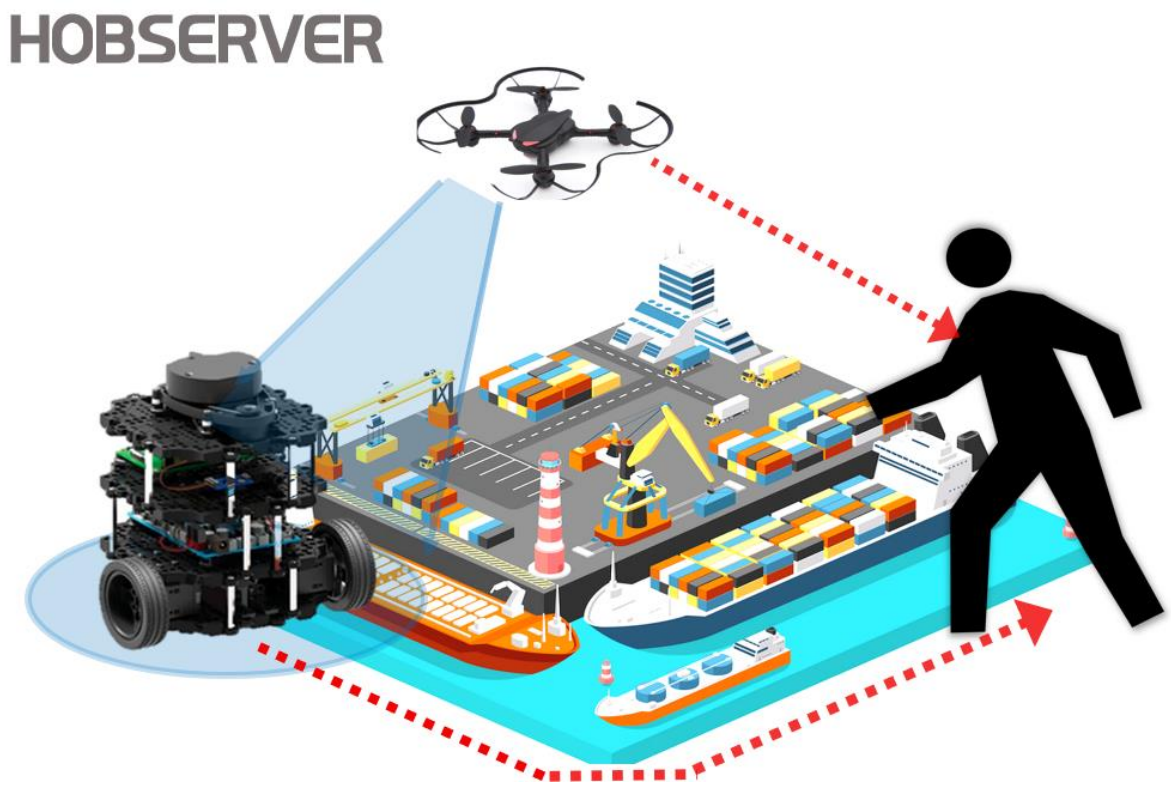
제18회 임베디드SW경진대회 개발완료보고서

[Smart Things]

□ 개발 요약

팀 명	KOBOT보드
	
작품명	Hobserver (홉저버)
작품설명 (요약)	· 드론과 자율주행 로봇을 이용한 항만감시 시스템이다. · 드론을 이용해 항만 전체를 감시하고, 저장되지 않은 사람이 감지되면 자율주행 로봇을 보내 해당 구역을 탐색하도록 한다. · 자율주행 로봇은 지정된 장소로 도달해 목표한 사람을 발견하면 OTP 인증을 요청해 인증 절차를 거치도록 한다.
소스코드	https://github.com/Johyeyoung/2020ESWContest_Smart-Things_5057
시연동영상	https://www.youtube.com/watch?v=n9Al9gaUGHQ&feature=youtu.be

□ 개발 개요

개발 작품 개요

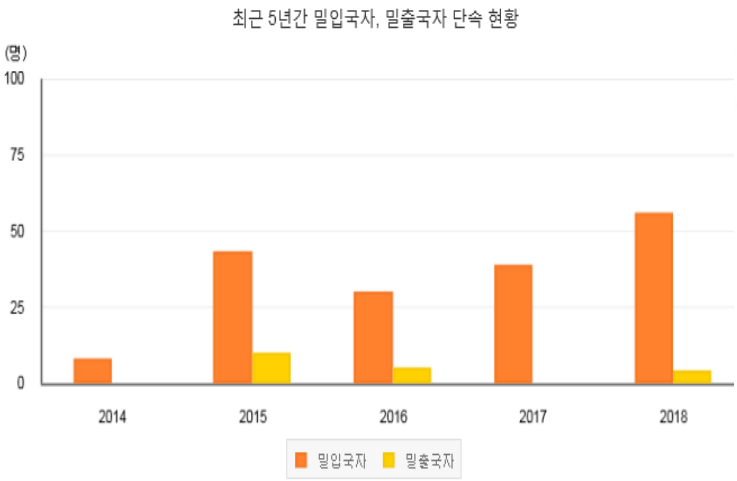


그림1. 최근 5년간 밀입국자, 밀출국자 단속 현황[1]

그림2. 최근 7월 항만 밀입국 사건[2]

[그림 1]은 2018 년도까지 항만에서의 불법 밀입국은 증가하는 추세를 보여준다.
[그림 2]는 올해 7 월까지 항만에서의 불법 밀입국 사건들이 빈번하게 발생하고 있다는 것을 보여준다. 이를 통해 밀입국과 관련된 문제가 여전히 해결되지 않은 채로 남아있다는 것을 알 수 있다.

그림 3. 소형 보트서 흔적을 찾는 해경

[그림 3]은 2020 년 5 월경 의향리 해변에서 발견한 소형보트를 감식하고 있는 해경의 사진이다. 중국에서 소형보트를 타고 밀입국을 시도하였고 중국인 6 명 중 단 1 명이 붙잡혔다.
이처럼 코로나 19 사태로 인해 느슨해진 항만 보안 경비 틈을 타 언제든 `범죄를 저지를 수 있는' 상태이다. 특히 울산항은 국가 보안시설이 밀집한 울산항 일부 부두가 일반인도 마음만 먹으면 뚫을 수 있는 보안 사각지대로 방치되고 있다. 또한 울산항만공사, 항만 업계 종사자 등에 따르면 지난 4 월 부터 남구 장생포 소형선 부두에 오후 11 시 부터 오전 5 시까지 항만 경비업무를 하지 않고 있다, 이 시간대에 해안을 경비하거나 선박 출입을 통제 하는 인원이 없는것이다. 이에 따라 울산항만공사는 드론을 띄워 CCTV 가 미치지 못하는 사각지대를 감시하는 계획을 이야기했다. 이러한 상황은 보안 인력과 장비등을 보완하고 보안 시스템을 치밀하게 점검하는것의 필요성을 나타내고 있다.
위와 같은 상황을 인지한 KOBOT 팀에서는 운영비 감소와 보안성을 높이기 위해 사람이 살펴보기 어려운 컨테이너 박스 사이, CCTV 가 명확하게 인식 불가능한 구역 등을 드론을 이용해 공간을 조망하여 맵을 생성하고 외부인이나 알 수 없는 객체를 사람이 직접 접근하지 않고 로봇이 객체 인식을 통해 이동하며 수상한 사람을 발견할 경우 인증을 요구하고 불응시 관제센터에 연락하는 등의 방식을 이용하여 신원을 확인하는 '홉저버' 경비 시스템을 개발하였다.

개발 목표

① 드론과 자율주행 로봇을 이용하여 항만 경비 시스템 구축

: 항만과 같이 넓은 공간을 사람이 경비를 서는 것은 범죄를 빠르게 잡기 쉽지 않고 인력 낭비가 될 수 있다. 이를 보완하기 위해 홉저버는 드론을 사용해 1 차적으로 하늘에서 카메라로 항만을 감시하고 침입자나 의심이 가는 객체를 발견했을 경우 서버로 객체의 위치와 사진을 전송한다. 그리고 자율주행로봇인 터틀봇을 객체의 위치로 보내 이 객체가 침입자(밀입국자)인지 유효한 인물인지 OTP 인증 통과 유무를 통해 구분하는 것을 목표로 한다.

② 드론과 터틀봇의 객체 인식

: 먼저 YOLO 를 이용하여 항만에서 유효한(인증받은) 사람을 학습한다. 차후에 드론이 항만을 탐색할 때 발견된 객체가 유효한지(인증받은 사람인지) YOLO 로 객체를 인식하는 것을 목표로 한다.

③ 드론을 이용한 항만 맵 촬영과 YOLO 를 이용한 객체 위치 좌표 반환

: 드론이 하늘에서 촬영한 항만 맵 영상을 바로 활용하기엔 영상의 기울어짐, 노이즈, 컨테이너 구분 등의 한계가 있다. 이러한 한계를 보완하기 위해 홉저버는 OpenCV 라이브러리를 활용한 자체 영상처리 알고리즘을 이용한다.

: 드론 pc 에서는 이 영상처리 알고리즘을 이용해 촬영한 영상의 휘어짐과 기울어짐을 보정한다. 그리고 YOLO 를 이용해 객체를 인식한다.

: 드론에서의 객체 인식 후 객체의 실제 위치 좌표를 구하기 위해 영상에서의 픽셀이 실제 공간에서 차지하는 길이를 변환한 후 실제 위치 좌표를 서버로 전송한다.

④ 최종 맵을 구하기 위한 영상처리와 터틀봇 자율주행을 위한 최단 거리 알고리즘

: 서버는 홉저버의 자체 영상처리 알고리즘을 이용해 드론이 촬영한 영상을 보정하고 컨테이너 구분을 처리한다. 그리고 터틀봇이 자율주행 가능한 영역과 구분과 그러지 못한 영역을 구분하여 최종 2 차원 맵을 구한다.

:드론으로부터 받은 객체 위치까지의 거리를 터틀봇이 빠르게 갈 수 있도록 최단 거리
알고리즘을 적용시켜 경로를 구하는 것을 목표로 한다.

⑤ 보안 인증을 위한 OTP 제작

: 객체로부터 인증을 받기 위해 OTP 를 제작을 하고 OTP 발생 알고리즘을 구현한다.

OTP 발생 알고리즘은 선형합동법 계산식을 이용한다. 선형합동법은 이전 OTP 를 다음 OTP 의 seed 값으로 사용하기 때문에 매일 같은 시간에 같은 OTP 가 생성되는 문제가 없다. 선형합동법 계산식은 다음과 같다.

(a = 사용자 비밀번호, c = 고유번호, m = 현재시간)

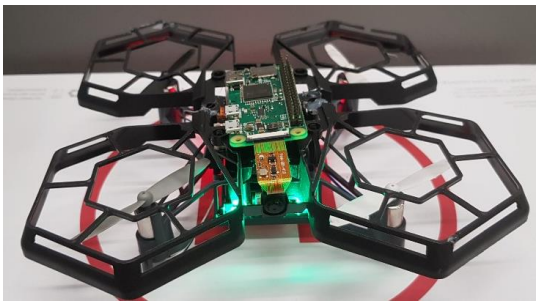
서버도 같은 방식으로 OTP 를 생성하여 클라이언트가 전송한 값의 유효성을 검사한다.

□ 개발 환경 설명

1. Hardware 구성

1-1. 하드웨어의 전체적인 구성

본 프로젝트 Hobserver에서 하드웨어는 자율주행 감시로봇(TurtleBot3), 항만Observer(CoDroneII), OTP 인증모듈(ATmega128)로 구성되어 있다. 하드웨어의 전체적인 구성은 다음과 같다



▲ 1) 항만 Observer(CoDrone II)

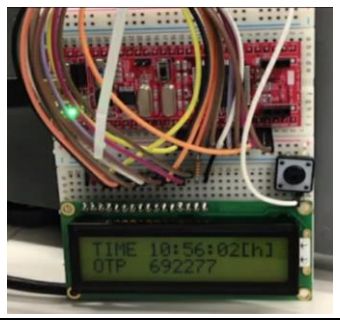
: 구역마다 항만 위를 비행하며 항만을 감시하다가 객체를 탐지하면 서버로 객체의 좌표와 항만의 현장 사진을 보낸다.

