

임무 소프트웨어 Guide

투하장치

msw_sparrow_gun

목 차

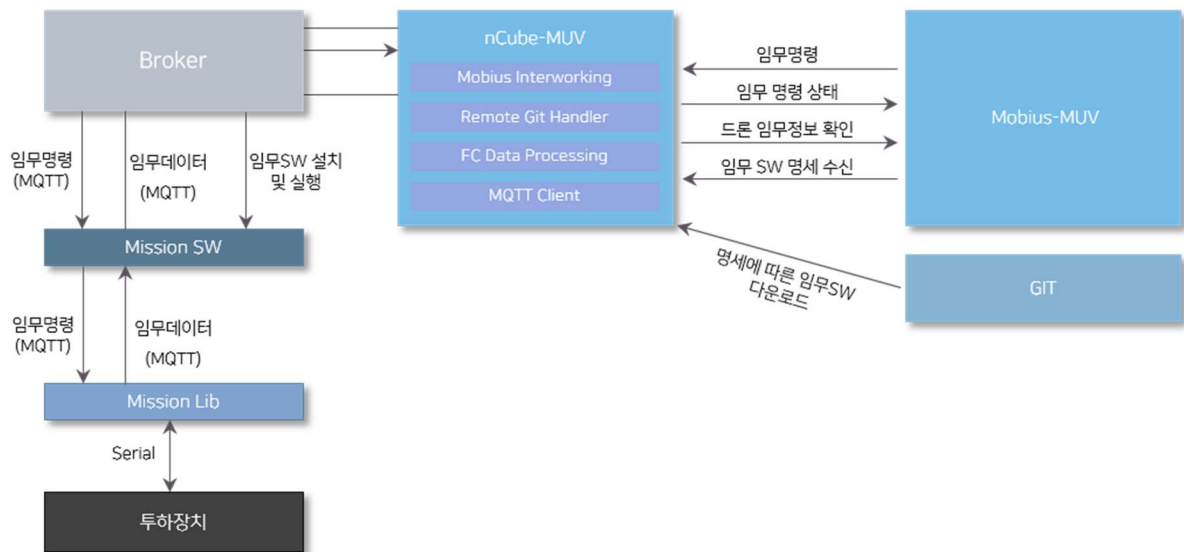
| | | |
|-----|-------------------------|----|
| 1 | 개요 | 3 |
| 2 | 투하장치 | 4 |
| 2.1 | 투하장치란?..... | 4 |
| 2.2 | 투하장치 탑재 | 5 |
| 3 | 임무Lib 및 임무SW 개발 | 6 |
| 3.1 | 임무Lib, 임무SW와 개발도구 | 6 |
| 3.2 | 임무Lib 개발 | 6 |
| 3.3 | 임무SW 개발 | 13 |

1 개요

이 문서는 중앙에서 임무 컴퓨터(Mission Computer, MC)를 통해 드론에 탑재된 투하장치를 제어하기 위한 방법을 설명하는 문서이다. 투하장치는 8x8 형태로 64개의 센서 탑재가 가능하다.

사용자는 투하장치를 제어하기 위한 제어 명령으로 중앙에 투하할 센서의 위치 좌표(예, 4,6)를 전송하고 중앙은 이 명령을 드론에 탑재된 MC에 전송한다. MC는 전송 받은 명령을 파싱하여 투하장치 데이터 프로토콜에 맞춰 패킷을 만들고 Serial 통신을 통해 투하장치에 명령을 전송한다. 최종적으로 명령을 전달 받은 투하장치는 제어 명령에 포함된 위치의 센서를 투하한다.

이를 위해 투하장치 HW에 대해 설명하고 임무장비와 직접적으로 연결되어 제어하는 임무라이브러리(Library, Lib), 임무 Lib를 포함한 형태로 nCube-MUV와 연동되어 외부와 통신하며 임무 Lib를 제어하는 임무 소프트웨어(Software, SW)에 대해 설명하고 동작하기 위한 방법을 설명한다.



< 투하장치 임무 소프트웨어(msw_sparrow_gun) 아키텍처 >

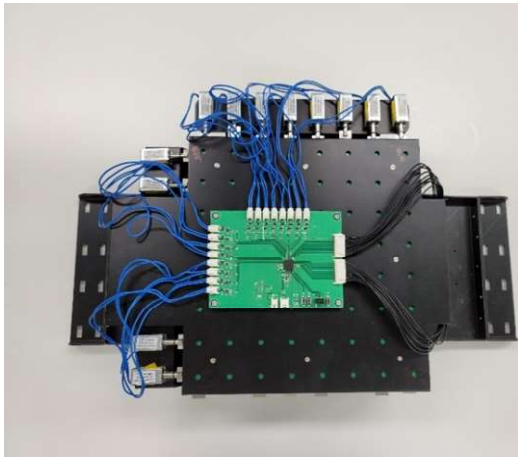
2 투하장치

2.1 투하장치란?

