error 발생 및 위급상황 시 울리는 부저를 위한 초기 설정이다.

2. serial_cmd.serial_init()

 - serial 명령을 받기 위해 필요한 통신 포트를 초기화하는 코드이다.

3. booting_led_set.set_booting_led_init()

- 보드의 정상 동작을 테스트 하기 위한 LED blink 초기 설정이다.

- 1 초마다 좌우 led 가 번갈아서 on-off 된다.

4. limit_switch.switch_init()

- 상하좌우, 네 개의 리미트 스위치, 보드 상의 위급 상황 스위치 관련 인터럽트 초기 설정이다.

5. step_motor_control.init(PARAMETER_STEP_CONTROL_INIT_ALL)

- 스텝모터 제어를 위한 초기 설정이다.

- 모터 드라이버에 존재하는 chipselectpin, decaymode, currentmilliamps36v4, step mode, direction 등이 존재하며 세 스텝모터를 정방향, 역방향을 반 바퀴씩 회전하며 동작을 점검한다.

6. servo_motor_control.init(PARAMETER_SERVO_CONTROL_INIT_ALL)

- 서보 모터 제어를 위한 초기 설정이다.

- 서보 모터의 위치를 초기 위치로 이동시킨다.

7. led_control.init(PARAMETER_LED_CONTROL_INIT_ALL)

- LED 초기 설정을 위한 내용이다.

- void loop()

1. serial_cmd.serialInput()

 - UART 시리얼 통신을 통해 전송받은 명령어를 파싱해 control 하는 매소드이다.

 - 오류 처리 내역으로 STX, length, checksum, ETX 를 확인하고 부저음을 출력한다.

 - 명령어 프로토콜 규칙이 올바르다면 각각의 control 명령어를 cmd 에 따라 제어한다.

2. booting_led_set.set_booting_led(PARAMETER_SET_BOOTING_LED_REVERSE)

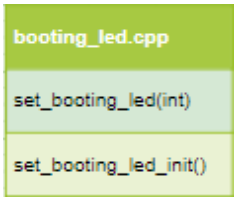
 - 보드 동작 여부를 확인하기 위해 led 부팅 상의 blink 예제이다.

프로젝트 파일 구조

booting_led.cpp	buzzer.cpp	LED.cpp	Limit_switch.cpp	Serial_port.cpp	step_motor.cpp	servo_motor.cpp	timer_handler.cpp	control.cpp
set_booting_led(int)	buzzer_init()	set_init(int, void*)	switch_init()	read_cmd()	set_init(int, void*)	set_init(int, void*)	startTimer(int)	control(int, void*)
set_booting_led_init()	warning()	get_init(int, void*)	swInterrupt()	attach_cmd(int, std::vector<String>)	get_init(int, void*)	get_init(int, void*)	setTimerFrequency(int)	init(int)
		set(int, void*)	LIMIT_MOTOR_ONE_SW_ONE_INTERRUPT()	serial_init()	set(int, void*)	set(int, void*)	TC3_Handler()	controlling(int, void*)
		get(int, void*)	LIMIT_MOTOR_ONE_SW_TWO_INTERRUPT()	serialInput()	get(int, void*)	get(int, void*)	serial_interrupt()	control_init(int, void*, int)
		set_LED_id(int*)	LIMIT_MOTOR_TWO_SW_ONE_INTERRUPT()	return_check_sum(std::vector<String>)	set_motor_id(int*)	set_motor_id(int*)		control_imergency_init()
		set_LED_R(int*)	LIMIT_MOTOR_TWO_SW_TWO_INTERRUPT()	read_motor_stepping()	set_dir(int*)	set_angle(int*)		control_stepping_init()
		set_LED_G(int*)		serial_interrupt_Input()	set_angle(int*)	get_motor_id(int*)		control_step_motor_data_init()
		set_LED_B(int*)			set_rotate(int*)	get_angle(int*)		
		set_LED_W(int*)			set_rpm(int*)			
		get_LED_id(int*)			get_motor_id(int*)			
		get_LED_R(int*)			get_dir(int*)			
		get_LED_G(int*)			get_angle(int*)			
		get_LED_B(int*)			get_rotate(int*)			
		get_LED_W(int*)			get_rpm(int*)			