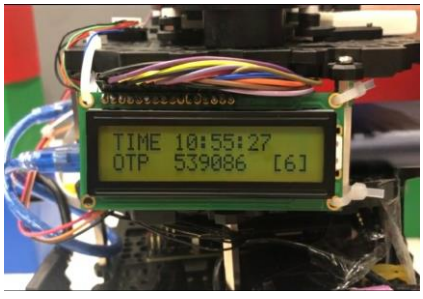
◀ 2) 자율주행 감시로봇(TurtleBot3)

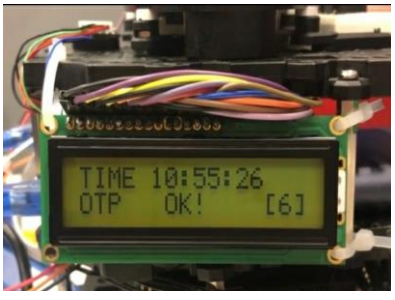
: 항만의 지상에서 대기모드를 유지하고 있다가 서버로부터 객체의 위치 경로를 받으면 해당 경로에 맞춰 주행해 객체를 찾고 객체를 발견하면 OTP 인증을 요청한다.

3) OTP 모듈(ATmega128)

: 자율주행 감시 로봇에 부착되어 있으며 서버로부터 OTP 인증 요청 신호를 받으면 OTP 인증을 실행한다.







1-2. 하드웨어의 세부적인 구성

			
그림 4. single board computer (Jetson nano)	그림 5. Lidar 센서(LDS-01)	그림 6. 코드론2 pro	그림 7. ATmega128
			
그림 8. Wifi Module Intel 8625AC M.2	그림 9. LC1621-SMLYH6-DH3	그림 10. Raspberry Pi Zero W	그림 11. TurtleBot3

표 1. 하드웨어 개발환경

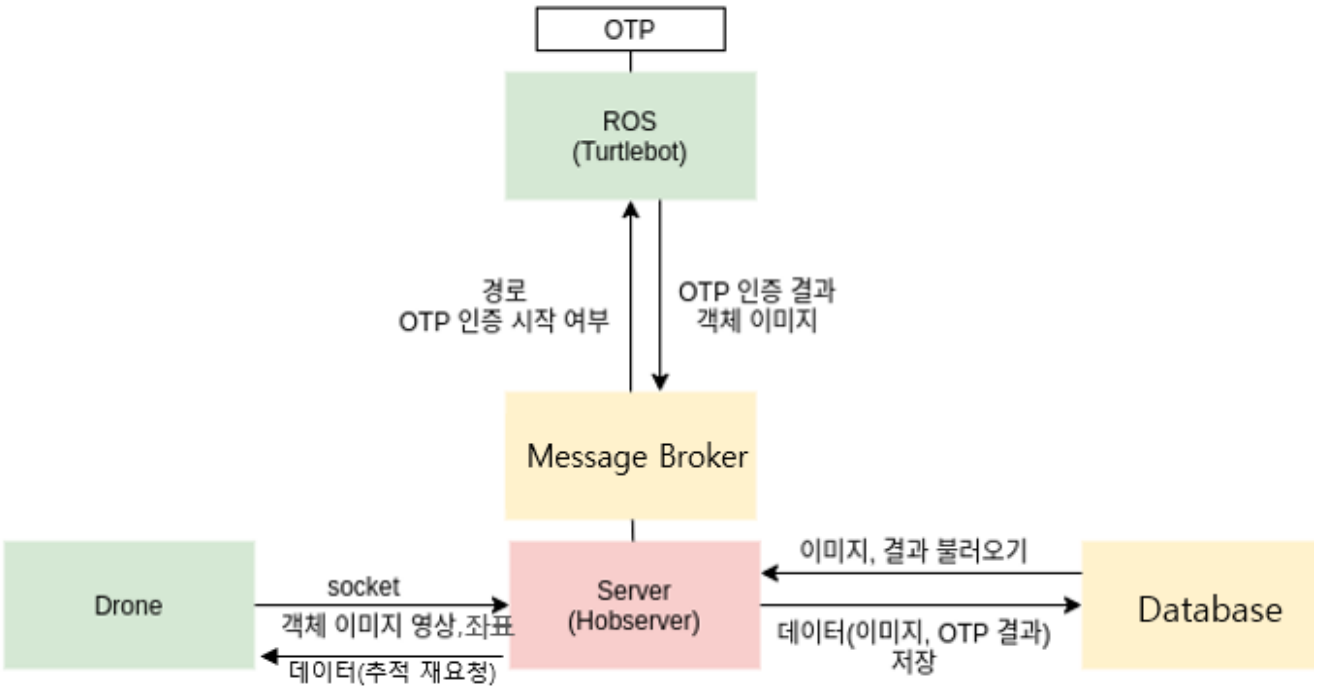
제품명	주요정보	
CoDrone2 Pro	size	180x190x65mm
	weight	150g
	Max Flight Time	7min
TurtleBot3	Embedded Controller	OpenCR(32-bit ARM® Cortex®-M7)
ATmega128	MCU	ATmega128A-AU,
	voltage	3.3V~5V
Wifi Module (8625AC)	Wi-DI CERTIFIED	802.11ac
	speed	867Mbps
Single Board Computer (Jetson nano)	CPU	Quad-core ARM A57
	GPU	128-core Maxwell™ GPU
	Memory	4 GB 64-bit LPDDR4
	Operating System	Boots from Micro SD, Linux operating system
Lidar 센서 (LSD-01)	Distance Range	120 ~ 3,500mm
	Sampling Rate	1.8kHz
	Operating supply voltage	5V DC +-5%
	Light source	Semiconductor Laser Diode(785nm)
	Scan Rate	300 +-10 rpm
	Angular Range	360°
	Angular Resolution	1°
Raspberry Pi Zero W	CPU	1GHz single-core CPU
	RAM	512MB RAM
	Camera connector	CSI camera connector
LC1621-SMLYH6-DH3	Display Format	16 Characters X 2 Line
	Outline Dimension	80.0(W) X 36.0(H) X 14.0(T)mm
	Weight	36g