투하장치는 드론에 탑재되어 MC와 통신하며 중앙에서 전달받은 명령을 통해 장착된 센서를 투하하는 장치이다. 투하장치는 x축, y축에 솔레노이드가 설치되어 각 축의 솔레노이드를 움직여 센서를 장착 및 해제가 가능하고 제어보드에서 MC와 통신하며 중앙으로부터 명령을 전달받아 각 좌표에 해당하는 솔레노이드를 동작하여 센서를 투하할 수 있다.

- 투하장치 구성요소

- 투하장치: 8x8 형태로 64개의 센서 장착 가능하고 솔레노이드를 통해 센서 고정 및 투하가 가능한 장치
- 투하장치 제어보드: Serial 포트, 12V 전원 포트, 솔레노이드 연결 포트로 구성되어 MC와 Serial 포트를 통한 통신, 투하장치의 솔레노이드 제어가 가능한 보드



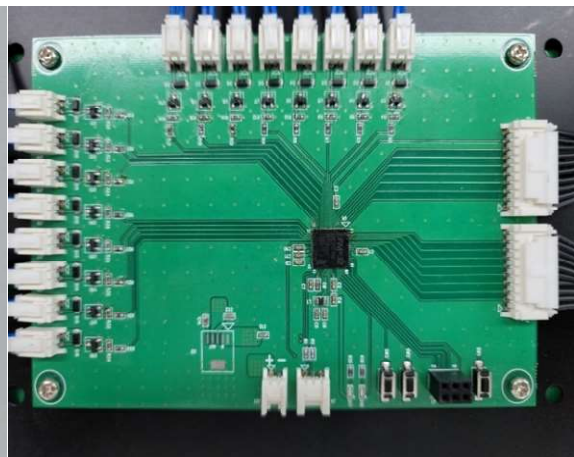
<투하장치 전체모습>



<센서가 장착된 모습>



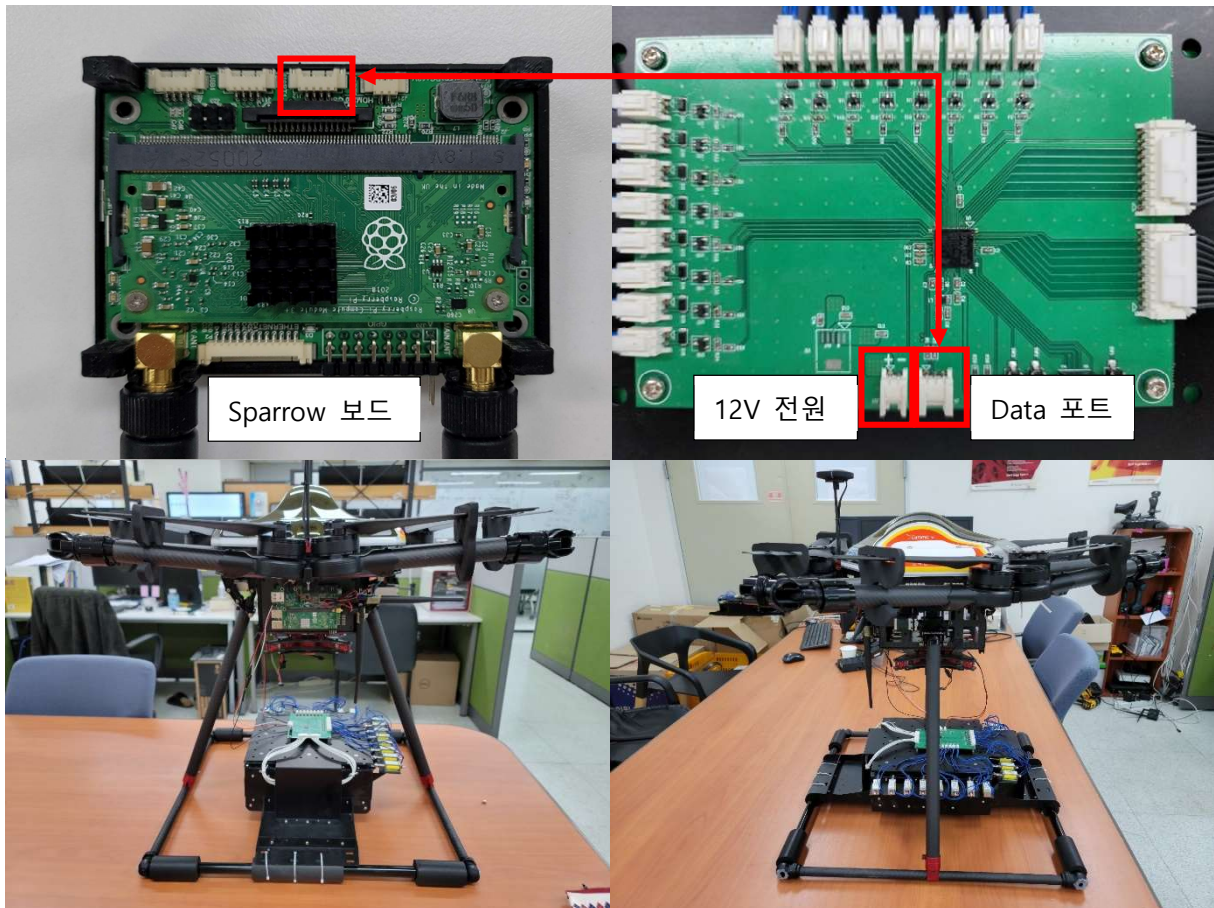
<투하장치 센서>



<투하장치 제어 보드>

2.2 투하장치 탑재

- 투하장치를 드론의 하단에 튼튼하게 고정시켜 장착한다.
- 투하장치의 Serial 포트를 MC라고 부르는 Sparrow Mini 보드의 Serial 포트 (/dev/ttyUSB3)에 연결한다.
- 전원은 12V를 사용하며 Serial 포트 옆의 Power 포트에 연결한다.