- 그림. 주요 파일 별 함수 메소드 –

1. booting_led.cpp



- 그림. booting_led.cpp–

1. set_booting_led(int)

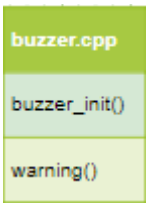
- 파라미터 값에 따라 LED 의 값을 1 초에 한번씩 현재 시간을 측정해 변경하는 코드.

- 설정된 핀번호에 해당하는 핀이 번갈아가며 좌우 on-off 를 반복한다.

2. set_booting_led_init()

- led 의 핀 번을 할당해주는 코드이다.

2. buzzer.cpp



- 그림. booting_led.cpp–

1. buzzer_init()

- 보드 초기 동작 시 울리는 부저음

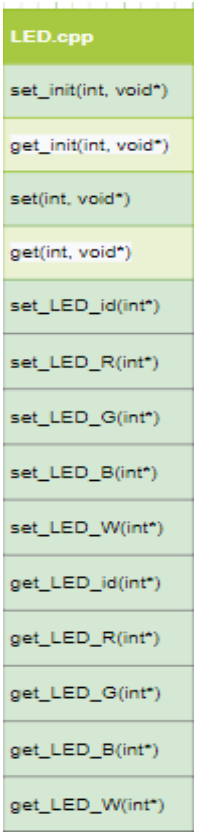
- tone 명령어를 통해 동작하며 두 개의 음이 연속해서 발생

2. warning()

- 에러상황, 또는 응급사항 발생 시 울리는 부저음

- tone 명령어를 통해 동작하며 한 음을 발생

3. LED



- 그림. LED.cpp –

1. set_init(int, void*);

- LED 객체의 led 중 led_id 와 R,G,B,W 를 void 포인터와 매칭하여 set 한다.

2. get_init(int, void*);

- LED 객체의 led 중 led_id 와 R,G,B,W 를 void 포인터와 매칭하여 get 한다.

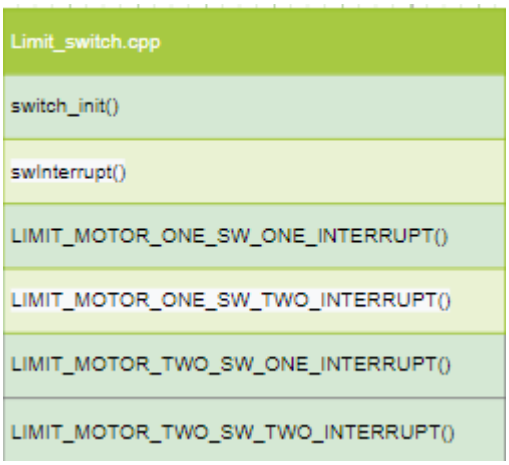
3. set(int, void*), get(int, void*)

- LED 객체의 led 중 파라미터에 따라 led_id 와 R,G,B,W 중 특정 값만 set, get 한다.

4. set_LED_[value](int, void*), get_LED_[value](int, void*)

- 파라미터가 아닌 방법으로 led_id 와 R,G,B,W 중 특정 값만 get, set 한다.

4. Limit_switch.cpp



- 그림. Limit_switch.cpp –

1. switch_init()

- 리미트 스위치 및 위험 비상 스위치 초기화 및 인터럽트 생성 코드

- input 핀 모드를 각각 비상 , 리미트 4 개 별로 선언한다.

- 인터럽트 생성 코드를 비상, 리미트 4 개 별로 분리해 선언한다.