표 2. 하드웨어 개발환경 상세

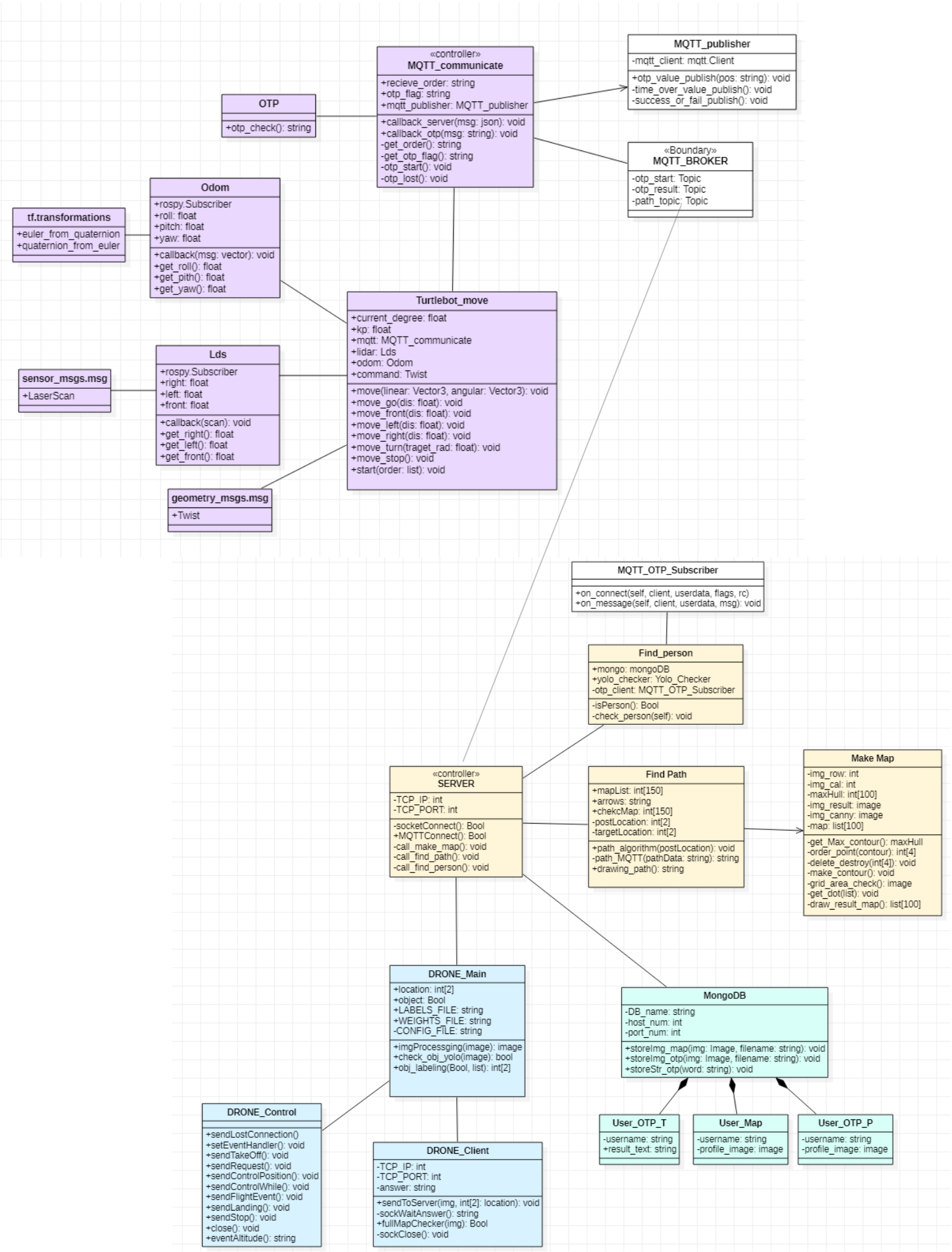
2. Software 구성

2-1 소프트웨어 구성 흐름도

2-2 소프트웨어 구성 데이터 흐름도

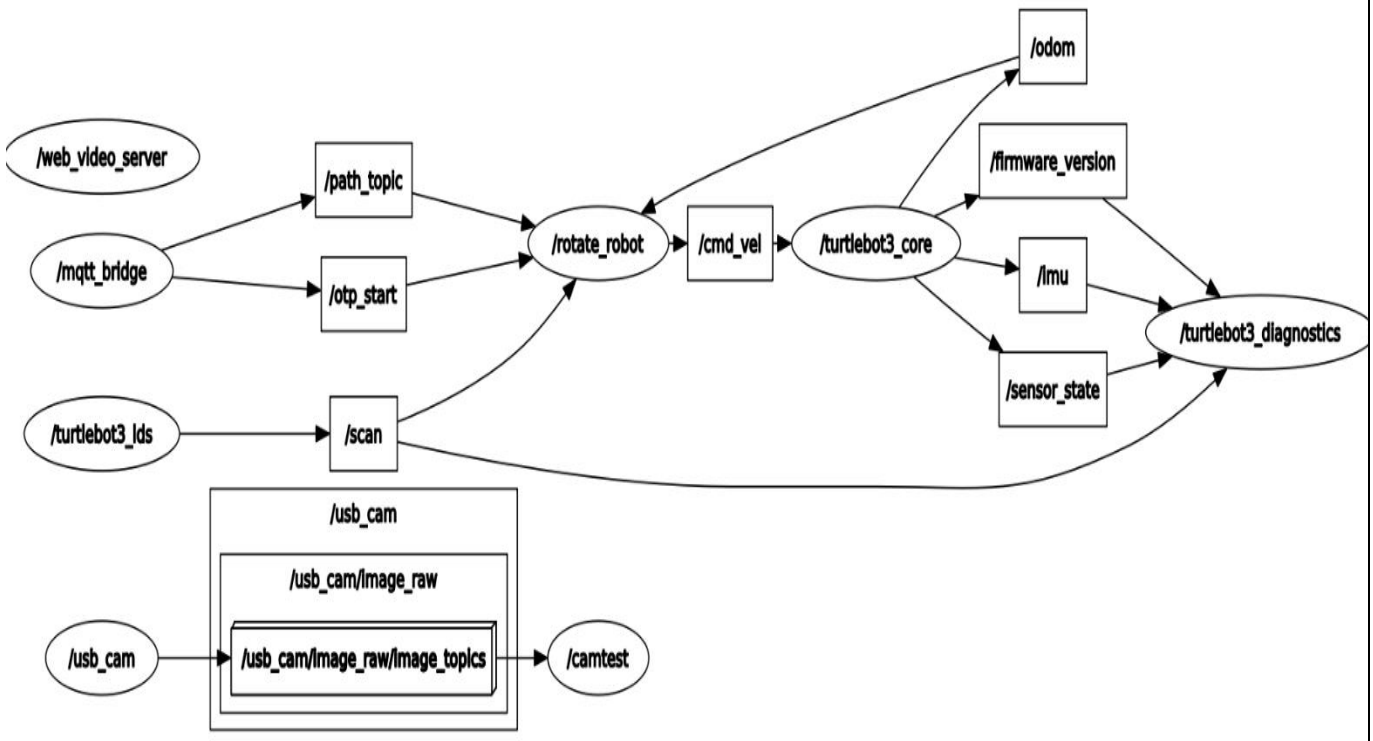


3. Software 설계도



4. Software 기능

1. ROS 주요 Topic과 Node 설명



< ROS rqt_graph>

주요 Topic 과 Node의 소개

1. /mqtt_bridge node:
   서버로부터 전송되는 메시지를 담당하며 이를 통해 받는 메시지를 통해 터틀봇을 원하는 경로로 움직이거나 OTP인증을 시도할 수 있다.

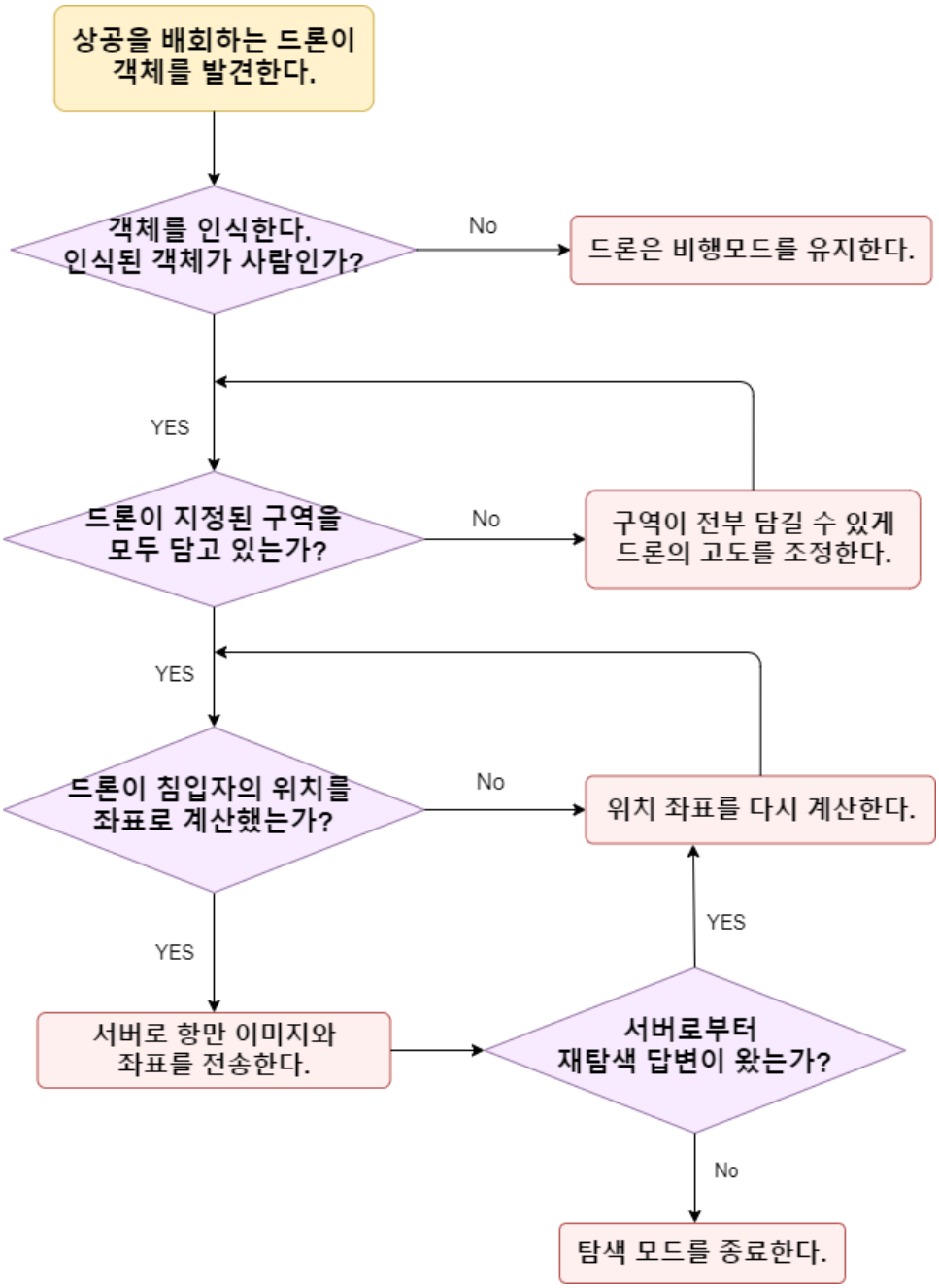
2. /turtlebot3_lds node:
   터틀봇에 부착된 Lidar센서값을 읽어들여 터틀봇이 주행할 때 장애물을 회피 할 수 있다.

3. /usb_cam node:
   터틀봇에 부착된 카메라와 연결되며 이를 이용하여 서버에게 터틀봇 정면의 주행 영상을 보낼 수 있다.

2. 주요 알고리즘

1-1) 드론의 객체 인식 및 추적 알고리즘

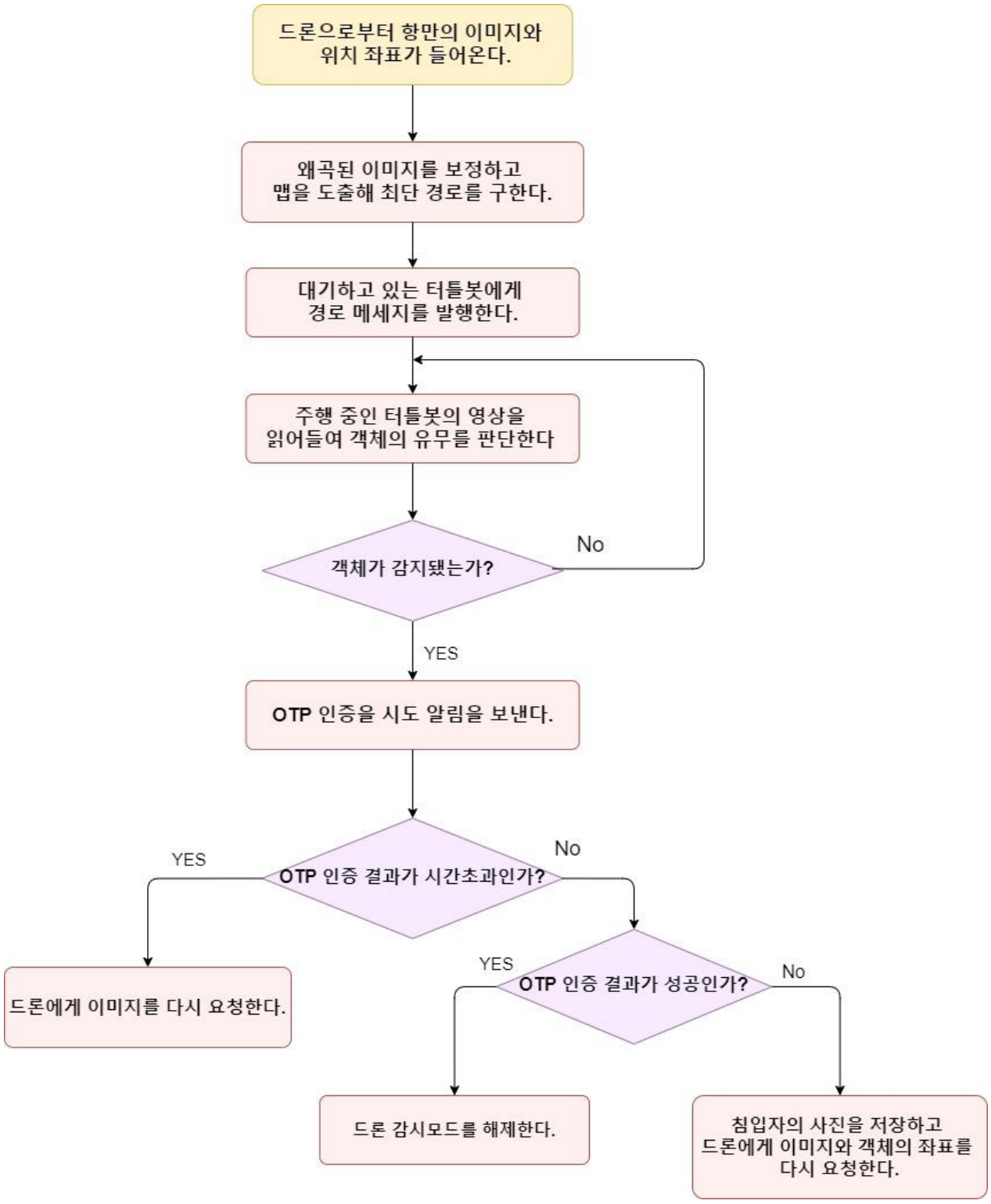
: 항만을 감시 중인 드론이 객체를 발견한 뒤 서버와 통신하는 과정을 담은 순서도이다.
드론은 객체를 발견한 뒤 카메라에 포착된 이미지를 바로 전송하는 것이 아닌 객체의 위치 데이터를 추출하고 담당하고 있는 구역 전체를 담기 위해 여러가지 이미지 operation과정을 거친다. 이러한 과정을 거쳐 적절한 이미지라고 판단되면 서버로 데이터를 전송한다.
데이터를 전송한 드론은 서버의 답변이 오기 전까지는 객체를 추적한다.
추적 모드의 드론에게 서버로부터 객체를 이미지와 관련해 재탐색을 요청하면 앞선 과정을 되풀이 하고 종료하라는 응답이 오면 다시 감시 모드로 돌아가 항만을 비행한다.



