

```

void loop() {
    // put your main code here, to run repeatedly:
    bool bDiffJoyState = false;
    bool bDiffBtnState = false;
    uint8_t u8SwitchValues[HW_COMMON_SWITCH_COUNT_MAX] = {0,};
    unsigned int uJoyValues[JOY_TYPE_END] = {0,};

    bDiffJoyState = getJoystickValues(uJoyValues, JOY_TYPE_END);
    bDiffBtnState = getSwitchValues(u8SwitchValues);

    // Send state through the HID when values are changed.
    if (bDiffJoyState || bDiffBtnState)
    {
        mappingJoystickValues(uJoyValues, JOY_TYPE_END);
        setJoystickValues(uJoyValues, u8SwitchValues);

        print_Joyval(uJoyValues);
        print_SWval(u8SwitchValues);
        if (g_bTestAutoSendMode == false) Joystick.sendState();
    }

    if (g_uHeartBeatCount++ == 1000)
    {
        Serial.print(".");
        g_uHeartBeatCount = 0;
    }
    delay(1);
}

```

(완성된 루프구조)









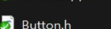


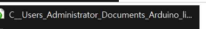









추후 검증이 필요하나 8.2 의 과정을 통해 실물 GCS 제작 후 가능한 것으로 보입니다.

## - 2 차 개발

이전 제작한 F/W 구조에서는 샘플용 코드여서 수정 및 운용에서 어려운 부분이 많았습니다.

따라서 저는 기존의 샘플 코드를 참고하여 자체적으로 새로운 F/W 구조를 개발하기로 하였습니다.

먼저 폴더 구조는 아래와 같습니다.

	PSL_GCS		
		PSL_GCS.ino	
		src (library)	
			Button
			 
			DynamicHID
			 
			Joystick
			 
			PSL_GCS
			 

해당 폴더 구조는 PSL\_GCS 라는 아두이노 코드 (확장자 명 : .ino)파일을 동작시키기 위한 라이브러리 폴더 (src)를 가진 형태이며, 라이브러리 폴더 안에는 자체 개발한 라이브러리 (PSL\_GCS, Button)과 기존 라이브러리 (DynamicHID, Joystick) 을 포함하고 있습니다.

main 코드와 라이브러리 항목에 대해 설명드리기 위해 **main 코드 먼저 설명**드리자면 아래와 같습니다.

```
PSL_GCS.ino > loop()
1  #include "src/PSL_GCS/PSL_GCS.h"
2
3  PSL_GCS_ CapstoneGCS;
4
5  uint16_t countHeartBeat;
6
7  static void btn1ChangeISR();
8  static void btn2ChangeISR();
9
10 void setup(){
11     delay(10);
12     Serial.begin(115200);
13
14     analogReference(DEFAULT);
15
16     attachInterrupt(digitalPinToInterrupt(CapstoneGCS.btn_mission1.returnPin()), btn1ChangeISR, CHANGE);
17     attachInterrupt(digitalPinToInterrupt(CapstoneGCS.btn_mission2.returnPin()), btn2ChangeISR, CHANGE);
18
19     delay(10);
20 }
```

PSL\_GCS 의 Class 생성 후 버튼 2 개에 대한 인터럽트 서비스 루틴 설정을 추가하였습니다.

위 인터럽트 서비스 루틴을 통해 버튼에 들어오는 Noise 를 디바운싱 할 수 있습니다.

```
void loop(){

    // Main Processes
    CapstoneGCS.getJoystickValues();
    CapstoneGCS.getButtonValues();

    CapstoneGCS.btn_mission1.debounceButton();
    CapstoneGCS.btn_mission2.debounceButton();

    CapstoneGCS.processModeChange();
    if(CapstoneGCS.getDiffState()) {

        if(CapstoneGCS.getStateMode() == TransData) {
            CapstoneGCS.updateJoystickValues();
            CapstoneGCS.sendGcsData();
        }

        //CapstoneGCS.updateSignalValues();
        //CapstoneGCS.printCurrentValues();
    }else {
        countHeartBeat++;
        if(countHeartBeat >= 1000){
            Serial.print(".");
            countHeartBeat = 0;
        }
    }
    delay(1);
}
```