<드론에 탑재된 투하장치와 전원 및 Serial포트 연결>

3 임무Lib 및 임무SW 개발

3.1 임무Lib, 임무SW와 개발도구

임무Lib는 임무 장비와 직접적으로 연결되어 임무 장비를 제어하는 소프트웨어이다. 필수 구현 요소로 임무 장비와 연결하기 위한 인터페이스(Serial, I2C, PWM 등) 연동, 내부적으로 임무SW와 통신하기 위한 MQTT 통신에 대한 내용이 있다. 또한 임무 장비, 사용자 또는 환경에 따라 데이터 모델 정의, 통신 모델 정의 등의 내용을 추가할 수 있다. 임무Lib를 임무SW에서 사용을 위해 개발자가 사전에 개발하여 응용프로그램 형태로 배포해야 한다.

임무SW는 임무Lib를 포함한 형태로 개발되며 외부와 통신하며 임무Lib를 제어하는 소프트웨어이다. 임무SW는 응용프로그램 형태의 임무Lib를 실행하기 위한 스크립트를 정의하는 내용이 필수이며, 또한 외부와 통신하기 위해 통신 모델을 정의하고 데이터 모델을 정의하는 내용을 필수로 한다.

개발자는 개발도구를 통해 임무SW를 좀 더 쉽게 개발하고 배포하여 드론에 탑재까지 가능하다. 개발도구는 블록코딩 형식으로 개발할 수 있으며 Github와 연동하여 이미 배포된 임무Lib와 임무SW를 재사용이 가능하고 개발된 임무SW를 배포할 수 있으며 드론과 임무에 대한 명세를 작성함으로써 자동으로 드론에 탑재와 동작이 가능하다.

3.2 임무Lib 개발

- 투하장치를 제어하기 위해 임무장비와 직접적으로 연결되어 제어가 가능한 임무Lib를 사전에 개발한다. 임무Lib는 MC에 탑재되어 투하장치와 직접적인 연결과 투하장치를 실질적으로 제어하는 역할을 수행한다. 임무Lib에는 아래의 목록이 필수적으로 구현되어야 한다. 개발된 임무Lib는 최종적으로 응용프로그램 형태로 뒤에 설명하는 임무SW에 포함된 형태로 동작한다.

- 임무Lib는 크게 4가지 부분으로 나뉜다.

① MC와 내부적으로 데이터 통신을 위한 MQTT 통신 예시코드

```
def msw_mqtt_connect(broker_ip, port):
    global lib
    global lib_mqtt_client
    global control_topic
    global data_topic
    global req_topic

    lib_mqtt_client = mqtt.Client()
    lib_mqtt_client.on_connect = on_connect
    lib_mqtt_client.on_disconnect = on_disconnect
    lib_mqtt_client.on_subscribe = on_subscribe
    lib_mqtt_client.on_message = on_message
    lib_mqtt_client.connect(broker_ip, port)
    control_topic = '/MUV/control/' + lib["name"] + '/' + lib["control"][0]
    lib_mqtt_client.subscribe(control_topic, 0)
    lib_mqtt_client.subscribe(req_topic, 0)

    lib_mqtt_client.loop_start()
    return lib_mqtt_client

def on_connect(client, userdata, flags, rc):
    print('[msg_mqtt_connect] connect to ', broker_ip)

def on_disconnect(client, userdata, flags, rc=0):
    print(str(rc))

def on_subscribe(client, userdata, mid, granted_qos):
    print("subscribed: " + str(mid) + " " + str(granted_qos))

def on_message(client, userdata, msg):
    global gun_event
    global data_topic
    global control_topic
    global req_topic
    global con
    global req

    if msg.topic == control_topic:
        con = msg.payload.decode('utf-8')
        gun_event |= CONTROL_E
    elif msg.topic == req_topic:
        req = msg.payload.decode('utf-8')
        gun_event |= DATA_E
```