1-2) 서버의 항만 맵 도출 및 경로 데이터 생성 알고리즘

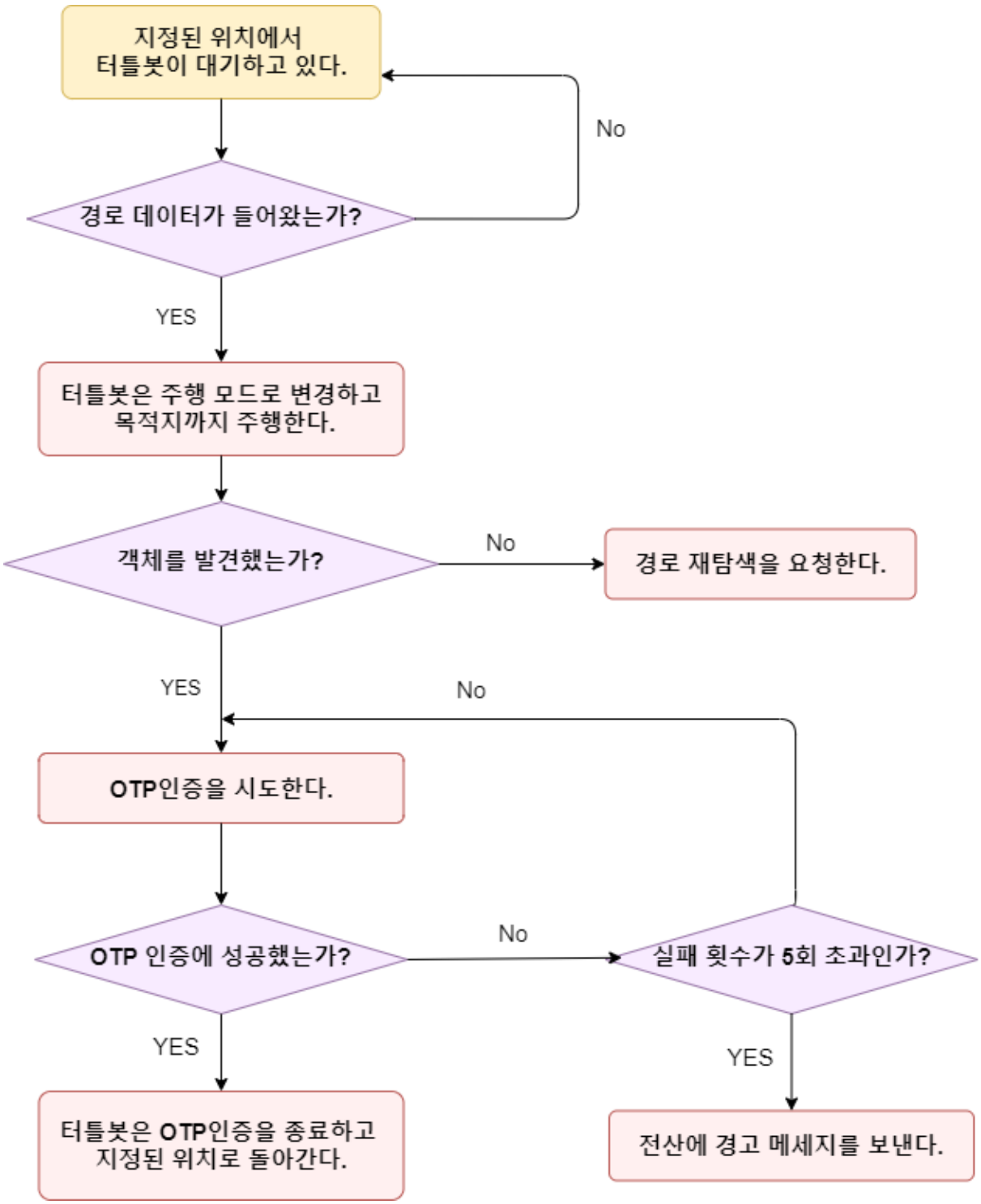
: 서버가 드론으로부터 이미지 데이터를 받아 맵을 도출하고 목적 객체까지의 최단 경로를 도출하여 터틀봇에게 그 결과를 통신하는 결과를 담은 순서도이다.

서버는 항시 통신 대기모드를 유지하고 드론으로부터 데이터가 들어오면 다음과 같은 과정을 통해 객체를 추적하는 연산을 하며 인증 결과를 바탕으로 판단을 내리는 역할을 한다.



1-3) 터틀봇의 객체 추적 및 OTP인증 알고리즘

: 대기 중이 터틀봇이 객체를 추적하고 OTP인증까지 시도하는 결과를 나타내는 순서도이다. 터틀봇은 받은 경로대로 이동하면서 목적지까지 도달한다. 이때 터틀봇이 자체적으로 객체의 판단 유무를 처리하지 않고 서버에게 객체 발견 판단을 위임하여 OTP 인증 시도와 그 결과 값을 주고 받으며 서버와 계속 통신한다.



3. 주요 연산 및 세부 로직

1-1) 맵 도출 알고리즘

1) 드론의 담당 지정 영역 도출하기

① get_Max_contour()

: 가장 큰 등고선을 찾기 위해 전체 이미지에 cv2.findContours() 처리 후 각각의 등고선을 이루는 원소의 개수를 기준으로 오름차순 정렬을 한다. 그 다음, 뒤로 갈수록 등고선을 이루는 원소 수가 많다고 판단하여 맨 끝에 있는 등고선을 최대치 크기의 등고선으로 보아 드론이 담당하는 구역으로 가정한다.

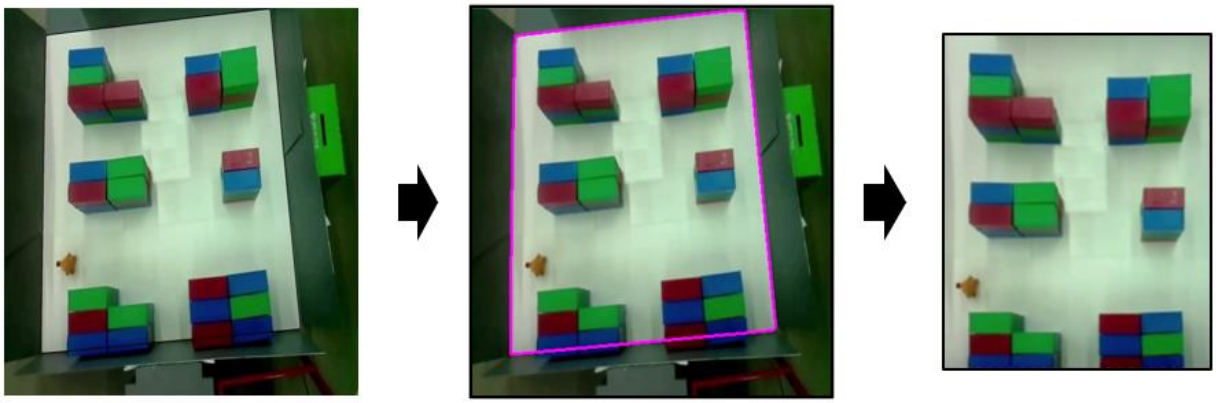
② order_point(contour)

: 영역으로 판단되는 부분을 사각형의 맵으로 도출하기 위해선 휘어짐을 제거해야 하는데 앞서 구한 등고선을 이루는 원소들의 x, y 값을 계산하여 꼭짓점을 도출한다.
x+y의 값이 가장 작은 원소는 원점에 가깝기에 LeftTop, 가장 큰건 원점에서 가장 먼 rightBottom이다, 마찬가지로 x-y의 값을 연산하여 나머지 꼭짓점도 구한다.

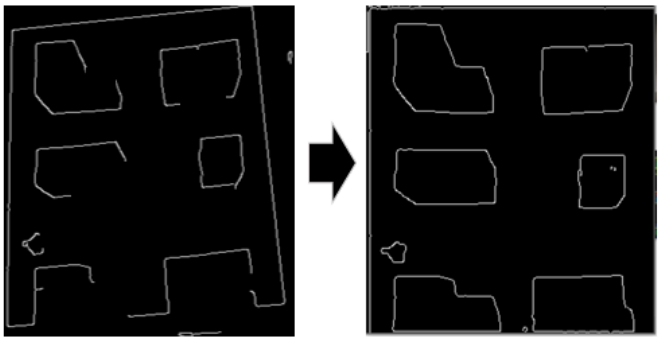
```
LeftTop: [43. 38.], LeftBottom: [ 33. 397.], RightTop: [371.  82.], RightBottom: [385. 358.]
```

③ delete_destroy(int[4])

이제 구한 꼭짓점을 이용하여 가장 넓은 너비와 가장 긴 높이를 구해 cv2.warpPerspective()에 넣어 해당 값을 기준으로 휘어진 이미지를 펼쳐 사각형의 맵을 구한다.



2) 잡음 제거 및 끊어진 경계 이어주기



make_contour()

: 이미지에 잡음이 섞여 있는 경우 컨테이너의 경계가 불분명한 경우가 있다. 이러한 이미지를 엣지검출을 하면 등고선이 끊어져 있어 장애물의 경계가 명확하게 드러나지 않는데 이때 끊어진 등고선을 이어주기 위해 기존의 이미지의 GrayScale값을 이용한다. 전체 픽셀의 GrayScale 값을 검사하여 일정 수준 이하의 value를 지니는 픽셀의 경우 GrayScale의 값을 0으로 주고 나머지는 255로 주어 이분화된 value 값으로 이뤄진 이미지로 변환한다. 그다음 특정 값을 갖는 픽셀의 수를 counting해 그 개수를 행 또는 열을 기준으로 검사 결과를 리스트에 담아둔다.

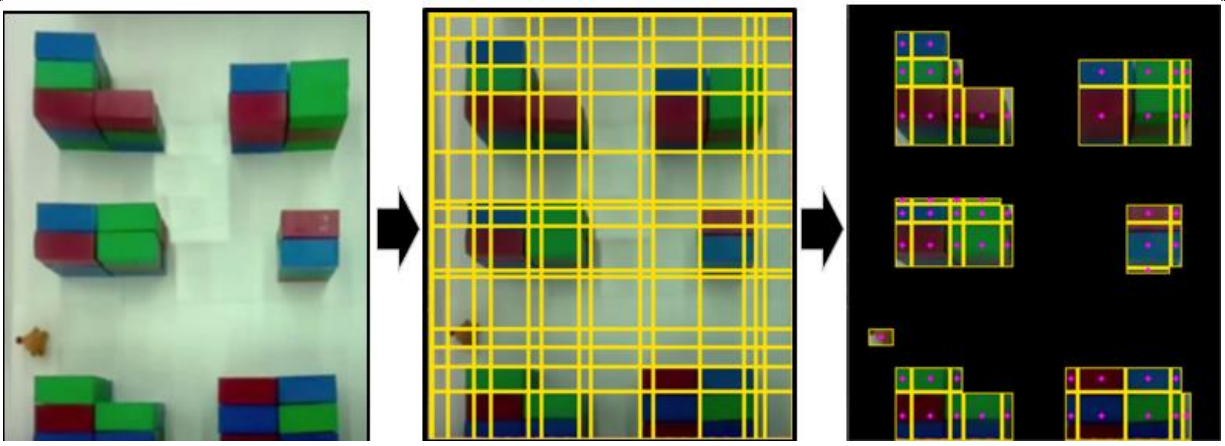
그런 뒤 일정 수준의 값을 지니는 경우는 잡음으로 판단하여 데이터를 정제하여 다시 리스트를 업데이트한다.

```
before: [1, 41, 2, 320, 7, 3, 2, 2, 2, 2, 2, 2, 2, 3, 4, 6, 4, 4, 4, 4, 4, 5, 6, 5, 5, 9
after : [0, 41, 2, 320, 0, 0, 2, 2, 2, 2, 2, 2, 2, 4, 4, 6, 4, 4, 4, 4, 4, 6, 6, 0, 0, 0
```

잡음 제거

```
20, 78, 53, 30, 12, 12, 12, 11, 11, 13, 19, 12, 14, 9, 10, 10, 9, 9, 9, 8, 10,
20, 78, 53, 30, 12, 12, 12, 11, 11, 13, 19, 12, 14, 10, 10, 10, 0, 0, 0, 0, 10,
```

3) 장애물과 도로를 구분하여 주행가능 영역 도출하기



① **get_dot(list)**

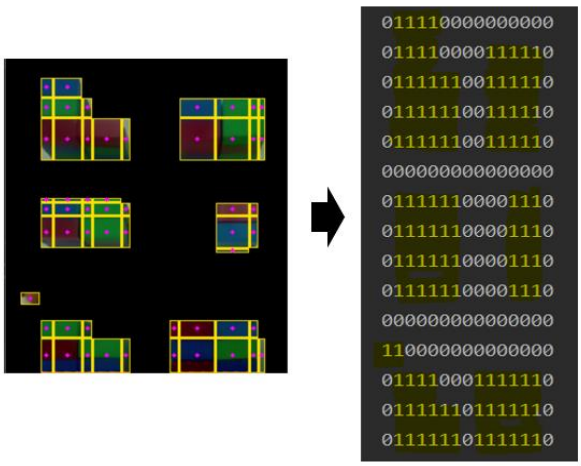
: 앞서 잡음을 제거한 결과를 이용하여 각각의 행과 열을 기준으로 각 라인별로 검사를 하여 연속된 0 으로 이어지다가 숫자 군집이 나타나면 그 군집의 중앙값을 좌표로 저장한다. 이 때 이 좌표값들은 컨테이너의 테두리가 있을 법한 라인으로 판단되어 저장된다.

