

# 임무 소프트웨어 Guide

LTE 조종기

msw\_lte\_rc

# 목 차

1	개요 .....	3
2	하드웨어 .....	4
2.1	TTL to SBUS/PWM Module .....	4
2.2	하드웨어 연결 .....	4
3	임무Lib 및 임무SW 개발 .....	5
3.1	임무Lib, 임무SW와 개발도구 .....	5
3.2	임무Lib 개발 .....	5
3.3	임무SW 개발 .....	11

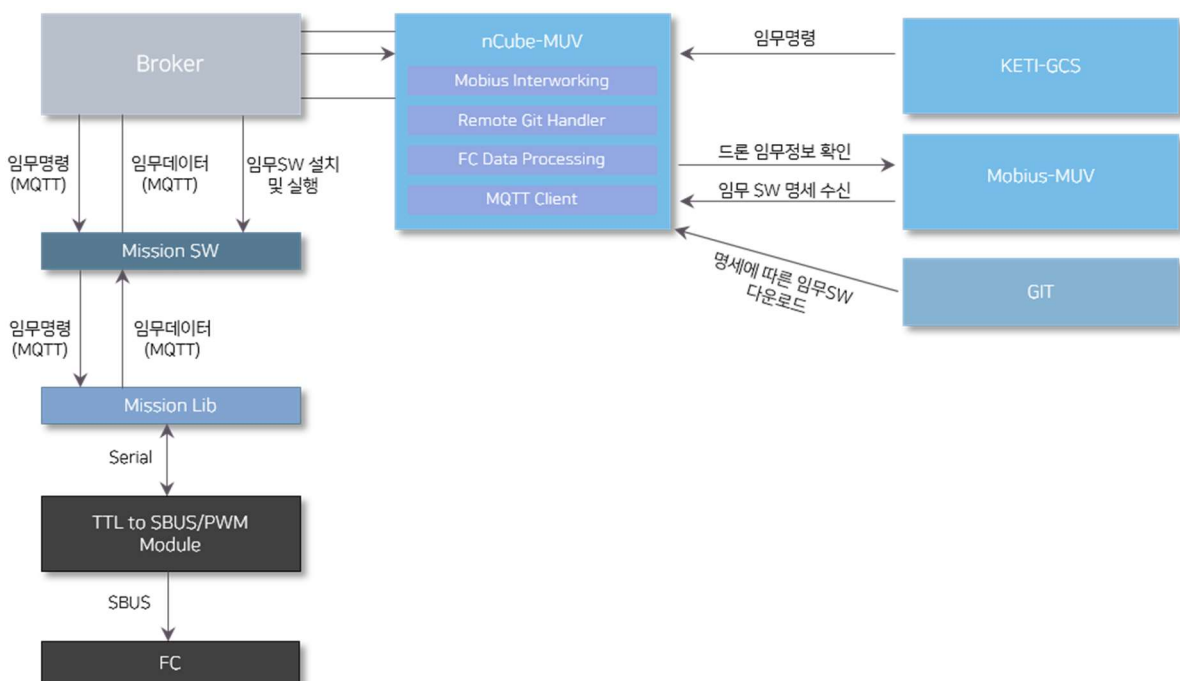
## 1 개요

이 문서는 기존 Futaba, Graupner 등의 조종기를 대체하여 드론 제어가 가능한 LTE 기반 조종기 SW와 연동하기 위한 임무 SW를 구현하는 방법을 설명하는 문서이다. 여기서 설명하는 임무 SW는 LTE 조종기 SW에서 전달된 데이터를 SBUS 프로토콜 신호로 변환하여 FC에 전달하는 역할을 한다.

MC는 KETI에서 Raspberry Pi 3 Compute Module이 탑재된 자체 제작한 보드를 사용하였고, SBUS 프로토콜을 짐벌에 전송하기 위해 TTL to SBUS Module을 사용하였다.

우선, 임무 SW는 PWM 프로토콜인 FC의 조종기 파라미터 값(Min, Max, Trim)을 읽어온다. 이후 Mobius에 업로드 된 LTE 조종기의 값을 전달받는다. 전달받은 LTE 조종기 값은 FC 조종기 파라미터 값에 맞게 정규화하는 과정을 거쳐 SBUS 신호와 유사하게 파싱하여 패킷을 만들고 Serial 통신을 통해 TTL to SBUS 모듈에 명령을 전송한다. TTL to SBUS 모듈은 TTL 신호를 SBUS 프로토콜 신호로 변환하여 FC에 전송한다.