2. swInterrupt()

- 위험 비상 스위치 발생시 작동하는 인터럽트 코드

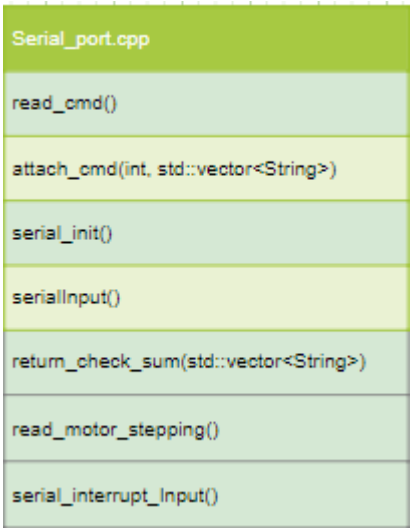
- 인터럽트 발생 시 기존의 정지 플래그를 모두 활성화 한다.

3. LIMIT_MOTOR_[NUM]_SW_[NUM]_INTERRUPT()

- 각각의 리미트 스위치가 발생했을 시 작동하는 인터럽트 코드

- 인터럽트 발생 시 해당 스위치에 해당하는 정지 플래그를 활성화한다.

5. Serial_port.cpp



- 그림. Serial_port.cpp –

1. read_cmd()

- serial_input 을 통해 얻은 데이터를 토큰 별로 분리해 파싱한 후 명령어를 attach_cmd 에 전달한다.

- STX, ETX, cmd length, checksum 을 확인한 후 에러처리 문을 거친다.

- 명령어 파싱이 올바르게 됐을 시, cmd 종류에 따라 attach_cmd 를 호출한다.

2. attach_cmd(int, std::vector<String>)

- 파라미터에 따라 step_motor, servo_motor, led 등의 객체에 파싱한 데이터를 넣은 후 control 제어 명령을 호출한다.

3. serial_init()

- 시리얼 통신을 위한 포트 추가 확장 및 초기화, 명령어 변수의 메모리 할당 등이 포함되어 있다.

- 보드의 통신 포트는 총 두 개가 있으며, 명령어 전송 포트는 serial5 를 사용했다.

4. serialInput()

- 보드의 시리얼 버퍼에 데이터가 수신되면, 모터의 동작 상태에 따라 명령어 셋을 전송해 제어한다.

- 시리얼 버퍼에 수신된 데이터를 개행문자가 발생하기 전까지 수신한 후 모터의 동작 상태에 따라 호출 메소드를 분류해 전송한다.

5. serial_interrupt_Input()

- 스텝 모터의 동작 와중에 시리얼 송신에 대한 명령어 처리를 담당하는 코드이다.

- serialInput 과 명령어 동작원리는 유사하다.

6. return_check_sum(std::vector< String > cmd_stack)

- cmd 값들의 checksum 값을 반환해 전송한다.

- cmd_stack 의 길이를 측정하여 256 으로 나눈 값을 반환하는 방식이다.

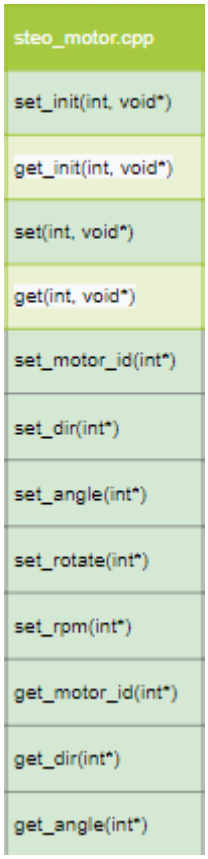
7. read_motor_stepping()

- 모터가 돌고 있는 와중에 일 때 serialInput 명령어에 의해 호출되는 메소드이다.

- 모터 동작 중에는 모터의 움직임, 모터 정지 명령어를 제외한 명령어는 모두 무시된다.

(초기 장소 이동도 마찬가지이다.)

6. step_motor.cpp



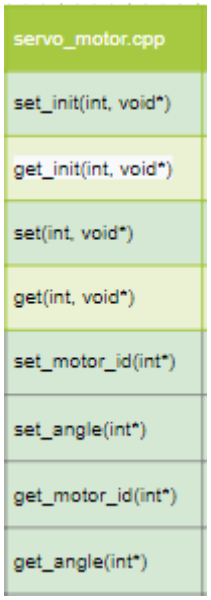
- 그림.step_motor.cpp –

1. set_init(int, void*), get_init(int, void*)

- motor_id, dir, step_angle, step_rotate_num, step_delay 등의 데이터를 set, get 한다.