② **grid_area_check()**

: 이렇게 모든 과정을 거쳐 x 좌표 리스트와 y 좌표 리스트가 모두 구해지면 조합을 통해 전체 이미지를 잘게 슬라이싱한다. 그런 뒤 작은 사각형들의 무게중심의 GrayScale 값을 확인하여 90 이상 255 이하의 값을 지니면 장애물이 아닌 도로의 영역이라고 생각하여 검은 사각형으로 칠해준다.

```
x_dox: [0, 2, 14, 34, 47, 76, 86, 115, 124, 162, 172, 208, 242, 251, 258, 280]
y_dox: [0, 2, 20, 41, 62, 107, 145, 150, 164, 198, 203, 243, 257, 272, 291, 328]
```

4) 이미지를 좌표 데이터로 변환



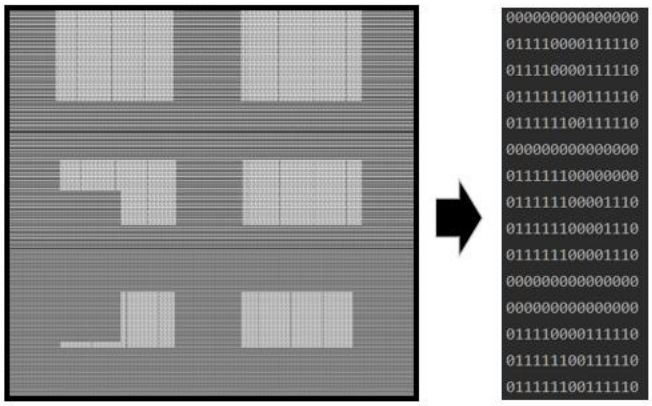
```
0111100000000000
0111100001111110
0111111001111110
0111111001111110
0111111001111110
0000000000000000
0111111000011110
0111111000011110
0111111000011110
0111111000011110
0000000000000000
1100000000000000
0111100011111110
0111111011111110
0111111011111110
```

draw_result_map()

: 이미지로 도출된 맵은 실제로 터틀봇이 주행 시 사용할 수 있는 맵이 아니다. 따라서 경로 알고리즘을 적용할 수 있고 터틀봇이 이용할 수 있는 맵으로 바꿔주기 위해 도로의 영역인 검은색은 0으로 그외의 영역은 1로써 장애물을 나타내 좌표 데이터를 완성시킨다.

1-2) 경로 도출 알고리즘

1) 실제 거리를 반영하여 맵 도출



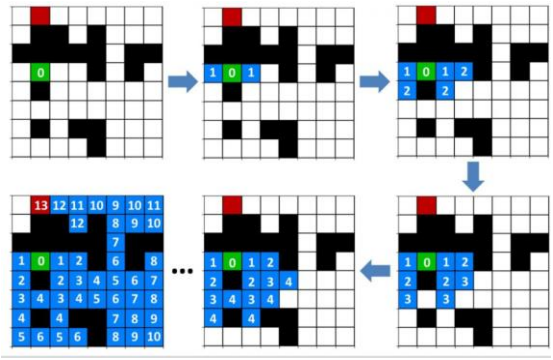
전체 맵 도출 시, 마지막 단계에서 경로 알고리즘이 적용될 수 있게 정수 변환 단계를 거친다. 이때 모든 픽셀에 대한 정보를 다 담게 되면 데이터가 필요 이상으로 커지기에 경로 알고리즘 적용시 불필요한 연산이 발생할 수 밖에 없다. 따라서 정수데이터로 변경할 때는 이동거리 단위를 설정하여 실제로 터틀봇이 움직일 단위만큼 맵을 나눠 전체적인 맵의 사이즈를 축소할 필요가 있다.

먼저 적절한 이동거리 단위를 반영하기 위해 항만의 실제 크기와 실제 거리를 유추할 수 있는 대상을 설정하여 그 대상이 차지하고 있는 픽셀의 개수를 이용해 하나의 픽셀이 담당하는 실제 크기를 구한다. 그런 뒤 앞서 설정한 이동거리 단위로 대치 가능한 필요한 픽셀의 수를 구하고 cv2.resize()를 하여 이미지를 축소시킨다. 정수데이터로 변환할 때는 원본 이미지가 아닌 축소시킨 이미지를 이용하여 변환한다.

```
real_width = 150                                  # 기준으로 설정할 대상: 드론이 담당하는 구역의 한면
picture_width = self.img_result.shape[0]
one_pixel = real_width/picture_width  # 하나의 픽셀이 담당하는 실제거리
print('한픽셀에 해당하는 실제 거리(cm):', one_pixel)
```

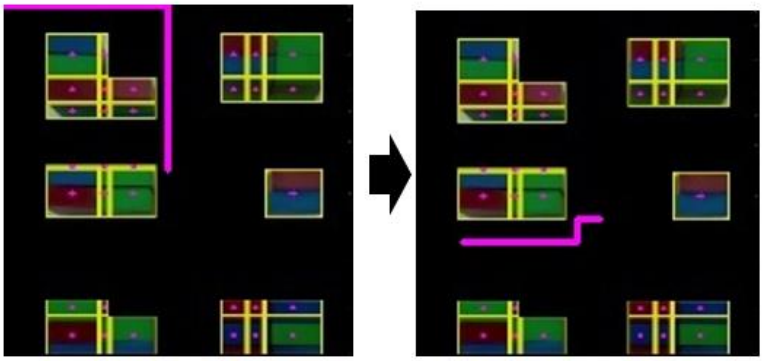
```
한픽셀에 해당하는 실제 거리(cm): 0.4573170731707317
```

2) 시작점과 목적지의까지의 경로 알고리즘



목적지까지의 최단 경로를 구할때, BFS(너비우선탐색)를 이용하여 최단 경로 알고리즘을 도출한다. 앞서 구한 좌표 데이터를 바탕으로 0이면 이동할 수 있는 노드, 1이면 이동할 수 없는 노드로 가정하고 최단 경로를 도출한다. 시작 지점은 터틀봇이 있는 위치이며 목적지는 드론에서 발견된 객체의 위치이다.
정밀한 주행을 위해 터틀봇의 몸체로 인해 차지되는 크기를 반영하여 경로 데이터를 만들도록 한다. 경로 데이터는 4방위의 표시로 방향을 알려준뒤 얼마나 이동할지 숫자데이터를 붙여서 표현한다.

▼ 초기 시작점에서 목적지까지 이동 후 목적지점에서 또 다른 경로로 이동하는 모습



```
침입자의 위치: (x, y) = (8, 9) | 좌표값 = 0
터틀봇의 위치: (x, y) = (1, 1) | 좌표값 = 0

원본 경로:  LLLLLLL/GGGGGGGG
최종 경로:  L70/G80
```

```
침입자의 위치: (x, y) = (3, 11) | 좌표값 = 0
터틀봇의 위치: (x, y) = (9, 10) | 좌표값 = 0

원본 경로:  R/G/RRRRR
최종 경로:  R10/G10/R50
```

1-3) 객체 인식 알고리즘

1) 객체 인식 및 판단

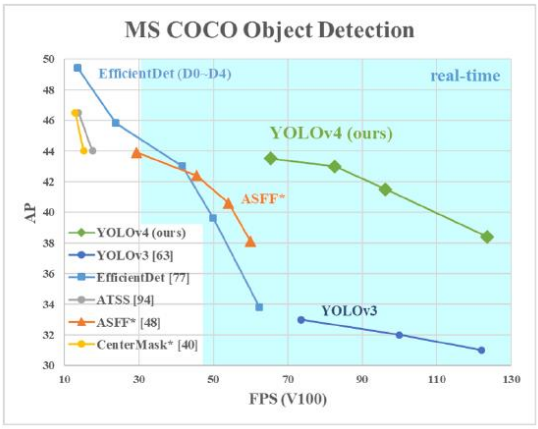
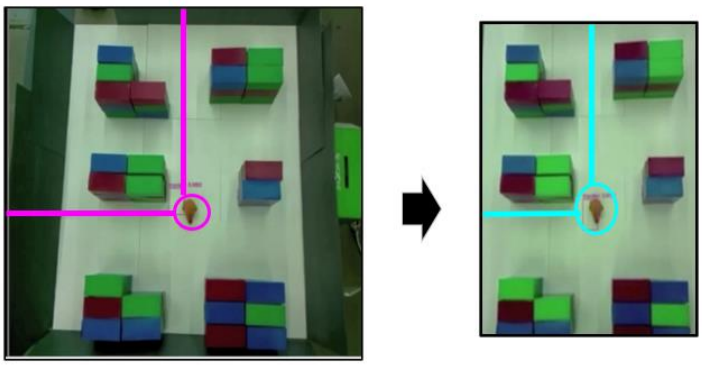


그림 4. MS COCO Object Detection 성능 비교 [4]

YOLO v4는 single GPU로도 충분히 real-time Object Detection 이 가능하고 그림에서 볼 수 있듯이 다른 detector들과 비교 했을 때 속도가 월등히 빠르다. 따라서 저사양 시스템으로 빠르고 정확하게 물체를 탐지해야 하는 이 프로젝트에 가장 적합하다고 판단해 YOLO v4를 사용했다. 홉저버에서는 Yolo를 이용하여 상공에서 사람의 모습을 인지할 수 있게 드론을 학습시켰으며 터틀봇은 사람의 형태를 모두 파악할 수 있도록 학습시켰다.