이를 위해 TTL to SBUS 모듈에 대해 간략히 설명하고 TTL to SBUS 모듈과 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.

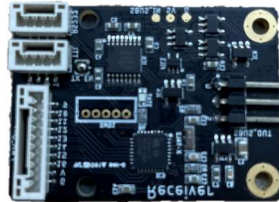


< LTE 조종기 임무 소프트웨어(msw\_lte\_rc) 아키텍처 >

## 2 하드웨어

### 2.1 TTL to SBUS/PWM Module

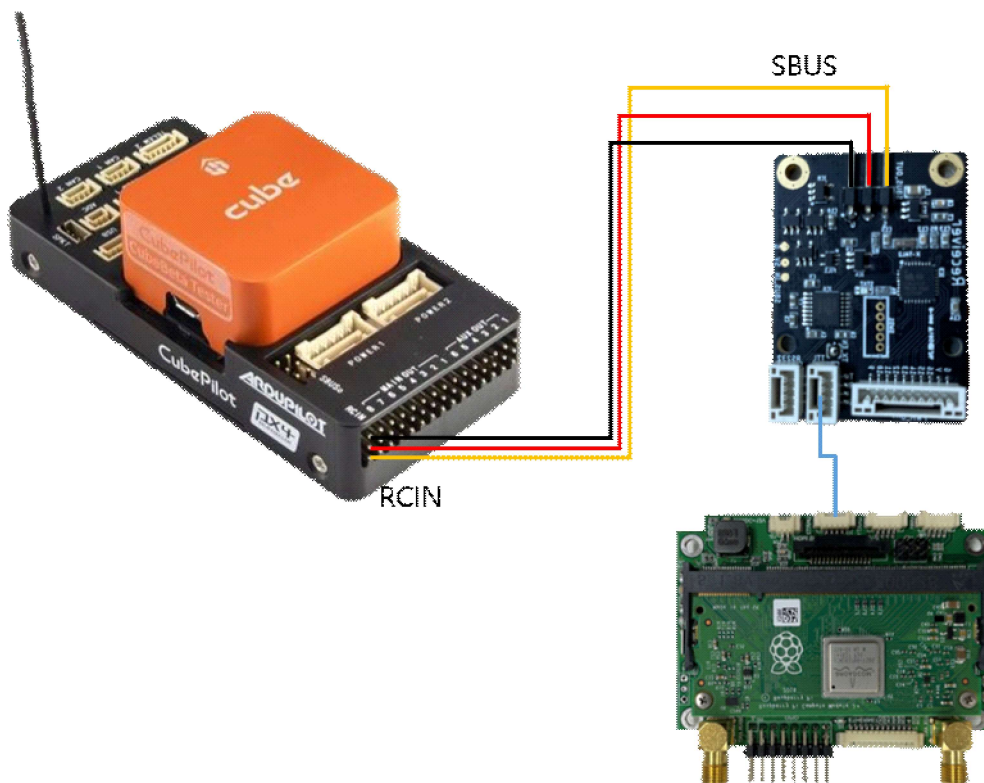
TTL이나 RS232로 들어오는 신호를 SBUS나 PWM으로 변환하는 모듈이다.



<TTL to SBUS/PWM Module>

### 2.2 하드웨어 연결

- 보드의 Serial(/dev/ttyUSB3) 포트와 TTL to SBUS 모듈의 TTL 포트와 연결한다
- TTL to SBUS 모듈의 SBUS 포트는 FC의 RCIN 포트에 연결한다.



<FC 제어를 위한 HW 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 보다 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

짐벌/카메라를 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 다음과 같이 개발한다.