2. 기타 set_[value](int*), get_[value](int*) 등은 LED 내용과 비슷하므로 생략한다.

7. servo_motor.cpp



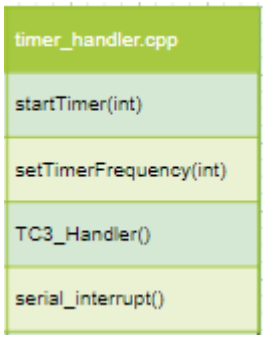
- 그림.servo_motor.cpp –

1. set_init(int, void*), get_init(int, void*)

- motor_id, angle 등의 데이터를 set, get 한다.

2. 기타 set_[value](int*), get_[value](int*) 등은 LED 내용과 비슷하므로 생략한다.

8. timer_handler.cpp(쓰이지 않음)



- 그림.servo_motor.cpp –

1. startTimer(int)

- 타이머 클락을 위한 레지스터 설정 코드이다.

- 타이머 인터럽트 발생을 위한 NVIC 설정을 담당한다.

2. setTimerFrequency(int)

- 클락 주파수를 선정하는 코드이다.

3. TC3_Handler()

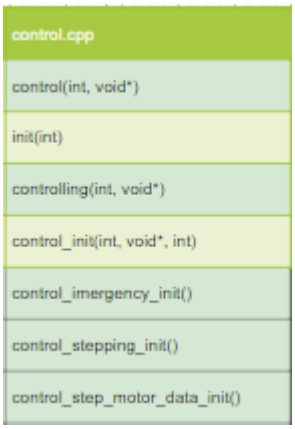
- TC3 에 대한 인터럽트 콜백함수를 호출하는 메소드이다.

4. serial_interrupt()

- 인터럽트 콜백 함수로 호출되는 함수이다.

- 시리얼 통신에 의해 송신된 명령어 string 을 똑같이 파싱한다.

9. control.cpp



- 그림.control.cpp –

모터 동작 시 유의사항
[1] step 모터 동작 이전에 리미트 스위치의 상태를 확인해 이미 리미트가 눌린 방향으로 진행하는 명령어는 무시한다. => 모터의 동작 중 이미 리미트 스위치가 눌려진 방향으로 이동하면 탈조 및 물리적 손상을 받을 가능성이 있기에 에러처리와 동시에 부저음 및 명령어를 무시한다. **[2] 위치 초기화의 경우 모든 모터가 멈춰있는 상태에서만 실행된다.** => 위치 초기화의 경우 어느 한 모터라도 동작 와중에는 제어하지 않도록 설계했다. (에러처리와 동시에 무시한다)

[3] 모터의 동작 와중에 속력은 가장 먼저 동작되던 모터의 속도를 따른다.

=> 버전 별로 두 가지가 존재한다.

V1 펌웨어의 경우 모터 명령어의 연산 속도가 빨라 모터의 최고속도의 범위가 크지만, 다중모터 동작 환경에서는 가장 먼저 동작되던 모터의 빠르기만을 가진다.

이 경우에는 모터의 step_delay 값을 세 모터 모두 일정하게 설정해야한다.

특히, 기준 step_delay 가 작을 경우 모터의 절대적인 회전속도보다 빨라 모터가 다 돌기 전에 명령어를 주게 되어 모터가 돌지 않거나, 기준 출력 전류, 전압의 한계치로 모터가 돌지 않을 가능성이 있다.

(예제 명령어 참조)

V2 펌웨어의 경우 모터 명령어의 연산 속도의 증대로 최고 속도가 느리지만, 다중 모터 동작 환경에서 각각의 step_delay 를 따로 연산해 주는 것이 가능하다.

그러나, 모터 속도가 현저하게 낮아 권장하지 않는다.