② 투하장치와 직접적으로 연결하기 위한 인터페이스(Serial) 연동 예시코드

```
def missionPortOpening(missionPortNum, missionBaudrate):
    global missionPort
    global status

    print('Connect to serial...')
    try:
        missionPort = serial.Serial(missionPortNum, missionBaudrate, timeout=2)
        if missionPort.isOpen():
            print('missionPort Open. ', missionPortNum, 'Data rate: ', missionBaudrate)
            status = 'open'
            send_data_to_msw(status)

    except serial.SerialException as e:
        missionPortError(e)
    except TypeError as e:
        missionPortClose()

def missionPortOpen():
    global missionPort

    print('missionPort open!')
    missionPort.open()

def missionPortClose():
    global missionPort
    global status

    status = 'close'
    send_data_to_msw(status)
    print('missionPort closed!')
    missionPort.close()

def missionPortError(err):
    global status

    print('[missionPort error]: ', err)
    status = 'error'
    send_data_to_msw(status)

def send_data_to_msw (obj_data):
    global lib_mqtt_client
    global data_topic
```



```
data_topic = '/MUV/data/' + lib["name"] + '/' + lib["data"][0]
lib_mqtt_client.publish(data_topic, obj_data)

def missionPortData():
    global status
    global req

    if req == "1":
        status = 'alive'
        send_data_to_msw(status)
```

③ 투하장치 제어명령을 투하장치 데이터 프로토콜에 맞게 변환하는 예시코드

```
def request_to_mission():
    global missionPort
    global con

    try:
        if missionPort != None:
            if missionPort.isOpen():
                con_arr = con.split(',')
                if (int(con_arr[0]) < 8) and (int(con_arr[1]) < 8):
                    stx = 'A2'
                    command = '030' + con_arr[0] + '0' + con_arr[1] + '000000000000'
                    crc = 0
                    for i in range(0, len(command), 2):
                        print('crc: ', crc)
                        crc ^= int(command[i+1], 16)
                    if crc < 16:
                        command += ('0' + str(crc))
                    else :
                        command += str(crc)

                    etx = 'A3'
                    command = stx + command + etx

                    msdata = bytes.fromhex(command)
                    missionPort.write(msdata)

    except (ValueError, IndexError, TypeError):
        print ('except Error')
        pass
```

④ 임무 Lib가 시작되는 메인 코드 예시

```
def main():
    global lib
    global lib_mqtt_client
    global missionPort
    global control_topic
    global data_topic
    global req_topic

    global gun_event
    global con
    global req

    my_lib_name = 'lib_sparrow_gun'
    my_msw_name = 'msw'+ my_lib_name[3:] + '_' + 'msw'+ my_lib_name[3:]

    cmd = ['./' + my_msw_name + '/' + my_lib_name, argv[1], argv[2]]
    pid_arr = []
    processWatch = [p.cmdline() for p in psutil.process_iter()].count(cmd)
    if processWatch > 2:
        for p in psutil.process_iter():
            if (p.cmdline() == cmd):
                print(p.pid)
                pid_arr.append(p.pid)
        os.kill(pid_arr[0], signal.SIGKILL)
        os.kill(pid_arr[0]+1, signal.SIGKILL)

    try:
        lib = dict()
        with open('./' + my_msw_name + '/' + my_lib_name + '.json', 'r') as f:
            lib = json.load(f)
            lib = json.loads(lib)

    except:
        lib = dict()
        lib["name"] = my_lib_name
        lib["target"] = 'armv6'
        lib["description"] = "[name] [portnum] [baudrate]"
        lib["scripts"] = './' + my_lib_name + ' /dev/ttyUSB3 9600'
        lib["data"] = ['GUN']
        lib["control"] = ['MICRO']
        lib = json.dumps(lib, indent=4)
        lib = json.loads(lib)

        with open('./' + my_msw_name + '/' + my_lib_name + '.json', 'w', encoding='utf-8') as json_file:
```

```

        json.dump(lib, json_file, indent=4)

lib['serialPortNum'] = argv[1]
lib['serialBaudrate'] = argv[2]

control_topic = '/MUV/control/' + lib["name"] + '/' + lib["control"][0]
data_topic = '/MUV/data/' + lib["name"] + '/' + lib["data"][0]
req_topic = '/MUV/data/' + lib["name"] + '/' + lib["data"][0] + 'req'

msw_mqtt_connect(broker_ip, port)
missionPortOpening(lib['serialPortNum'], lib['serialBaudrate'])

while True:
    if gun_event & CONTROL_E:
        gun_event &= (~CONTROL_E)
        request_to_mission()
    elif gun_event & DATA_E:
        gun_event &= (~DATA_E)
        missionPortData()

if __name__ == "__main__":
    main()

```

- 임무Lib는 임무SW와 연동을 위해 실행파일 형태로 배포한다. 파이썬으로 개발된 투하장치 임무Lib는 아래의 명령어를 통해 응용프로그램 형태로 만들 수 있다.

```
$ python3 -m PyInstaller -F lib_sparrow_gun.py
```

- Python으로 개발되어 응용프로그램 형태로 배포된 투하장치 임무Lib는 아래 Github 주소에서 확인할 수 있다.