2) 객체 위치 (좌표) 도출

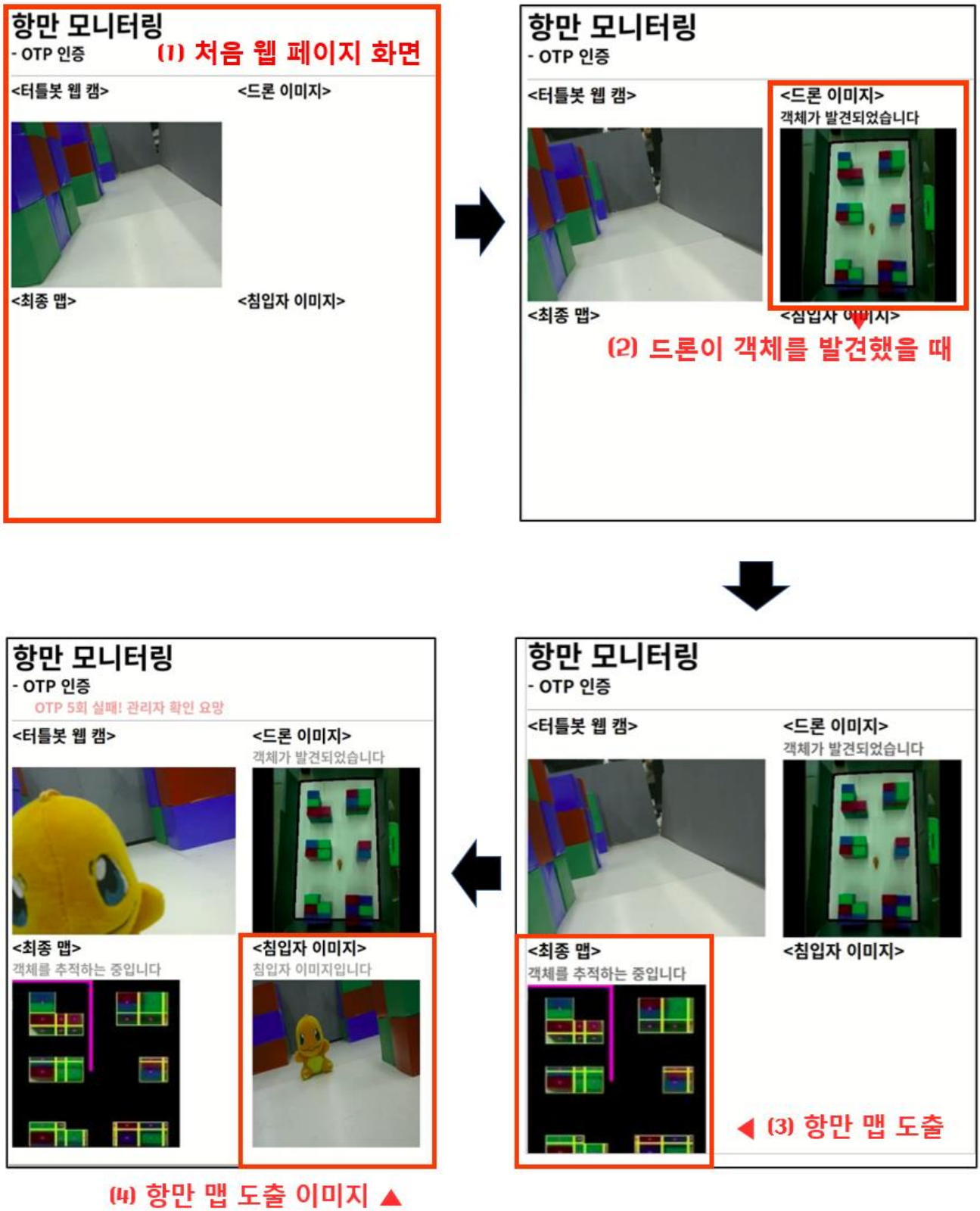


Yolo를 이용하여 Object Detection을 거치면 드론의 화면을 기준으로 (0, 0)부터 계산된 객체의 위치가 주어진다. 이는 터틀봇과 통신할 때 적절하지 않은 객체의 위치이며 드론이 이동할때마다 변경되는 상대적인 위치이다. 따라서 이를 절대적인 위치로 바꾸기 위해 드론이 지정하는 구역내에서 해당 맵을 기준으로 객체의 위치를 다시 계산하도록 한다. 객체의 절대적인 위치를 구하기 위해서는 약속된 절대적인 맵이 있어야 한다. hoserver에서는 각각의 드론이 담당하는 구역을 절대적인 맵으로 두어 해당 구역을 기준으로 객체의 위치가 계산되도록 설계한다. 화면에 들어온 이미지에서 담당 구역이 절대적인 맵이 될 수 있도록 앞서 서버의 맵 도출에 사용됐던 지정 구역 도출 알고리즘을 이용하여 전체 이미지에서 지정 구역만 crop한다. 그런 뒤 crop된 이미지를 다시 yolo의 Object detection을 거치도록 한다. 이후 labeling 과정에서 주어지게 된 객체의 위치는 드론의 화면이 아닌 담당하는 구역을 기준으로 계산된 결과이므로 경로 알고리즘에 적합한 데이터가 된다.

5. UI 사용법 (구성 및 설명)

1) 관제 시스템의 메인 UI/UX

관제 시스템 화면 구성시, 사람의 직접적인 개입 없이도 간접적으로 모든 상황을 한눈에 파악할 수 있도록 현장을 담은 영상이나 이미지들로 배치하였다. 최상단에는 OTP인증 결과를 나타내는 텍스트를 배치하였고 좌측 상단에는 터틀봇의 시야의 웹 캠을 두어 지상에서의 항만 상황이 보일 수 있도록 하였다. 우측 상단에는 드론이 전송한 객체 포착 이미지를, 하단 좌측에는 항만 이미지로 도출한 항만의 맵, 우측에는 침입자의 모습을 담은 현장 사진을 배치하였다.



[1] 처음 웹 페이지

처음 웹 페이지에서는 현재 운용되고 있는 터틀봇의 웹 캠 영상을 확인할 수 있다.

[2] 드론이 객체를 발견했을 때

드론이 객체를 인식했을 때 항만의 이미지와 함께 객체가 발견됐다는 메시지를 브라우저에 띄운다.

[3] 항만 맵 도출 이미지

드론이 전송한 항만 이미지를 서버에서 처리한 항만 맵과 터틀봇이 객체까지 도달할 최단 경로를 표시한 이미지와 객체를 추적 중임을 알리는 메시지를 사용자에게 보여준다.

[4] 침입자가 발견되었을 때

OTP 인증 5회 실패 시 객체를 침입자로 판단하고 이 때 터틀봇이 전송한 침입자라고 판단되는 객체의 이미지를 띄운다

상황에 따른 OTP 인증 확인 메시지 출력

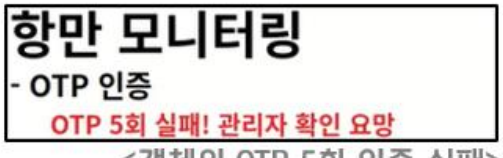
객체의 OTP 인증 결과에 따라 총 4가지의 메시지를 사용자에게 보여준다.

메세지는 OTP 인증 성공, OTP 인증 실패, OTP 인증 5회 이상 실패, OTP 인증 시간 초과가 있는데 드론에게 탐색을 재요청하는 경우나 관리자의 확인이 필요한 상황 등 관리자의 판단이 요구될 때는 빨간색 텍스트로 표현하여 사용자의 주의를 끌 수 있도록 하였다.

<객체의 OTP 인증 성공>

<객체의 OTP 인증 시간 초과>

<객체의 OTP 인증 실패>



<객체의 OTP 5회 인증 실패>

6. 개발환경 (언어, Tool, 사용시스템 등)

사용 요소	분류			
	Drone-Side	**서버, Back-end**	**사용자 서비스**	**Turtlebot-Side**
운영 체제	Linux	Windows	Windows	Linux
사용 언어	Python	Python	Javascript	Python, C
프레임워크/ 라이브러리	Yolo_v4	TensorFlow, openCV	Express	ROS
DBMS	-	MongoDB	MongoDB	-
데이터 통신	TCP/IP	Mosquitto, TCP/IP	-	Mosquitto
개발 도구	Pycharm	Pycharm	Atom	Pycharm, Visual Studio
협업/ 버전 관리	Git/ Git hub			

□ 개발 프로그램 설명

1. 파일 구성

Module	역할, 설명
드론	
DRONE_control.py	지정된 구역이 한 화면에 담길 수 있게 드론의 모션을 조정한다.
DRONE_main.py	상공에서 비행 중인 드론이 특정 객체를 발견하면 YOLO를 이용하여 이 객체가 유효한 사람인지 판단하고 맵에서의 객체 좌표를 구한다
DRONE_client.py	드론과 서버의 TCP/IP 통신을 담당하며 드론에서 얻은 항만의 이미지와 객체의 위치 좌표를 서버로 전송한다.
서버 (Back-end)	
MongoDB.py	서버의 데이터베이스에 접속하여 목적에 맞는 컬렉션을 생성하고 이미지 또는 string 데이터를 저장한다.
MakeMap.py	드론으로부터 받은 항만 사진을 보정하고 터틀봇이 이동할 맵을 생성.
Find_path.py	생성된 맵에서 객체의 위치까지의 최적의 경로를 찾고 경로 데이터와 경로가 그려진 이미지를 반환한다.
Find_person.py	터틀봇의 시야 영상을 통해 사람인 객체를 판단하고 OTP 인증을 시도한다. 그런 뒤, 그 결과와 관련된 핵심 로직 구현한다.
SERVER.py	드론과 터틀봇의 통신을 담당하는 파일로 TCP/IP통신 또는 MQTT를 사용하며 데이터를 받아 서버 내부에서 자체적을 가진 로직을 구동시키고 새로운 데이터를 도출하고 이러한 값들을 통신하는 파일이다.
서버 - Web page (node.js)	
app.js	객체를 생성하고 환경 설정을 하는 파일이다.
bin/www	웹서버를 실행하는 코드가 들어있는 파일이다.
modules/socketIoImage.js	드론이 포착한 객체의 사진, 서버가 만든 항만의 최종 map 이미지 그리고 터틀봇이 침입자의 이미지를 서버의 데이터베이스로 전송한다. 이 때 데이터베이스에서 저장된 이미지를 클라이언트에게 보여주기 위해 이미지를 가져와 소켓으로 전송한다.
modules/socketIoOTP.js	터틀봇이 객체로부터 OTP 인증을 받은 결과를 서버가 MQTT 프로토콜 통신으로 받은 후 데이터베이스에 저장한다. 이 때 OTP 결과를 클라이언트에게 시각적으로 보여주기 위해 소켓으로 전송한다.
public/javascript/event.js	서버와 소켓 통신을 위한 자바스크립트 파일이다.
public/hobserver.html	웹 브라우저에서 데이터를 보여주는 html 파일이다.
터틀봇	
Lidar.py	터틀봇의 부착된 Lidar로부터 주변 사물의 거리를 측정한다.
Main.py	목표물 추적 프로그램을 실행시키고 프로그램의 전체적인 흐름을 제어한다.
Move.py	여러 가지 센서값을 읽어와 터틀봇의 움직임을 제어하고 목표물을 탐색한다.

Mqtt_communicate.py	터틀봇과 서버의 mqtt통신을 담당한다.
Mqtt_publisher	터틀봇에 장착된 OTP인증결과를 전송하거나 목표물을 발견하지 못했을 때 서버에게 목표물 재탐색을 요청한다.
odometry	터틀봇에 부착된 다이나믹 셀로부터 현재 위치의 x,y좌표값을 읽어들인다.
Otp.py	OTP인증을 구동시키는 파일이다.

2. 함수별 기능

1. 서버

① MongoDB.py

Class	Method	역할, 설명
User_Map	-	항만과 관련된 이미지를 담는 객체이다.
User_OTP_T	-	String type의 OTP 인증 결과를 담는 객체이다.
User_OTP_P	-	목적지에 도착 후 현장 이미지를 담는 객체이다.
MongoDB	앞서 서술된 여러 형식의 데이터 객체들을 생성하여 DB에 내용을 저장한다.	
	storeImg_map (img, filename)	"User_Map" 객체에 실시간 도출된 맵을 저장한다.
	storeImg_otp (img, filename)	"User_OTP_P"객체에 침입자의 이미지를 저장한다.
	storeStr_otp (word)	"User_OTP_T" 객체에 OTP 결과(word)를 저장한다.

② SERVER.py

Flow

```
mongo = MongoDB()
mqtt = mqtt.Client("loadFinder")
mqtt.connect("localhost", 1883)
TCP_IP = '192.168.0.15'
TCP_PORT = int(sys.argv[1])
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(True)
```

1) 데이터를 주고받을 환경을 구성 : 통신을 위해서 MQTT 클라이언트와 TCP/IP 서버 소켓을 데이터 저장을 위해 mongoDB의 객체를 생성한다

```
if map_flag:
    time.sleep(2)
    makeMap = Make_Map(decimg)
    makeMap.get_Max_contour()
    makeMap.delete_destroy()
    map = makeMap.draw_result_map()
    map_flag = False
```

2) 들어온 이미지를 이용하여 맵을 도출 : 데이터 (항만 이미지, 객체의 위치 좌표)가 들어오면 MakeMap객체를 생성해서 맵을 도출하는 단계로 넘어간다. 이때 완성된 맵은 객체 추적모드 동안 다시 참조될 수 있기에 데이터베이스에 저장한다.