[4] 모터의 제어 원리 중 overshoot, undershoot 에 대한 내용을 참조해 제어한다.

- 같은 움직임이라도 Microstep1(1.8 도) * 200 번으로 360 도를 만드는 것과

Microstep256(1.8 도 / 256) * 51200 번으로 360 도를 만드는 것은 다음과 같은 차이가 있다.

1. 한 스텝 이동 시 어떨 땐 이동하고, 어떨 땐 반대로 가는 듯이 보인다.

- 모터가 스텝 당 위치를 잡을 때 마다 정확한 자리를 잡기 위해 미세하게 정방향, 역방향을 운동하며 방향을 찾아간다.(overshoot, undershoot 현상)

- 이 현상으로 인해 1.8 도를 한 스텝으로 두고 한번 이동하는 것과 1.8 도를 256 스텝으로 나누어 256 번 회전하는 것 중 동작 자체는 256 스텝으로 회전하는 것이 안정적이다.

- 동작이 어떨 땐 이동하고, 어떨 땐 반대로 가는 것 같다면 스텝을 정교하게 나누어 제어하면 해결된다.

2. 모터의 소음이 진동하면서 크게 발생한다.

- 스텝 모터가 한 스텝 움직일 때마다 로터가 제 위치를 잡을 때 약간 울리는 소음이 발생한다.

- 이 소음은 모터의 자연 공명 주파수로, 모터 구조로 인해 발생한다. 일반적으로 모터의 스텝 분해능이 높은데(Microstep1 에 가깝게) 저속으로 회전할 때 가장 크게 발생한다.

- 이를 해결하기 위해 마찬가지로 모터 분해능을 정교하게 변경한다. (MicroStep1 로 저속으로 진행할수록 크게 들리지만, MicroStep256 으로 저속으로 진행하면 소음과 진동이 현저하게 감소한다.)

[5] 스텝 모터의 동작 와중에는 서보모터와 LED 를 제어할 수 없다.

- 스텝 모터의 동작 와중에는 오로지 스텝모터의 또 다른 동작, 스텝모터의 정지만 가능하도록 설계했다.

1. control(int, void*)

- 파라미터에 따라 step 모터, servo 모터, LED 를 제어하는 코드이다.

- step 모터, servo 모터, LED 객체 값을 set 명령을 통해 받아 제어하는 원리이다.

- 크게 다섯 가지의 순서로 진행된다. V1, V2 펌웨어 모두 큰 틀에서 같은 원리로 동작한다.

[1] 동작 시작 전 control_init() 메소드를 호출한다.

[2] 동작 명령어가 '위치 초기화' 명령인지 확인한다.

[3] 현재 진행하고자 하는 방향이 리미트 스위치가 눌린 방향인지 참조하여 확인한다.

[4] 모터의 위급 상황 플래그, 정지 플래그를 확인하며 모터를 동작 시킨다.

[5] 동작 종료 시 control_init(int, void*, int) 메소드를 호출하고 종료한다.

2. init(int, void*)

- 맨 초기 보드 부팅 시 모터 동작을 체크하기 위해 작동하는 코드이다.

- 스텝모터 3 개 와 서보모터 2 개를 제어한다.

3. controlling(int, void*)

- 모터의 동작 와중에 시리얼 명령어가 수신됐을 시에 호출되는 메소드이다.

- control 과 동작 구성은 같으나, 제어하고자 하는 모터의 동작 flag 및 데이터 값을 수정하는 것 외는 별도로 step 명령을 제어하지 않는다.

- controlling 메소드가 호출되려면 항상 control 루프문 동작 와중에만 호출되기 때문에 controlling 에서는 데이터만 수정해 전달하면 control 루프문에서 해당 데이터를 참조해 모터 명령어를 수행하는 방식이다.

4. control_init(int, void*, int)

- control 호출 직후, 모터 동작 완료 후 기타 변수 데이터를 정리하는 메소드

- 내부에서 control_data_update 호출 후 파라미터에 따라 control_imergency_init(), 모터 동작 완료 후에는 control_imergency_init(), control_stepping_init(), control_step_motor_data_init()을 진행한다.