https://github.com/loTKETI/lib_sparrow_gun.git

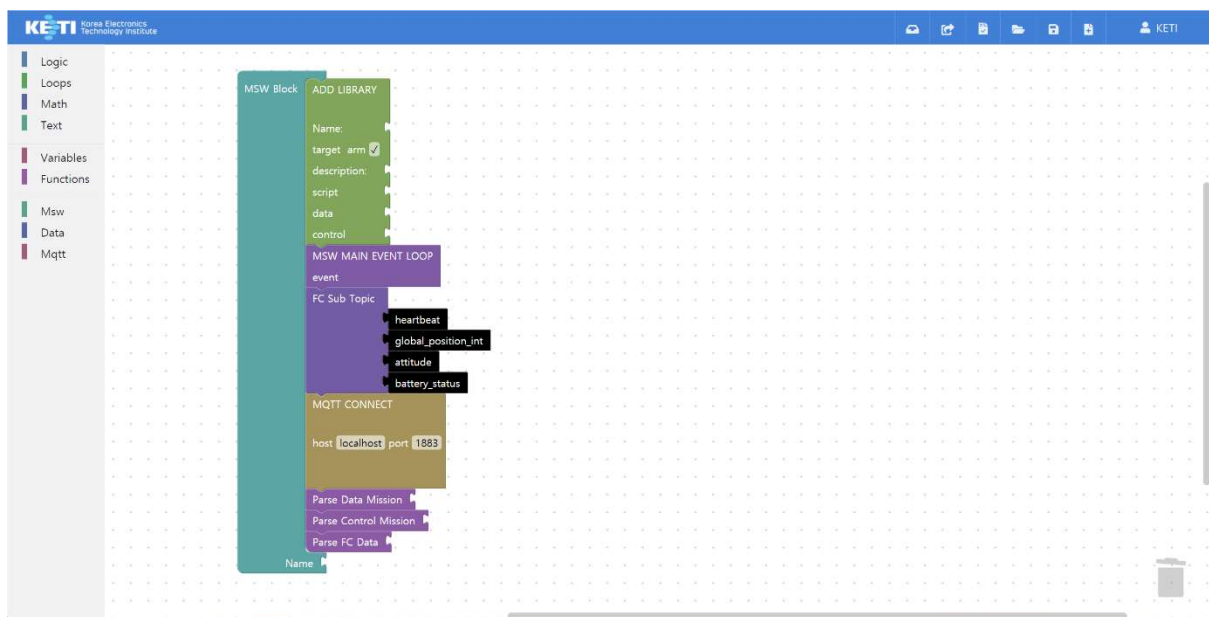
3.3 임무SW 개발

3.3.1 MUV 개발도구

- 개발자가 임무SW를 손쉽게 개발하고 배포하여 MC에 자동으로 탑재까지 가능하도록 지원하는 블록코딩 형태의 웹 기반 지원도구이다.
- <http://203.253.128.177:7505> 주소로 접속하여 로그인하여 사용 가능하다.
- 코딩을 소스코드를 작성하는 형태가 아닌 블록코딩을 통해 쉽게 개발할 수 있으며 Github와 연동하여 업로드가 가능하고 approval이라는 명세를 입력함으로써 드론에 탑재가 가능하다.

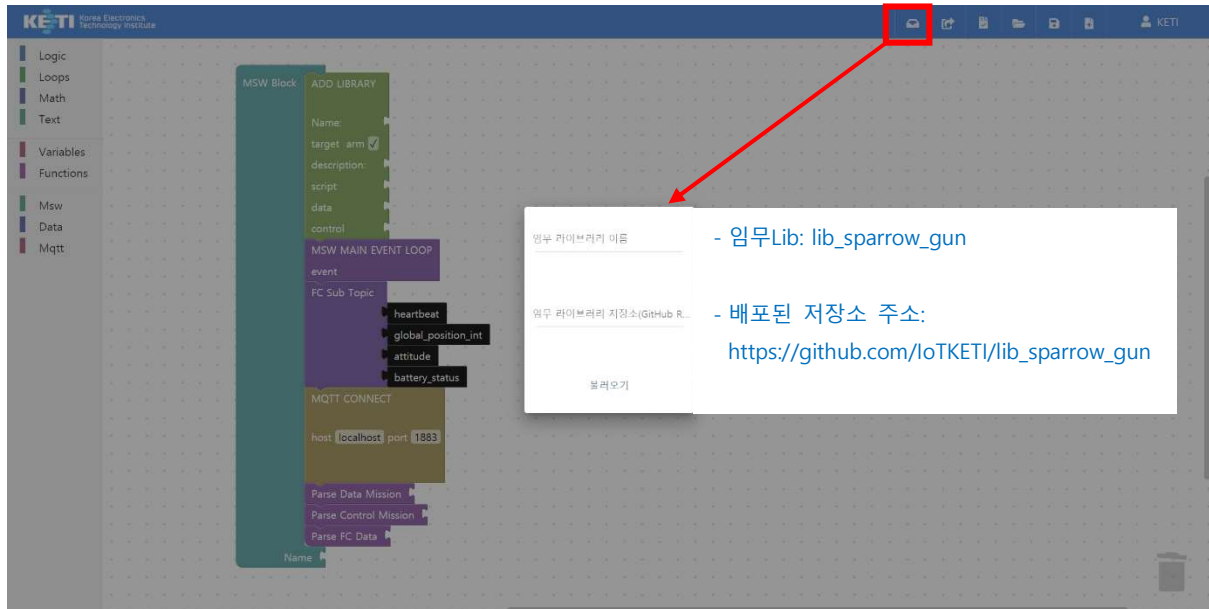
3.3.2 임무 S/W 개발과정

- ① 아래 사진과 같이 임무SW의 기본이 되는 형태까지 블록으로 코딩한다. 이 구조는 임무SW도 동일한 구조를 가진다.