=> HW 날 값 추출 (getJoystickValues, getButtonValues)

RaspberryPi 로부터 오는 Signal 값은 모델 학습 최종 종료 후 진행 예정

=> 미션 트리거가 되는 “특정 시간 이상 버튼 누르기” 과정에서 디바운싱 처리를 합니다 (debounceButton)

=> 객체 감지 후, 자동화 조종을 위한 HW 값 차단 모드인지 차단 해제 모드인지에 따라 다르게 동작합니다.

만약 자동화 조종 모드 일 시, RaspberryPi 로부터 받은 Signal 을 토대로 정해진 축 값을 통해 이동합니다.

자동화 조종 모드가 아닐 시, 앞서 추출한 HW 값을 바탕으로 데이터를 Window 상으로 전송합니다.

이 동작은 HW 값에 변동이 있을 시에만 동작합니다. < if(CapstoneGCS.getDiffState()) >

=> HW 값에 변동이 없다면 Arduino 동작이 원활하게 되는지 외적인 변동이 없기에

HeartBeat 를 추가하여 print 되도록 하였습니다.

=> delay(1)은 위 동작을 너무 빨리 반복할 필요는 없기에 추가하였습니다.

**라이브러리 하위 항목 별로 설명** 드리자면 아래와 같습니다.

- DynamicHID : HID 통신을 위한 기본적인 코드가 들어가 있습니다. Joystick Library 가 이를 활용해 윈도우 상에서 HID 장치로 인식할 수 있게끔 합니다.

```
class DynamicHID_ : public PluggableUSBModule
{
public:
    DynamicHID_(void);
    int begin(void);
    int SendReport(uint8_t id, const void* data, int len);
    void AppendDescriptor(DynamicHIDSubDescriptor* node);

protected:
    // Implementation of the PluggableUSBModule
    int getInterface(uint8_t* interfaceCount);
    int getDescriptor(USBSetup& setup);
    bool setup(USBSetup& setup);
    uint8_t getShortName(char* name);

private:
    #ifdef _VARIANT_ARDUINO_DUE_X_
        uint32_t epType[1];
    #else
        uint8_t epType[1];
    #endif

    DynamicHIDSubDescriptor* rootNode;
    uint16_t descriptorSize;

    uint8_t protocol;
    uint8_t idle;
};
```

위 Class 구조에서 Joystick 은

customHidReportDescriptor 을 만들어 sendReport 를 통해 데이터를 전송한다고 보시면 됩니다.

- Joystick : Joystick 의 기본적인 코드가 들어가 있습니다. 주로 실제 조종기에서 모두 조이스틱을 쓰지만 어떤 것은 Rudder, Throttle, Rx,Ry 등등 명칭이 다 다르기에 HW 에서 추출한 값에 명칭을 붙여 전달하기 위한 용도로 쓰입니다.

```

// Set Axis Values
void setXAxis(int16_t value);
void setYAxis(int16_t value);
void setZAxis(int16_t value);
void setRxAxis(int16_t value);
void setRyAxis(int16_t value);
void setRzAxis(int16_t value);

// Set Simulation Values
void setRudder(int16_t value);
void setThrottle(int16_t value);
void setAccelerator(int16_t value);
void setBrake(int16_t value);
void setSteering(int16_t value);

void setButton(uint8_t button, uint8_t value);
void pressButton(uint8_t button);
void releaseButton(uint8_t button);

void setHatSwitch(int8_t hatSwitch, int16_t value);

void sendState();

```

class 자체가 좀 커서 모두 담긴 어려워 핵심 부분만 담았습니다.

위 함수들을 사용하여 X 축 값, Y 축 값, Rudder 축 값, 버튼 값 등의 데이터를 특정 배열에 셋업 후 sendState 를 통해 window 에 전송하게 됩니다.