5. control_data_update(void*, int)

- 모터 제어에 사용되는 step_motor_control_data 변수를 set 하는 메소드

6. control_imergency_init()

- 리미트 스위치 또는 비상 스위치 발생 시 인터럽트에 의해 변화된 flag 변수들을 모두 초기화 해주는 메소드

7. control_stepping_init()

- 모터 동작 flag 를 모두 초기화 해주는 변수

8. control_step_motor_data_init()

- 모터 동작을 위해 set 된 step_motor_control_data 변수들을 memset 으로 초기화해주는 변수

모터 동작 시 유의해야 할 점

모터 동작 시 유의사항
[1] step 모터 동작 이전에 리미트 스위치의 상태를 확인해 이미 리미트가 눌린 방향으로 진행하는 명령어는 무시한다. => 모터의 동작 중 이미 리미트 스위치가 눌려진 방향으로 이동하면 탈조 및 물리적 손상을 받을 가능성이 있기에 에러처리와 동시에 부저음 및 명령어를 무시한다. **[2] 위치 초기화의 경우 모든 모터가 멈춰있는 상태에서만 실행된다.** => 위치 초기화의 경우 어느 한 모터라도 동작 와중에는 제어하지 않도록 설계했다. (에러처리와 동시에 무시한다) **[3] 모터의 동작 와중에 속력은 가장 먼저 동작되던 모터의 속도를 따른다.** => 버전 별로 두 가지가 존재한다. V1 펌웨어의 경우 모터 명령어의 연산 속도가 빨라 모터의 최고속도의 범위가 크지만, 다중 모터 동작 환경에서는 가장 먼저 동작되던 모터의 빠르기만을 가진다. 이 경우에는 모터의 step_delay 값을 세 모터 모두 일정하게 설정해야한다.

모터 동작 시 유의해야 할 점

특히, 기준 step_delay 가 작을 경우 모터의 절대적인 회전속도보다 빨라 모터가 다 돌기 전에 명령어를 주게 되어 모터가 돌지 않거나, 기준 출력 전류, 전압의 한계치로 모터가 돌지 않을 가능성이 있다.

(예제 명령어 참조)

V2 펌웨어의 경우 모터 명령어의 연산 속도의 증대로 최고 속도가 느리지만, 다중 모터 동작 환경에서 각각의 step_delay 를 따로 연산해 주는 것이 가능하다.

그러나, 모터 속도가 현저하게 낮아 권장하지 않는다.

결론적으로, 모터를 다중제어하기 위해선 맨 처음 동작 모터의 step_delay 가 다른 두 개의 모터에서도 모두 동작 가능한 범위여야한다.

[4] 모터의 제어 원리 중 overshoot, undershoot 에 대한 내용을 참조해 제어한다.

- 같은 움직임이라도 Microstep1(1.8 도) * 200 번으로 360 도를 만드는 것과

Microstep256(1.8 도 / 256) * 51200 번으로 360 도를 만드는 것은 차이가 있다.

1. 한 스텝 이동 시 어떨 땐 이동하고, 어떨 땐 반대로 가는 듯이 보인다.

- 모터가 스텝 당 위치를 잡을 때 마다 정확한 자리를 잡기 위해 미세하게 정방향, 역방향을 운동하며 방향을 찾아간다(overshoot, undershoot 현상)

모터 동작 시 유의해야 할 점

- 이 현상으로 인해 1.8 도를 한 스텝으로 두고 한번 이동하는 것과 1.8 도를 256 스텝으로 나누어 256 번 회전하는 것 중 동작 자체는 256 스텝으로 회전하는 것이 안정적이다.

- 동작이 어떨 땐 이동하고, 어떨 땐 반대로 가는 것 같다면 스텝을 정교하게 나누어 제어하면 해결된다.

2. 모터의 소음이 진동하면서 크게 발생한다.