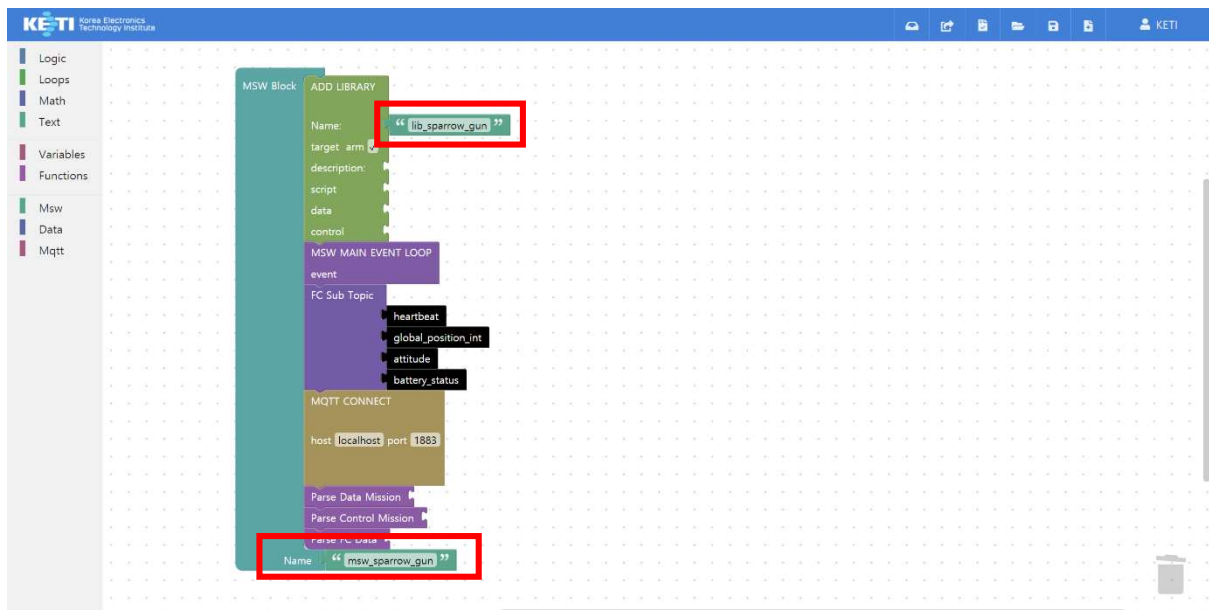


② 사전에 개발해서 배포한 임무Lib를 호출한다.

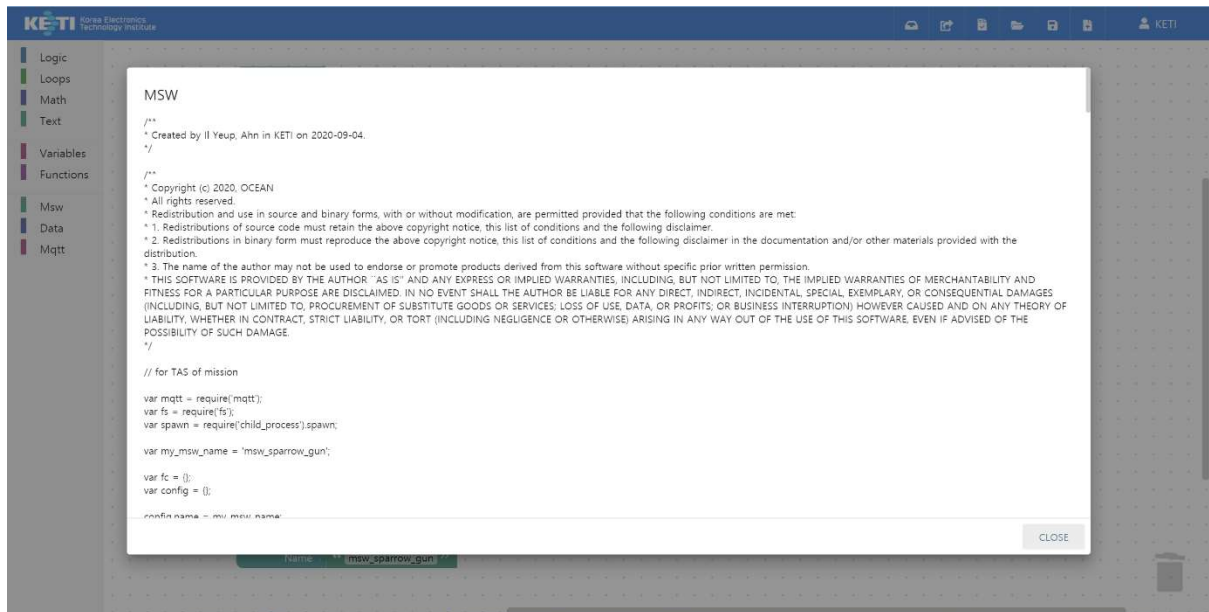
- ✓ 임무Lib 이름: 사전에 응용 프로그램 형태로 배포한 임무Lib의 이름
- ✓ 임무Lib 저장소: Github에 배포된 임무Lib를 가져오기 위한 주소