① MC와 내부적으로 데이터 통신을 위한 MQTT 통신 예시코드

```
let lib_mqtt_client = null;

lib_mqtt_connect('localhost', 1883);

let control_topic = '/MUV/control/' + lib.name + '/' + lib.control[0]
let data_topic = '/MUV/data/' + lib.name + '/' + lib.data[0]

function lib_mqtt_connect(broker_ip, port) {
  if (lib_mqtt_client == null) {
    var connectOptions = {
      host: broker_ip,
      port: port,
      protocol: "mqtt",
      keepalive: 10,
      protocolId: "MQTT",
      protocolVersion: 4,
      clean: true,
      reconnectPeriod: 2000,
      connectTimeout: 2000,
      rejectUnauthorized: false
    };

    lib_mqtt_client = mqtt.connect(connectOptions);
  }

  lib_mqtt_client.on('connect', function () {
    console.log('[lib_mqtt_connect] connected to ' + broker_ip);
    lib_mqtt_client.subscribe(control_topic);
    console.log('[lib_mqtt_connect] control_topic: ' + control_topic);
  });

  lib_mqtt_client.on('message', function (topic, message) {
    if (topic === control_topic) {
      let obj_lib_data = JSON.parse(message);
      let ch_num = parseInt(obj_lib_data.num);
      let ch_val = parseInt(obj_lib_data.value);

      key_to_signal(ch_num, ch_val);
    }
  });

  lib_mqtt_client.on('error', function (err) {
    console.log(err.message);
  });
}
```

② TTL to SBUS/PWM Module과 직접적으로 연결하기 위한 인터페이스(Serial) 연동 예시 코드

```
sbusPortOpening();

function sbusPortOpening() {
  if (sbusPort == null) {
    sbusPort = new SerialPort(sbusPortNum, {
      baudRate: parseInt(sbusBaudrate, 10),
    });

    sbusPort.on('open', sbusPortOpen);
    sbusPort.on('close', sbusPortClose);
    sbusPort.on('error', sbusPortError);
    sbusPort.on('data', sbusPortData);
  } else {
    if (sbusPort.isOpen) {

    } else {
      sbusPort.open();
    }
  }
}

function sbusPortOpen() {
  console.log('sbusPort open. ' + sbusPortNum + ' Data rate: ' + sbusBaudrate);
}

function sbusPortClose() {
  console.log('sbusPort closed.');
```

```
  setTimeout(sbusPortOpening, 2000);
}

function sbusPortError(error) {
  let error_str = error.toString();
  console.log('[sbusPort error]: ' + error.message);
  if (error_str.substring(0, 14) === "Error: Opening") {

  } else {
    console.log('sbusPort error : ' + error);
  }

  setTimeout(sbusPortOpening, 2000);
}

function sbusPortData(data) {
  //console.log(data.toString());
}
```

### ③ SBUS 데이터를 중앙에 업로드하기 위한 코드

```
function sbusData() {
  let sbus = {};
  sbus.ch1 = ch1;
  sbus.ch2 = ch2;
  sbus.ch3 = ch3;
  sbus.ch4 = ch4;
  sbus.ch5 = ch5;
  sbus.ch6 = ch6;
  sbus.ch7 = ch7;
  sbus.ch8 = ch8;
  sbus.ch9 = ch9;
  sbus.ch10 = ch10;
  sbus.ch11 = ch11;
  sbus.ch12 = ch12;
  sbus.ch13 = ch13;
  sbus.ch14 = ch14;
  sbus.ch15 = ch15;
  sbus.ch16 = ch16;
  sbus.ch17 = ch17;
  lib_mqtt_client.publish(data_topic, JSON.stringify(sbus));
}
```

### ④ 채널에 따라 FC의 조종기 파라미터 값에 따라 정규화 후 매핑

```
function key_to_signal(ch_num, ch_val) {
  try {
    if (ch_num === 1) {
      ch1_target_val = min_max_scaler(ch_val);
      if (ch1_target_val > sbus_module_value[rc_map.rc1_max]) {
        ch1_target_val = sbus_module_value[rc_map.rc1_max];
      } else if (ch1_target_val < sbus_module_value[rc_map.rc1_min]) {
        ch1_target_val = sbus_module_value[rc_map.rc1_min];
      } else {
      }
    } else if (ch_num === 2) { // Pitch
      ch2_target_val = min_max_scaler(ch_val);
      if (ch2_target_val > sbus_module_value[rc_map.rc2_max]) {
        ch2_target_val = sbus_module_value[rc_map.rc2_max];
      } else if (ch2_target_val < sbus_module_value[rc_map.rc2_min]) {
        ch2_target_val = sbus_module_value[rc_map.rc2_min];
      } else {
      }
    } else if (ch_num === 3) { // Throttle
      ch3_target_val = min_max_scaler(ch_val);
      if (ch3_target_val > sbus_module_value[rc_map.rc3_max]) {
        ch3_target_val = sbus_module_value[rc_map.rc3_max];
      } else if (ch3_target_val < sbus_module_value[rc_map.rc3_min]) {
        ch3_target_val = sbus_module_value[rc_map.rc3_min];
      } else {
      }
    } else if (ch_num === 4) { // Yaw
      ch4_target_val = min_max_scaler(ch_val);
      if (ch4_target_val > sbus_module_value[rc_map.rc4_max]) {
        ch4_target_val = sbus_module_value[rc_map.rc4_max];
      } else if (ch4_target_val < sbus_module_value[rc_map.rc4_min]) {
        ch4_target_val = sbus_module_value[rc_map.rc4_min];
      } else {
      }
    } else if (ch_num === 5) { //
      ...
    }
  }
}
```