- 스텝 모터가 한 스텝 움직일 때마다 로터가 제 위치를 잡을 때 약간 울리는 소음이 발생한다.

- 이 소음은 모터의 자연 공명 주파수로, 모터 구조로 인해 발생한다. 일반적으로 모터의 스텝분해능이 높은데(Microstep1 에 가깝게) 저속으로 회전할 때 가장 크게 발생한다.

- 이를 해결하기 위해 마찬가지로 모터 분해능을 정교하게 변경한다. (MicroStep1 로 저속으로 진행할수록 크게 들리지만, MicroStep256 으로 저속으로 진행하면 소음과 진동이 현저하게 감소한다.)

[5] 스텝 모터의 동작 와중에는 서보모터와 LED 를 제어할 수 없다.

- 스텝 모터의 동작 와중에는 오로지 스텝모터의 또 다른 동작, 스텝모터의 정지만 가능하도록 설계했다.

기타 참고 자료

- 라이브러리

라이브러리
[1] step 모터 => #include <HighPowerStepperDriver.h> - pololu 사의 스텝모터 아두이노 드라이버를 사용하였다.(https://github.com/pololu/high-power-stepper-driver-arduino) - 주요 코드는 다음과 같은 원리로 구현되며, 동작원리는 다음과 같다.

```
#include <SPI.h>
#include <HighPowerStepperDriver.h>

const uint8_t CSPin = 4;

// This period is the length of the delay between steps, which controls the
// stepper motor's speed.  You can increase the delay to make the stepper motor
// go slower.  If you decrease the delay, the stepper motor will go faster, but
// there is a limit to how fast it can go before it starts missing steps.
const uint16_t StepPeriodUs = 2000;

HighPowerStepperDriver sd;

void setup()
{
  SPI.begin();
  sd.setChipSelectPin(CSPin);

  // Give the driver some time to power up.
  delay(1);

  // Reset the driver to its default settings and clear latched status
  // conditions.
  sd.resetSettings();
  sd.clearStatus();
```

```
  // Select auto mixed decay.  TI's DRV8711 documentation recommends this
mode
  // for most applications, and we find that it usually works well.
  sd.setDecayMode(HPSDDecayMode::AutoMixed);

  // Set the current limit. You should change the number here to an
appropriate
  // value for your particular system.
  sd.setCurrentMilliamps36v4(1000);

  // Set the number of microsteps that correspond to one full step.
  sd.setStepMode(HPSDStepMode::MicroStep32);

  // Enable the motor outputs.
  sd.enableDriver();
}

void loop()
{
  // Step in the default direction 1000 times.
  sd.setDirection(0);
  for(unsigned int x = 0; x < 1000; x++)
  {
    sd.step();
    delayMicroseconds(StepPeriodUs);
  }

  // Wait for 300 ms.
  delay(300);

  // Step in the other direction 1000 times.
  sd.setDirection(1);
  for(unsigned int x = 0; x < 1000; x++)
  {
    sd.step();
    delayMicroseconds(StepPeriodUs);
  }

  // Wait for 300 ms.
  delay(300);
}
```

1. SPI.begin() : spi 통신을 위한 초기 세팅을 시작한다.

2. sd.setChipSelectPin(CSPin): 모터 드라이버 코드 내용을 살펴보면 몇 번 핀의 드라이버를 사용할지 선택할 수 있다. SPI 통신이므로 CSPin 을 통해 해당 핀에 데이터 전송한다.

3. sd.resetSettings(), sd.clearStatus() : 모터 세팅 초기화 및 클리어

4. sd.setStepMode(): mode 별로 Slow Fast, Automixed 등의 방법이 있다.

5. sd.setCurrentMilliamps36v4(1000) : 모터 전류의 양을 제어 => 회전 속도가 높을수록 높아야한다. 작성한 코드는 3000 까지 높였다.

6. sd.setStepMode(HPSDStepMode::MicroStep32) : 모터 한 스텝 당 회전 각을 조율한다.

데이터 시트를 보면 1~256 의 MicroStep 을 제공한다.