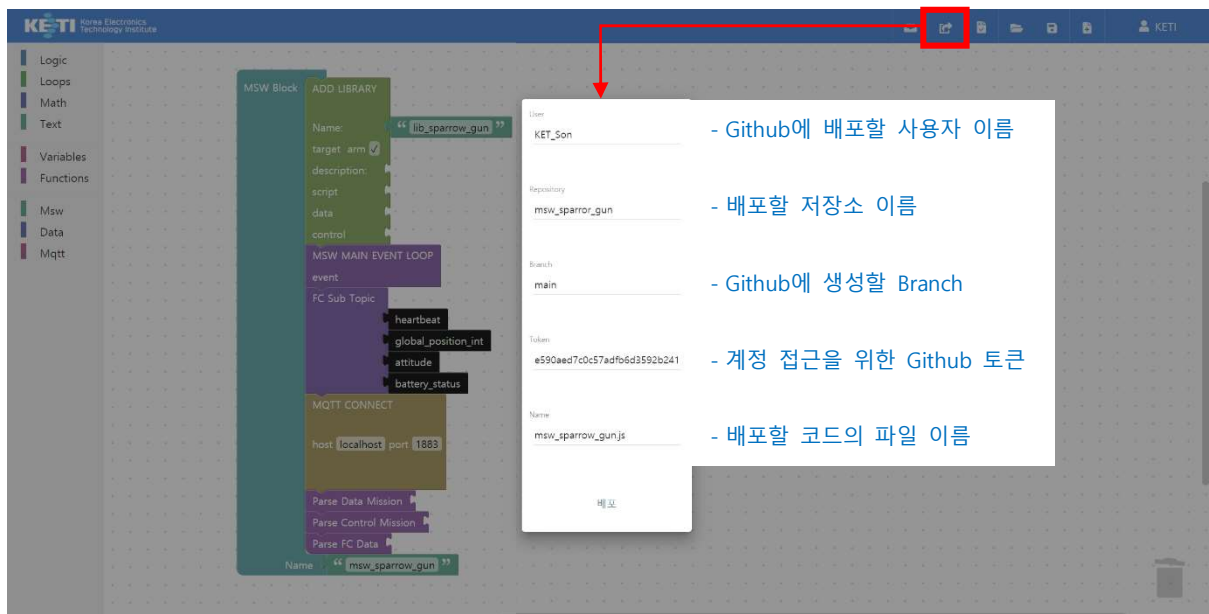
③ 생성된 임무Lib 블록 추가 및 임무SW의 이름 입력



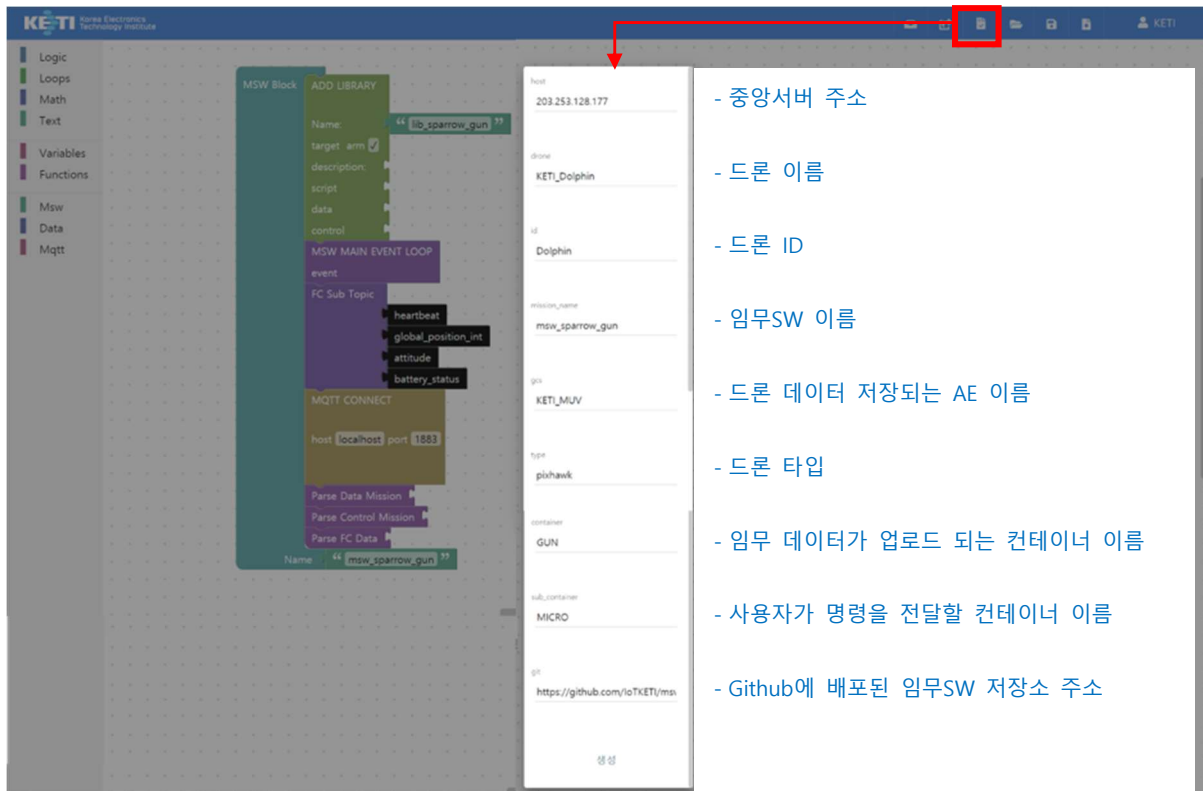
④ 블록으로 코딩한 임무SW를 배포하기 전 소스코드 형태로 확인



⑤ 임무SW 배포에 필요한 정보 입력하여 배포



⑥ 드론 탑재를 위한 드론 명세 작성



host: 203.253.128.177 - 중양서버 주소

drone: KETI_Dolphin - 드론 이름

id: Dolphin - 드론 ID

mission_name: msw_sparrow_gun - 임무SW 이름

gcs: KETI_MUV - 드론 데이터 저장되는 AE 이름

type: pixhawk - 드론 타입

container: GUN - 임무 데이터가 업로드 되는 컨테이너 이름

sub_container: MICRO - 사용자가 명령을 전달할 컨테이너 이름

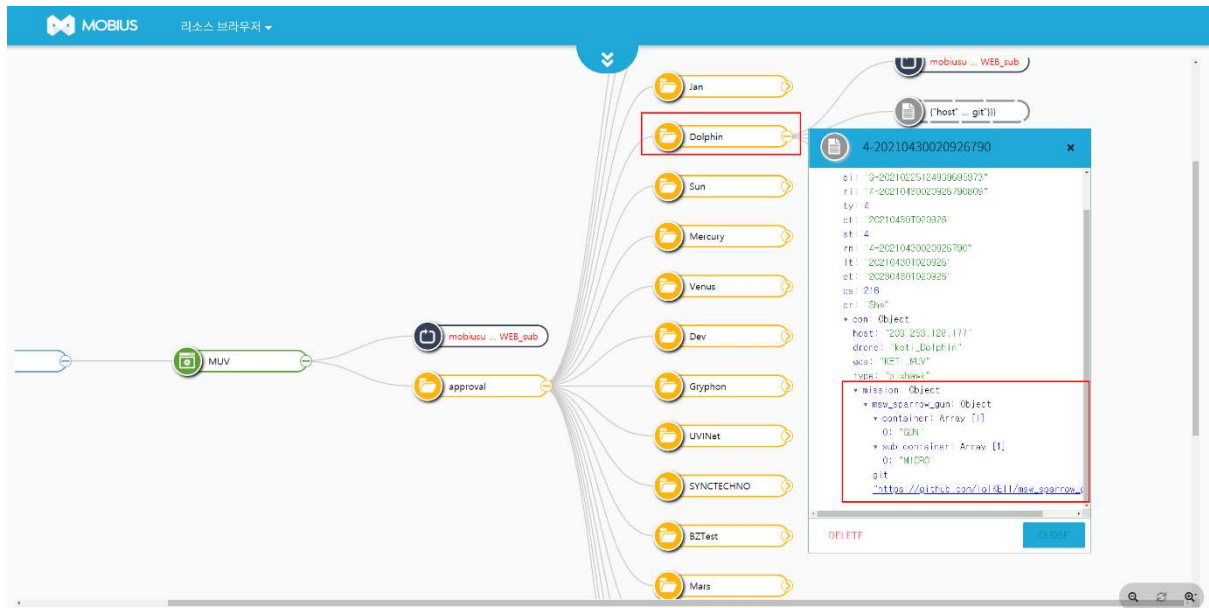
git: https://github.com/loTKETI/msw - Github에 배포된 임무SW 저장소 주소

⑦ MC의 nCube-MUV 폴더에서 flight.json 정보 수정 (nCube-MUV 가이드 참고)

- ✓ flight에 ⑥의 과정에서 업로드한 명세의 드론 ID를 추가한다

```
{} flight.json U X
{} flight.json > ...
1  {
2      "approval_gcs": "MUV",
3      "flight": "Dolphin"
4  }
```


⑧ 명세 업로드 확인



⑨ 투하장치 좌표값 (제어 명령) 전송

- ✓ /Mobius/{GCS 이름}/Mission_Data/{드론 이름}/msw_sparrow_gun/MICRO 컨테이너 아래에 Content Instance를 생성하여 con값에 센서가 설치된 좌표값을 입력한다.
- ✓ 정상적으로 작동한다면 투하장치 제어보드에서 좌표값을 전달받아 솔레노이드에 신호를 전송하고, “딸깍”하는 소리와 함께 센서가 투하된다.