### ⑤ 채널 값 포함하여 SBUS로 변환하기 위한 신호 변경

```
function channel_val() {
  // console.log('ch1: ', ch1, 'ch2: ', ch2, 'ch3: ', ch3, 'ch4: ', ch4);

  rxbuf = '';
  rxbuf += 'C0';
  rxbuf += 'D0';

  // CH1 - Roll
  ch1 = ch1_target_val;
  hex_ch1 = ch1.toString(16);
  hex_ch1 = hex_ch1.padStart(4, '0');
  let ch1_high_byte = hex_ch1.substr(0, 2);
  let ch1_low_byte = hex_ch1.substr(2, 2);
  rxbuf += ch1_high_byte;
  rxbuf += ch1_low_byte;

  // CH2 - Pitch
  ch2 = ch2_target_val;
  hex_ch2 = ch2.toString(16);
  hex_ch2 = hex_ch2.padStart(4, '0');
  let ch2_high_byte = hex_ch2.substr(0, 2);
  let ch2_low_byte = hex_ch2.substr(2, 2);
  rxbuf += ch2_high_byte;
  rxbuf += ch2_low_byte;

  // CH3 - Throttle
  ch3 = ch3_target_val;
  hex_ch3 = ch3.toString(16);
  hex_ch3 = hex_ch3.padStart(4, '0');
  let ch3_high_byte = hex_ch3.substr(0, 2);
  let ch3_low_byte = hex_ch3.substr(2, 2);
  rxbuf += ch3_high_byte;
  rxbuf += ch3_low_byte;

  // CH4 - Yaw
  ch4 = ch4_target_val;
  hex_ch4 = ch4.toString(16);
  hex_ch4 = hex_ch4.padStart(4, '0');
  let ch4_high_byte = hex_ch4.substr(0, 2);
  let ch4_low_byte = hex_ch4.substr(2, 2);
  rxbuf += ch4_high_byte;
  rxbuf += ch4_low_byte;
```

•  
•  
•

```
hex_ch17 = ch17.toString(16);
hex_ch17 = hex_ch17.padStart(4, '0');
let ch17_high_byte = hex_ch17.substr(0, 2);
let ch17_low_byte = hex_ch17.substr(2, 2);
rxbuf += ch17_high_byte;
rxbuf += ch17_low_byte;
checksum_extra();
// rxbuf += '00';
// console.log(rxbuf);
// console.log(Buffer.from(rxbuf, 'hex'));
sbusPort.write(Buffer.from(rxbuf, 'hex'));
sbusData();
}
```

## ⑥ checksum 생성

```
function checksum_extra() {
  var crc_res = 0;
  for (let i = 0; i < 34; i++) {
    crc_res += parseInt(rxbuf.substr(4 + (i * 2), 2), 16);
    crc_res = crc_res & 0x00ff;
  }

  crc_res = crc_res & 0x00ff;

  crc_res = crc_res.toString(16);
  hex_crc_res = crc_res.toString(16);
  hex_crc_res = hex_crc_res.padEnd(4, '0');
  let crc_res_byte = hex_crc_res.substr(0, 2);
  let tail_byte = hex_crc_res.substr(2, 2);
  rxbuf += crc_res_byte;
  rxbuf += tail_byte;
}
```

- 기존 임무Lib와 달리 Node.js 버전의 임무Lib는 임무SW에서 require 메소드를 통해 외부 모듈로 불러와 연동한다.
- Node.js로 개발되어 응용프로그램 형태로 배포된 LTE 조종기 임무Lib는 아래 Github 주소에서 확인할 수 있다.

[https://github.com/loTKETI/lib\\_lte\\_rc.git](https://github.com/loTKETI/lib_lte_rc.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.