3) 목표 객체까지의 최단 경로 생성 : 생성된 맵 위에서 터틀봇의 위치에서 객체까지의 경로를 도출.

```
    if location[0]//10 != postLocation[0]//10 and location[1]//10 != postLocation[1]//10:
        find_path = Find_path(decimg, location, map)
        find_path.path_algorithm(postLocation)
        postLocation = location
        img, path_data = find_path.drawing_path()
        mqtt.publish("pathList", json.dumps({"data": path_data}))
```

```
find_person = Find_person()
result = find_person.check_person()
if result == "Success" or result == "Real_Fail":
    conn.send('DRONE_close'.encode('utf-8'))
    break
else:
    conn.send('DRONE_again'.encode('utf-8'))
```

4) 객체의 OTP 인증 및 재탐색 판단:
주어진 경로를 주행하는 터틀봇의 영상을 읽어 객체를 확인하고 OTP인증을 시도한다. 그런 뒤 터틀봇으로부터 인증 결과를 받고 드론에게 추적모드를 유지할지 중단할지 결과를 전송한다.

③ MakeMap.py

Class	Method	역할, 설명
MakeMap	주어진 항만의 이미지에 operation을 가하여 터틀봇이 주행할 수 있는 맵을 완성한다.	
	get_Max_contour()	주변의 잡음을 제거하고 담당하는 구역만 Map으로 도출하기 위해 이미지 전체를 cv2.findContours() 처리해 가장 큰 면적을 가진 등고선(contours)을 도출한다.
	order_point(contour)	가장 큰 면적을 가진 등고선(contour)을 맵의 테두리로 가정하고 해당 등고선의 정보를 인자로 받아 가상의 꼭짓점을 구한다.
	delete_destroy()	사각형으로 맵을 복원하기 위해 맵에 해당되는 부분만 크롭한다. 그런 뒤 warpPerspective()을 이용하여 이미지의 휘어짐을 제거한다.
	make_contour()	이미지를 Grayscale로 변환하여 특정값 이하의 value를 가진 픽셀과 그렇지 않은 픽셀로 구분하여 양분화된 value로 값을 조정하여 전체 픽셀의 Grayscale 값이 이진화가 되도록 조정한다.
	get_dot(list)	앞서 이진화를 거친 Grayscale 이미지에서 각각의 행과 열을 기준으로 특정 값을 갖는 픽셀의 수를 counting해 그 개수가 일정 수준 이상인 행 또는 열을 컨테이너의 테두리일 판단해 좌표로 변환하여 리스트에 담아둔다.
	pixel_content()	컨테이너로 판단되는 영역의 무게중심을 구해서 해당 픽셀이 가진 grayscale의 값이 기준값 이상인 경우는 컨테이너의 영역으로 판단하여 일정한 값의 픽셀로 변경한다. 끊어진 테두리를 이어주도록 한다.
	draw_result_map()	한 픽셀 당 실제거리를 계산한 뒤, 한 픽셀이 10cm를 담당할 수 있게 이미지를 cv2.resize()한다. 그런 뒤 pixel_content를 이용해 해당 이미지를 저장하고 실제 경로 알고리즘에 사용될 좌표를 넘겨줄땐 잡음으로 인한 오차를 줄이기 위해 터틀봇이 이동할 수

		있는 칸의 크기로 0.1배수 더 줄인 리스트 객체를 넘겨준다.

④ **Find_path.py**

Class	Method	역할, 설명
Find_path	도출된 맵에서 터틀봇의 위치와 목표 객체의 위치까지의 최단 경로를 구하는 클래스	
	path_algorithm(postLocation)	주어진 터틀봇의 위치(postLocation)를 기준으로 목표 객체의 위치까지의 최단 경로를 구한다.
	path_MQTT(pathData)	구해진 경로데이터(pathData)를 터틀봇과 약속한 데이터의 형태로 변환한다.
	drawing_path()	구해진 최단 경로를 맵 위에 그려주고 해당 이미지를 반환.

⑤ **Find_person.py**

Class	Method	역할, 설명
MQTT_OTP _Subscriber	터틀봇이 발행하는 OTP 인증 결과 메시지를 읽어오는 클래스	
	on_connect (client, userdata, flags, rc)	생성된 Mqtt Client 객체가 OTP 결과값과 관련된 "otp_result" 토픽을 구독하게 한다.
	on_message (client, userdata, msg)	메시지 대기 모드로 루프를 돌면서 터틀봇으로부터 들어오는 메시지를 받아 데이터베이스에 저장한다. 1이면 Success, 0이면 Fail, 3은 Time over로 변경하여 데이터와 결과를 매칭시킨다.
Yolo_checker	주어진 이미지에 대하여 객체의 존재 유무를 판별하고 사람인지 확인한다.	
	isPerson(img)	터틀봇으로부터 들어오는 영상(img)에서 발견되는 객체가 사람인지 판단하는 함수이다. 사람을 학습시킨 yolo를 이용하며, 만약 사람이 발견되면 True를 반환한다.
Find_person	OTP 인증의 핵심 로직을 구현하는 Class이다.	
	Check_person()	터틀봇의 영상을 읽다가 사람이 발견되면 OTP 시작 알림을 보내고 터틀봇의 인증 결과를 기다린다. 인증 결과가 성공인 경우는 프로그램을 종료하고 실패 5회 초과나 시간 초과의 경우는 현장 사진 저장한다.

2. 드론

① **DRONE_control.py**

Class	Method	역할, 설명
Drone	드론의 모션을 포함하는 클래스	
	open()	드론과 컨트롤러를 연결한다.

	sendLostConnection()	드론과 컨트롤러의 연결이 끊어졌을 때 수행할 명령을 설정한다.
	setEventHandler()	드론으로부터 data 를 수신했을때 실행할 함수를 설정한다.
	sendTakeOff()	드론에게 이륙신호를 송신한다.
	sendRequest()	드론에게 data 를 요청한다.
	sendControlPosition()	드론에게 지정 위치로 이동 명령을 송신한다.
	sendControlWhile()	드론에게 호버링 명령을 송신한다.
	sendFlightEvent()	드론에게 지정비행 명령을 송신한다.
	sendLanding()	드론에게 착륙비행 명령을 송신한다.
	sendStop()	드론에게 stop 명령을 송신한다.
	close()	드론과 컨트롤러의 연결을 해제한다
-	eventAltitude()	드론으로부터 data가 들어오면 온도와 기압 드론 자세 정보를 출력한다.

② DRONE_Client.py

Class	Method	역할, 설명
DRONE_Client	드론이 항만의 이미지와 객체의 위치 좌표를 서버로 전송하는 클래스	
	sendToServer(img, lct)	적절하다고 판단된 이미지(img)를 받아 binary 타입으로 변환한 뒤, 객체의 위치(lct) 데이터를 이어 붙인다. 그런 뒤, 해당 data를 서버에게 소켓통신으로 전송한다.
	fullMapChecker(img)	객체의 모습이 담긴 항만의 이미지가 서버로 전송되기전 거치는 마지막 단계로 드론이 담당하는 구역이 모두 담겼는지 맵의 가이드 영역 표시와 맵의 중심점과 드론 화면의 중심점을 비교하여 판단 후 결과와 관련해 참, 거짓을 반환한다.
	sockWaitAnswer()	추적모드를 유지할지 중지할지 서버의 답변을 기다리는 함수로 서버로부터 받은 결과를 반환한다.
	sockClose()	서버와의 소켓 통신을 끊는다.

③ DRONE_Main.py

Class	Method	역할, 설명
Detect_Obj_yolo	터틀봇이 발행하는 OTP 인증 결과를 읽어와 데이터베이스에 저장하는 클래스	
	imgProcessing(img)	객체의 좌표를 지정된 영역 내에서의 위치로 구하기 위해 휘어진 이미지를 보정하고 맵 부분만 크롭한다.
	check_obj_yolo(image)	학습된 yolo를 이용하여 객체를 발견한다.
	obj_labeling(boxes, classIDs, confidences, idxs, image)	발견된 객체를 이름과 라벨링한 뒤 객체의 중심 좌표를 구한다.

3. 터틀봇

① Lidar.py

Class	Method	역할, 설명
Lds	터틀봇에 부착되어 있는 Lidar를 통해 측정한 값을 가공하여 터틀봇에 제공한다	
	callback_lidar(scan)	scan topic을 통해서 Lidar의 값을 전달 받는다.
	get_right()	터틀봇의 오른쪽 30도 방향의 Lidar값을 반환한다
	get_left()	터틀봇의 왼쪽 30도 방향의 Lidar값을 반환한다
	get_front()	터틀봇의 정면 Lidar값을 반환한다

② move.py

Class	Method	역할, 설명
Turtlebot_move	서버로 부터 받은 경로를 바탕으로 터틀봇을 움직이고 목표물을 발견한다.	
	order_reversed(order)	터틀봇이 다시 원래 자리로 되돌갈 수 있도록 서버로터 받은 이동경로를 재생성 한다.
	move(linear, angular)	터틀봇을 앞으로 이동시키며 동시에 좌우에 물체가 일정거리 이상 근접하면 회피한다.
	move_go(dis)	터틀봇의 좌표를 확인하며 각방향으로 얼마만큼 이동했는지 확인하며 전달받은 거리만큼 이동했을때 멈춘다, 동시에 이동중 목표물을 발견하면 OTP인증을 요구한다.
	move_front(dis))	터틀봇의 0º 방향으로 회전시키고 전달받은 거리만큼 이동시킨다.
	move_back(dis)	터틀봇의 180º 방향으로 회전시키고 전달받은 거리만큼 이동시킨다.
	move_left(dis)	터틀봇의 90º 방향으로 회전시키고 전달받은 거리만큼 이동시킨다.
	move_right(dis)	터틀봇의 -90º 방향으로 회전시키고 전달받은 거리만큼 이동시킨다.
	move_stop()	터틀봇을 정지시킨다.
	start(order)	전달 받은 경로로 목표물 탐색을 시작한다

③ odom.py

Class	Method	역할, 설명
Odom	터틀봇의 위치 정보를 전달한다.	
	callback_odom(msg)	OpenCR에서 다이나믹셀의 위치값을 이용해서 계산한 터틀봇의 위치정보를 odom topic을 통해 전달받는다
	get_roll()	roll 값을 반환한다.
	get_pitch():	pitch 값을 반환한다.

	get_yaw()	yaw값을 반환한다.
	get_position_x()	터틀봇의 x좌표 값을 반환한다.
	get_position_y()	터틀봇의 y 좌표값을 반환한다.

④ mqtt_communicate.py

Class	Method	역할, 설명
MQTT_communicate	MQTT 통신을 이용하여 주행경로를 전달받거나 OTP인증을 요구받고 목표물의 재확인을 요청한다.	
	callback_server(msg)	서버로부터 터틀봇의 이동 경로를 받는다.
	callback_otp(msg)	서버로부터 OTP인증을 요구받는다.
	get_order():	터틀봇의 이동경로를 반환한다.
	get_otp_flag()	OTP인증을 실행하기 위한 otp_flag를 반환한다
	otp_start()	OTP인증을 실행한다.
	otp_lost()	묵표물의 재확인을 서버에게 요청한다.

⑤ MQTT_publisher.py

Class	Method	역할, 설명
MQTT_publisher	서버에게 OTP인증의 결과를 전달하거나 목표물의 재확인을 요청한다.	
	otp_value_publish()	pos 값을 서버에게 전달한다.
	time_over_value_publish()	서버에게 목표물의 재확인을 요청한다.
	success_or_fail_publish()	OTP를 최대 5번 실행시키고 결과를 서버에게 전달한다.