- PSL\_GCS : 이번 PrecisionLanding 을 위한 Class 를 제작한 라이브러리입니다.

총 구성해야 하는 기능은 아래와 같습니다

1. HW 값 추출 후 저장( Joystick, Button, Signal - from RaspberryPi )
2. HW 데이터 가공 (Invert)
3. 특수 기능 추가 ( ModeChange , autoMode )
4. 디버깅 기능 추가 ( printCurrentValues - 상태값 출력 )

```

class PSL_GCS_
{
private:
    bool State_mode;
    bool State_diffJoysticks;
    bool State_diffButtons;
    bool State_diffSignals;

    uint16_t Value_Joysticks[COUNT_JOYSTICK_MAX];
    bool Value_Buttons[COUNT_BUTTON_MAX];
    uint16_t Value_Signals[COUNT_SIGNAL_MAX];

    uint32_t Time_modeChange;

    joystick_hw_config_t hw_config = {
        .PIN_XAxis      = A0,
        .PIN_RudderAxis = A1,
        .PIN_YAxis      = A3,
        .PIN_ThrottleAxis = A2,

        .HW_INVERT_RUDDER = true,
        .HW_INVERT_THROTTLE = false,
        .HW_INVERT_X = false,
        .HW_INVERT_Y = false
    };
};

public:
    PSL_GCS_();

    Joystick_ joystick;

    Button_ btn_mission1;
    Button_ btn_mission2;

    uint16_t Value_Update_Joysticks[COUNT_JOYSTICK_MAX];
    bool Value_Update_Buttons[COUNT_BUTTON_MAX];
    uint16_t Value_Update_Signals[COUNT_SIGNAL_MAX];

    void getJoystickValues();
    void getButtonValues();
    void getSignalValues();
    bool getStateMode();

    void updateJoystickValues();
    void processModeChange();
    void processSignal();

    void sendGcsData();

    uint16_t invertValue(uint16_t value);

    bool getDiffState();
    void printCurrentValues();
};

```

- Button : HW 장치 중 Button 에 대한 Class 를 작성하였습니다.

디바운싱, 상태 값 출력, 미션을 위한 버튼 플래그 생성 등의 기능이 있습니다.

```
class Button_{
private:
    uint8_t PIN;
    uint32_t Time_pressStart;
    bool State_Button;

    bool State_isButtonPushed;

public:
    Button_(uint8_t pin);

    void ISR_isChange();
    void debounceButton();

    bool getButtonStateRealtime();
    bool getButtonPushed();
    void set_isPushedToFalse();

    uint8_t returnPin();
};
```

아래는 부가 기능들에 대한 자료입니다.

<HW 구성에 따른 값 Mapping 기능>

- HW 장비들의 방향에 따라 값이 달라지므로 이를 고려해 값을 매핑해줄 수 있는 기능을 제작하였습니다.

(동일 제품 조이스틱이 HW 구성에서 0 도 돌아가서 들어간 것과 180 도 돌아서 삽입된 것은 상하좌우 값을 반전시킴)

```
joystick_hw_config_t hw_config = {
    .PIN_XAxis      = A0,
    .PIN_RudderAxis = A1,
    .PIN_YAxis      = A3,
    .PIN_ThrottleAxis = A2,

    .HW_INVERT_RUDDER = true,
    .HW_INVERT_THROTTLE = false,
    .HW_INVERT_X = false,
    .HW_INVERT_Y = false
};

void PSL_GCS::updateJoystickValues(){
    if(hw_config.HW_INVERT_RUDDER) joystick.setRudder(invertValue(Value_Joysticks[RudderAxis]));
    else joystick.setRudder(Value_Joysticks[RudderAxis]);

    if(hw_config.HW_INVERT_THROTTLE) joystick.setThrottle(invertValue(Value_Joysticks[ThrottleAxis]));
    else joystick.setThrottle(Value_Joysticks[ThrottleAxis]);

    if(hw_config.HW_INVERT_X) joystick.setXAxis(invertValue(Value_Joysticks[XAxis]));
    else joystick.setXAxis(Value_Joysticks[XAxis]);

    if(hw_config.HW_INVERT_Y) joystick.setYAxis(invertValue(Value_Joysticks[YAxis]));
    else joystick.setYAxis(Value_Joysticks[YAxis]);
}
```