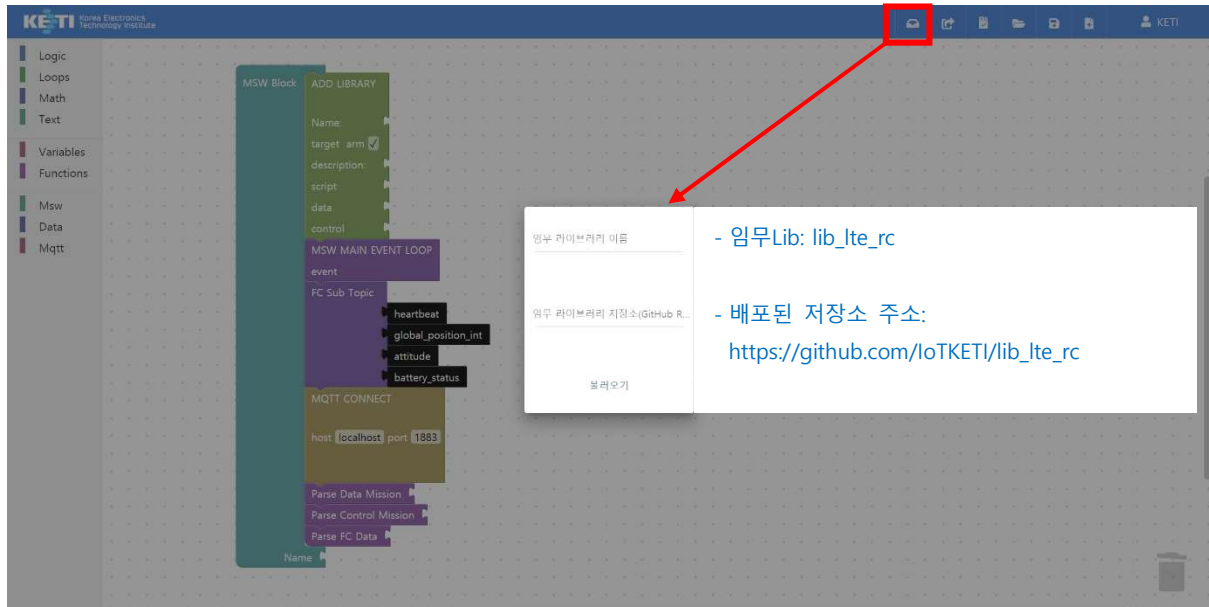
#### 3.3.2 임무 S/W 개발과정

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



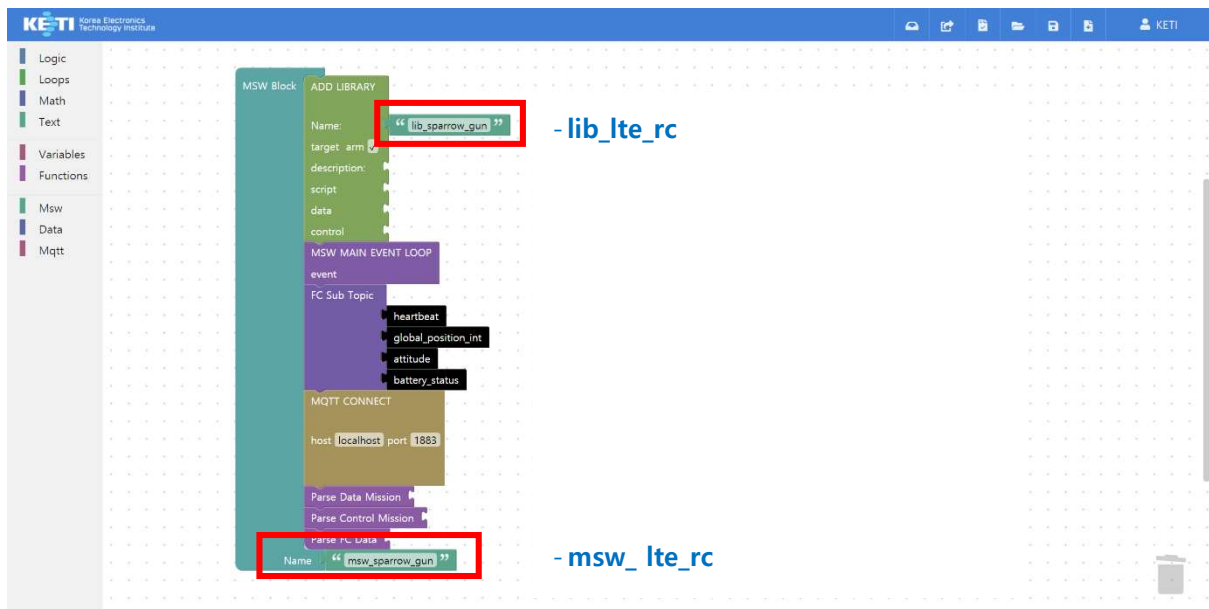
② 사전에 개발해서 배포한 임무Lib를 호출한다.