모터 자체에서의 한 스텝은 1.8 도 인데 MicroStep32 으로 세팅하면 한 스텝의 각도가 1.8/32 가 된다. 즉, 한 바퀴를 돌기 위해선 MicroStep1 이 200 스텝 돌아야 360 도가 되고, MicroStep32 의 경우 200*32 = 6400 스텝을 돌아야 한 바퀴가 된다.

7. sd.enableDriver() : 모터 작동 모드를 한다. 만약 `sd.disableDriver()`로 하면 모터에 명령어를 줘도 작동하지 않는다.

8. sd.setDirection(0): 모터 방향 설정. 0 이면 정방향, 1 이면 역방향이다.

9. sd.step() : 모터 한 스텝을 도는 명령이다. 만약 200 스텝을 돌고 싶다면 for 문으로 200 번 실행하면 된다.

10. delayMicroseconds(StepPeriodUs) : 모터 속도 제어는 다음의 delay 로 결정된다. 속도를 빠르게 하고 싶으면 delay 의 값을 조금 주고, 느리게 하고 싶으면 많이 주면 된다. 하지만 전압 상태에 따라 delay 를 조금 줘도 작동이 안될 때가 있다. 따라서 모터 속도가 일정 범위에서 빨라지지 않는다면 위의 5 번을 통해 전류의 양을 증가시켜야한다.

[2] servo 모터 => #include <Servo.h>

```
//해당 핀 번호에 servo 모터를 확인한다.
servo_pan.attach(SERVO_MOTOR_PAN);
servo_tilt.attach(SERVO_MOTOR_TILLT);
delay(1000);

//서보 각도를 조절한다(110 도)
servo_pan.write(110);
delay(1000);
//서보 모터의 작동을 중지시킨다.(detach 를 하지 않으면 모터가 끊임없이 PWM 을 전송하며 모터 버징이 일어난다.)
servo_pan.detach();

delay(1000);

servo_tilt.write(90);
delay(1000);
servo_tilt.detach();
```

1. servo.attach(int pin) : 서보모터 제어를 위한 pwm 신호선 초기화

2. servo.write(int angle) : 서보모터 angle 각도로 이동

3. servo.detach() : 서보모터 pwm 신호 생성 중지

[3] LED => #include <Adafruit_NeoPixel_ZeroDMA.h>

- Adafruit ZeroDMA 라이브러리를 사용했다.(https://learn.adafruit.com/dma-driven-neopixels)

```
switch( ((LED*)value)->LED_id){
  case 1:
      //LED_DATA1 에 LED_NUM(여기선 90)만큼 사용 초기화
      neoPixelLED.init(LED_DATA1, LED_NUM);
      break;
  case 2:
      neoPixelLED.init(LED_DATA2, LED_NUM);
```

```
            break;
        default:
            neoPixelLED.init(LED_DATA1, LED_NUM);
    }
    //밝기 0~255 중 100 으로 선정
    neoPixelLED.setBrightness(100);
     //clear 를 실행하면 기존의 show 된 led 를 모두 off 시킴
    neoPixelLED.clear();
    for(int led_num = 0; led_num < 90; led_num++){
    //각각의 led 번호에 color 를 설정함.
      neoPixelLED.setColor(led_num, ((LED*)value)->R,
      ((LED*)value)->G, ((LED*)value)->B, ((LED*)value)-
      >W );
    }
    //show 를 실행하면 설정된 led 가 켜진다.
    neoPixelLED.show();.
```

- neoPixelLED.init(int pin, int LED_NUM) : pin 번호에 LED_NUM 만큼을 할당해 전송한다.

- neoPixelLED.setBrightness(100) : led 밝기를 조절한다. 0~255 사이를 선택하면 된다.

- neoPixelLED.clear() : 기존의 show 된 led 를 모두 off 시킨다.

- neoPixelLED.setColor(int led_num, int R, int G, int B, int W) : 해당 led 값의 RGBW 값을 세팅한다.

- neoPixelLED.show() : 설정된 LED 를 on 시킨다.