<PSL\_GCS 생성자 하위 파라미터 자동화>

```

PSL_GCS::PSL_GCS() : joystick(JOYSTICK_DEFAULT_REPORT_ID,
    JOYSTICK_TYPE_JOYSTICK, COUNT_BUTTON_MAX, 0,
    true, true, false,
    false, false, false,
    true, true,
    false, false, false),
    btn_mission1(BTN_BASE_PIN),
    btn_mission2(BTN_BASE_PIN+1) {}

```

생성자에 joystick 과 Button 생성자를 추가하여 자동으로 하위 항목도 생성되도록 설정하였습니다.

#### <미션 트리거가 되는 버튼 동작>

먼저 Interrupt 함수로 버튼의 상태값이 반전 될 시, 상태값을 받아옵니다.

```

void Button_::ISR_isChange(){
    if(digitalRead(PIN)) State_Button = Not_pushed;
    else State_Button = Pushed;
}

```

이후 main loop 에서 동작하는 디바운스 함수로 실제 사람이 버튼을 길게 누른 것인지 노이즈인지 판별합니다.

```

void Button_::debounceButton(){
    if(State_Button == Pushed){
        Time_pressStart++;
    }
    else{
        if(Time_pressStart >= 10){
            if(Time_pressStart >= ValidTime){
                Serial.print("Detect user push button [");
                Serial.print(PIN);
                Serial.print("] while :");
                Serial.println(Time_pressStart);
                State_isButtonPushed = true;
                Time_pressStart = 0;
            }
            else{
                Serial.println("Not yet.. holding push state more...");
                Time_pressStart = 0;
            }
        }
    }
}

```

판별이 되었다면 미션 수행을 해도 된다는 Flag 인 State\_isButtonPushed 를 활성화 해줍니다.

마지막으로 미션 수행 후 해당 State\_isButtonPushed 를 비활성화 해줍니다.

```

void PSL_GCS_::processModeChange(){
    if(btn_mission1.getButtonPushed()) {
        State_mode = !State_mode;
        Serial.print("Mode Change!! [ ");
        if(State_mode) Serial.print("TransData");
        else Serial.print("DontTransData");

        Serial.print(" ] to [ ");
        if(State_mode) Serial.println("DontTransData ");
        else Serial.println("TransData ");

        btn_mission1.set_isPushedToFalse();
    }
}

```

## Mode Change - TransData 모드 일 시, MissionPlanner 앱에서 정상적으로 인식하는 모습

```
01:57:01.486 -> Mode Change!! [ DontTransData ] to [ TransData ]
01:57:04.131 -> .....
```



## Mode Change - DontTransData 모드 일 시, MissionPlanner 앱에서 받지 않는 모습

```
01:57:01.486 -> Mode Change!! [ DontTransData ] to [ TransData ]
01:57:04.131 -> .....Detect user push button [2] while :1828
01:58:43.720 -> Mode Change!! [ TransData ] to [ DontTransData ]
01:58:44.989 -> ...
```



보다 자세한 시연 과정은

“ Capstone\_Design\1.PrecisionLandingModule\2.Version\_1.0\2.GroudControlStation\1.Src\Test ” 내 영상을 참조해주시면 됩니다.

### [참고 자료]

Capstone\_Design\1.PrecisionLandingModule\2.Version\_1.0\2.GroudControlStation\1.Src\joystick 의 joystick.ino 참조

Capstone\_Design\1.PrecisionLandingModule\2.Version\_1.0\2.GroudControlStation\1.Src\PSL\_GCS 폴더 참조

기록자	김교원	점검자	박정은
서명	(sign)	서명	(sign)