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

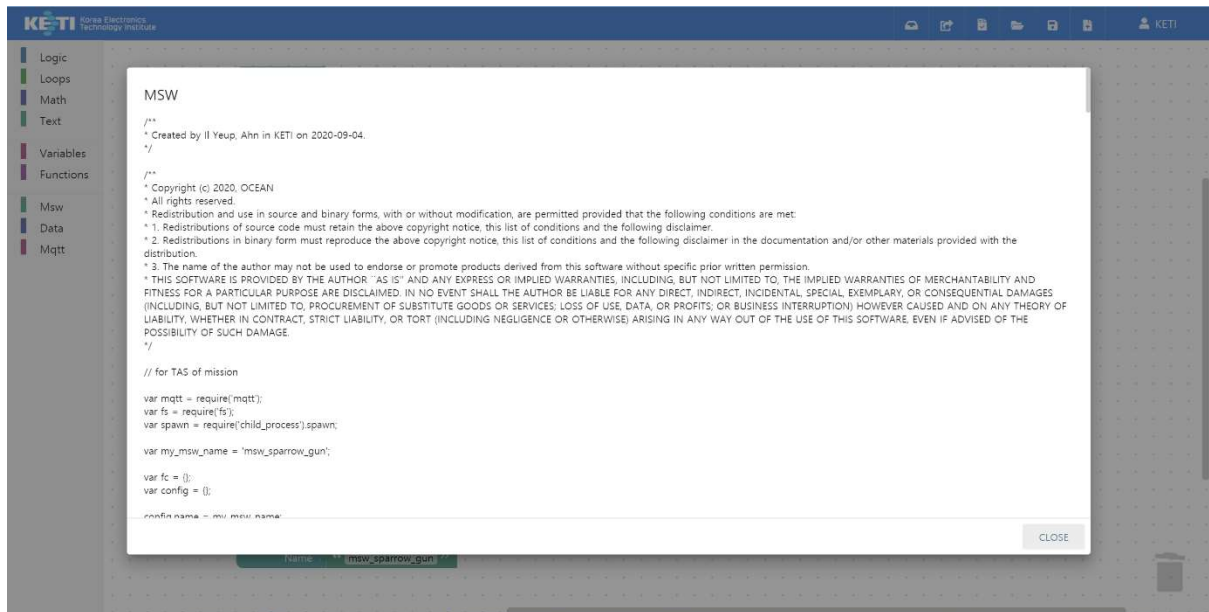


③ 생성된 임무Lib 블록 추가 및 임무SW의 이름 입력

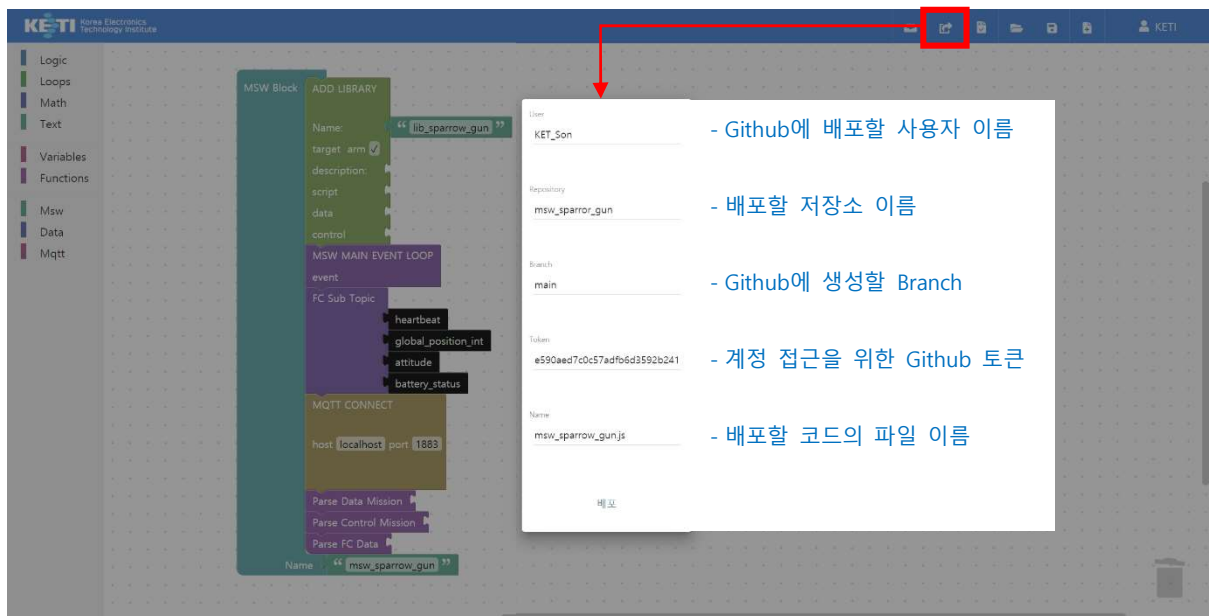
- ✓ lib\_sparrow\_gun 대신 lib\_lte\_rc



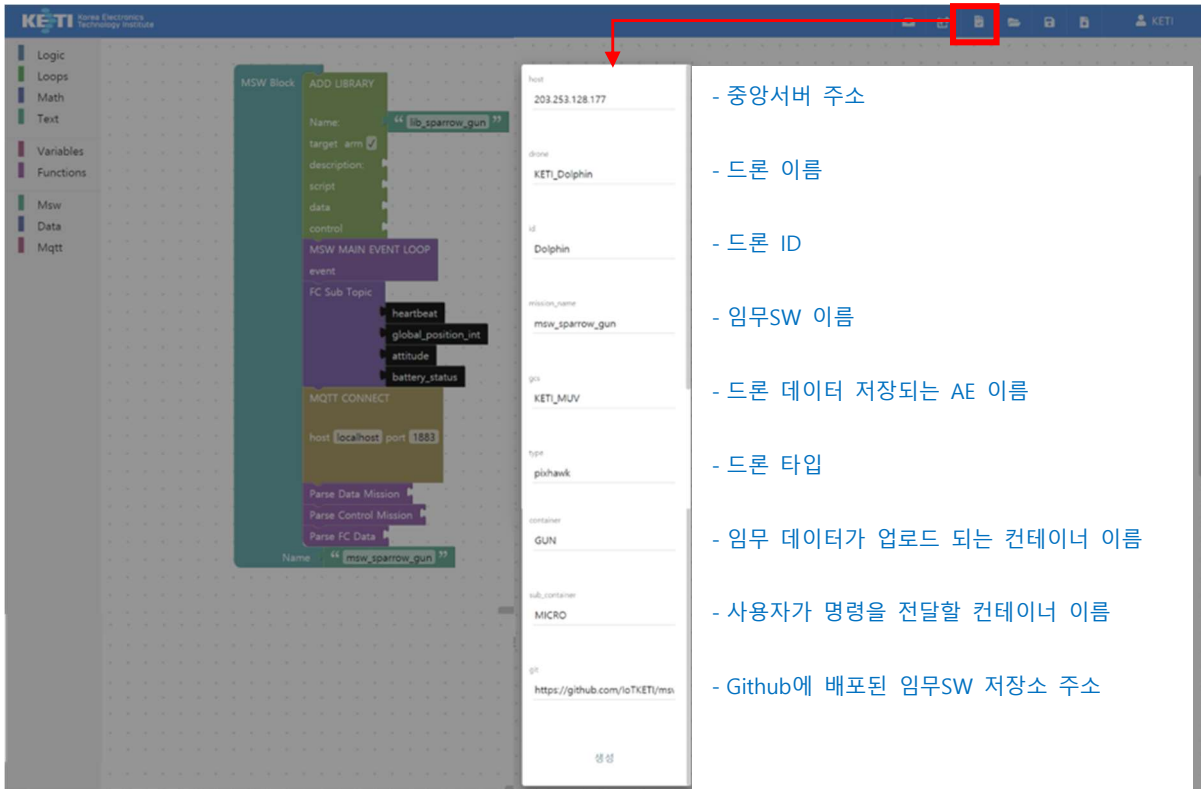
④ 블록으로 코딩한 임무SW를 배포하기 전 소스코드 형태로 확인



⑤ 임무SW 배포에 필요한 정보 입력하여 배포



⑥ 드론 탑재를 위한 드론 명세 작성



- 중앙서버 주소
- 드론 이름
- 드론 ID
- 임무SW 이름
- 드론 데이터 저장되는 AE 이름
- 드론 타입
- 임무 데이터가 업로드 되는 컨테이너 이름
- 사용자가 명령을 전달할 컨테이너 이름
- Github에 배포된 임무SW 저장소 주소

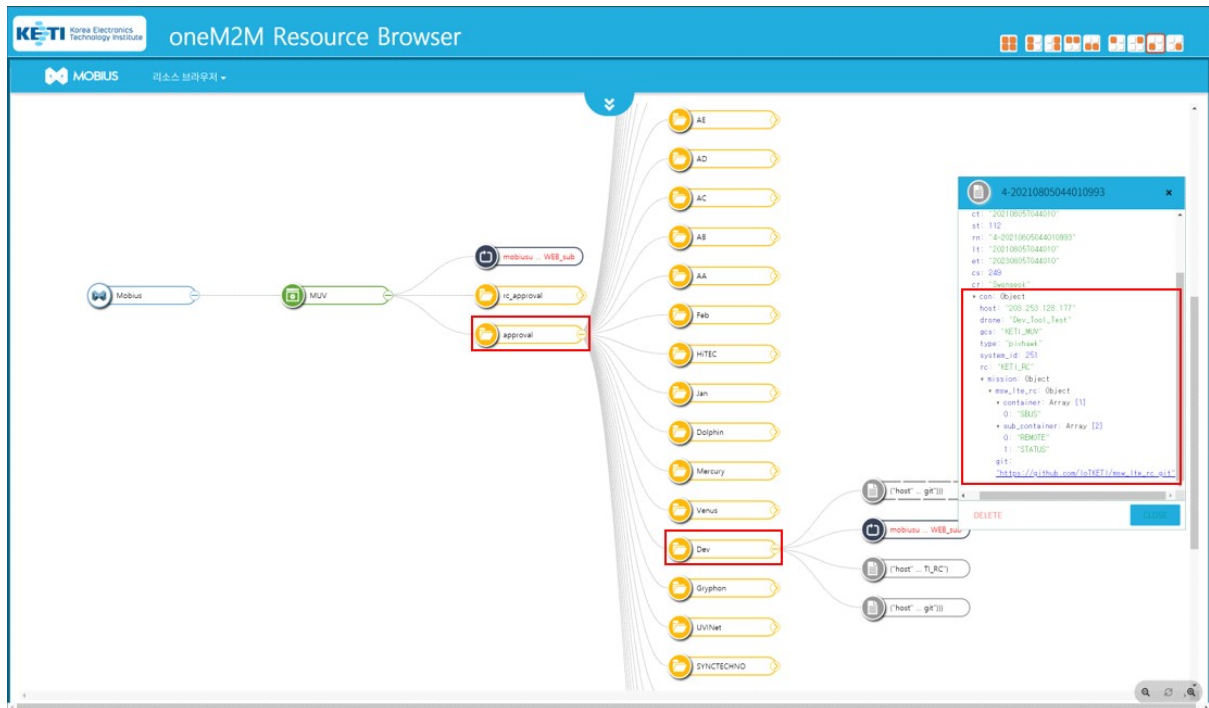
⑦ MC의 nCube-MUV 폴더에서 flight.json 정보 수정 (nCube-MUV 가이드 참고)

- ✓ flight에 ⑥의 과정에서 업로드한 명세의 드론 ID를 추가한다

```

1  {
2      "approval_gcs": "MUV",
3      "flight": "Dev"
4  }
    
```

⑧ 명세 업로드 확인



⑨ LTE 조종기 값 전달받기 위한 명령 (제어 명령) 전송

- ✓ /Mobius/{GCS 이름}/Mission\_Data/{드론 이름}/msw\_lte\_rc/**STATUS** 컨테이너 아래에 Content Instance를 생성하여 con값에 "ON"을 입력한다. (OFF 시 데이터 받는 것을 중단함)