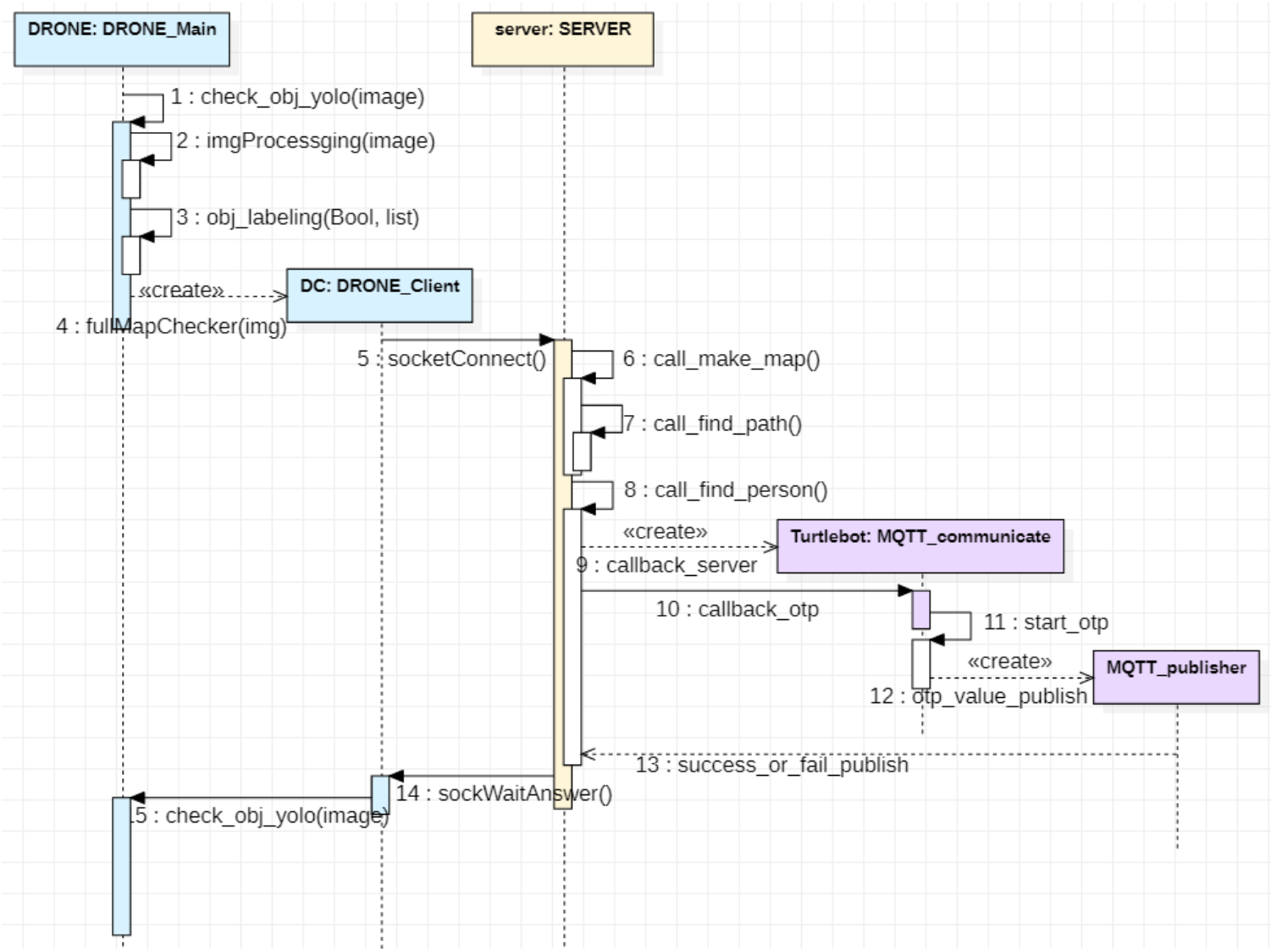
⑥ otp.py

Class	Method	역할, 설명
-	OTP를 실행시키고 실패또는 성공여부를 판단한다.	
	otp_check()	OTP를 실행시키고 성공일 경우 "1", 실패일 경우 "0"을 반환한다.

⑦ main.py

Class	Method	역할, 설명
-	터틀봇을 이용하여 침임자를 탐색한다	
	main	터틀봇을 이용하여 침입자의 탐색을 시작한다.

3. 주요 함수의 흐름도



4. 기술적 차별성

맵 생성: 기존의 자율주행 로봇이 라이다 센서를 통해 직접 돌아다니며 맵을 생성하였지만 홉저버에서는 드론 카메라가 촬영한 영상을 이용해 항만의 맵을 한번에 구할 수 있게 하였다. 촬영한 영상을 OpenCV 라이브러리를 이용해 영상처리를 하고 항만의 보다 선명한 맵을 구할 수 있다.
MQTT 프로토콜의 장점인 토픽의 구독, 발행의 기능을 이용해 실제 항만에서 여러 대의 자율주행 로봇이 있는 경우 길찾기 모션을 주는 토픽을 구독한다면 여러대가 동시에 정보를 주고 받을 수 있다.
기존 항만에서 드론은 단순히 CCTV 의 사각지대를 보완하기 위해서 사용됐다면 Hobserver 는 드론을 이용해 CCTV 사각지대를 보완함과 동시에 터틀봇을 이용하여 드론으로 발견한 목표물을 추적하고 OTP 인증을 요구해 항만의 보안성을 높인다.
주행시 Lidar 센서를 사용한다: 초음파 센서와 다르게 Lidar 센서는 360° 모든 각도의 물체의 거리 측정이 가능해 장애물을 좀더 효율적으로 회피할 수 있다.

□ 개발 중 장애요인과 해결방안

1) 드론이 촬영한 항만 영상에서의 Map 도출 과정에서 노이즈 처리

장애 요인) 항만의 이미지가 들어오면 주변의 잡음과 정확하지 않은 경계로 인해 도로와 컨테이너의 구분이 어려워 주행할 수 있는 영역과 그렇지 못한 영역이 뚜렷히 구분된 맵을 도출하기가 어려움.

해결 방안) 이미지에 다양한 전처리 operation을 가해 해결한다. 등고선의 정보에만 의지하는 것이 아닌 밝기 차이와 색감의 정보도 고려하여 픽셀마다의 고유의 특징을 잡고 이진화된 좌표객체로 변화하여 실제 컨테이너의 위치라고 가정할 법한 좌표들로만 확률을 높여서 계산한다. 또한 조합을 이용하여 전체 항만 이미지를 잘게 사각형으로 쪼개 중심 픽셀 좌표의 색상고도의 값이 일정 수준인 픽셀들만 컨테이너의 영역으로 잡아 좀 더 정확한 맵을 구한다.

2) 탐색 영역 내에서의 객체의 상대적 위치 구하기

장애 요인) 드론에서 yolov4에서 인식된 객체의 위치를 판단하고 주는 객체의 위치 좌표는 현재 드론의 화면에서의 절대적인 x, y의 값이다. 하지만 추적을 해야하는 터틀봇은 드론의 화면의 기준에 구해진 절대적인 위치가 아닌 지정된 구역내에서의 x, y의 값이 필요하다.

해결 방안) 드론과 터틀봇이 모두 공유할 수 있는 절대적인 객체의 좌표가 필요하다. 이 때 사용하는 기준을 각 드론이 담당하는 구역으로 설정하여 객체의 좌표를 구할 때 담당 구역 내에서의 상대적인 위치를 구할 수 있도록 좌표 개념을 통일하였다.

3) 터틀봇 주행 시 컨테이너와 같은 장애물을 피하기

장애 요인) 목적지까지의 경로에 맞춰 주행하는 터틀봇은 맵에서 잡히지 못한 미세한 장애물들과 정렬되지 못한 컨테이너와 같은 장애물에 부딪힐 위험이 있으며 일정치 않은 Odometry 의 값으로 인해 이러한 장애물을 회피하고 원하는 위치로 정확하게 이동하는 것에 어려움이 있었다.

해결 방안) 터틀봇은 주어진 경로대로 주행하면서 튀어나온 컨테이너나 드론에서 잡히지 못한 얘기치 못한 장애물들을 피해가기 위해 Odometry 값에만 의지하는 것이 아닌 Lidar 센서를 보조적으로 이용하여 주변 사물과에 거리를 측정하고 장애물을 회피하며 안전하게 목표 위치까지 주행한다.

4) 터틀봇 배터리의 전력 부족으로 인한 성능저하

장애 요인) 터틀봇 배터리의 출력전압이 낮아 터틀봇 메인보드인 Jetson Nano의 CPU core를 하나밖에 쓸 수 없었고 그로 인해 터틀봇 자체에서 영상처리가 어려웠다.

해결 방안) 터틀봇이 직접 객체의 유무를 판단하는 것이 아닌 판단의 주체를 서버로 둔다.
서버는 터틀봇의 웹캠을 읽어들여 yolo로 객체 발견 여부를 즉각적으로 판단하고 터틀봇에게 그 결과를 알려준다 이로써 터틀봇은 모션만 구동하고 객체 판단 로직은 서버가 맡아 앞서 발생한 저전력으로 인한 문제를 해결한다.

5) Push 버튼의 chattering

장애 요인) OTP 모듈의 입력장치로 사용한 Push 버튼이 전자 회로 내의 스위치 접점이 닫히거나 열리는 순간에 기계적인 진동에 의해 매우 짧은 시간안에 스위치가 붙었다가 떨어지는 것을 반복하는 현상인 chattering 으로 인해 중복입력이 매우 빈번하게 일어났다.

해결 방안) chattering 으로 인한 문제를 제어하기 위해 버튼이 한번 눌리면 10ms 동안 입력을 받지 않고 10ms 후에 다시 받는 방식을 사용했다.

□ 개발결과물의 차별성

1) 드론을 이용한 항만 감시

현재 항만에는 물류 분실, 불법 입국 등 여러 가지 안전 사고 문제가 발생하고 있다. 이와 관련된 해결책으로 CCTV 가 설치되어 있지만 사각지대가 많은 항만의 특성상 감시의 효율성이 떨어지고 더불어 기기 자체의 성능이 뛰어나지 않아 물체를 정확하게 인식하기 어려운 문제가 발생한다. 때문에 KOBOT 팀에서는 드론을 사용하여 CCTV 가 명확하게 인식 불가능한 구역을 감시할 뿐만 아니라 보다 넓은 시야각을 가지고 자율적으로 구역을 이동하며 감시기능을 수행한다.

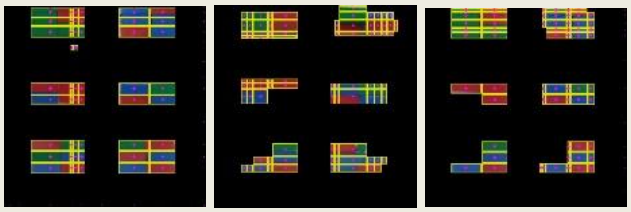
2) MQTT 프로토콜 사용

IOT 기기에 특화된 MQTT 프로토콜을 사용한다.
이는 저전력, 신뢰할 수 없는 네트워크 환경에서도 데이터 전송에 효과적이기에 넓은 항만에서의 사용이 적합하다. 더불어 메시지 브로커의 특징인토픽의 구독, 발행의 기능을 이용해 실제 항만에서 여러 대의 자율주행 로봇이 있는 경우 길찾기 모션을 주는 토픽과 관련하여 여러대가 동시에 정보를 받고 그 중에서 가장 가까운 로봇이 움직일 수 있어 더욱 효과적일 것이다.

3) 전체 Map 업데이트 속도 증진

- 터틀봇과 같은 지상의 로봇들은 자율주행을 위해서 실제 맵 탐색의 과정이 필요하다. 이때 드론을 이용해 상공에서 찍은 이미지를 이용해 한번에 맵을 만든다면 자율 주행 로봇의 맵 탐색 과정보다 시간적 부담과 비용을 줄일 수 있을 것이다.

<컨테이너의 배치에 따른 다양한 맵>



- 항만 특성상 컨테이너와 같은 화물의 승선과 화선에 의해 환경의 변화가 잦다. 이러한 변화에 대응해 드론을 이용하여 맵을 구성한다면 날마다 최신의 항만의 전체 Map 을 빠르게 업데이트 시킬 수 있을 것이다.

4) 사람의 간섭을 최소화한 경비 시스템

경비가 필요한 공간을 매번 사람의 인력에 의존하여 확인하는 것은 감시의 강도도 떨어지고 시간적 측면이나 비용적 측면에 있어서도 효율적이지 못한 방식이다. 하지만 드론과 서버에서 YOLO 를 이용해 사람을 인식하고 터틀봇을 이용해 OTP 인증을 받는 과정까지 관제 시스템을 통해 한눈에 지켜볼 수 있게 된다면 기존의 경비 시스템이 지닌 인력 낭비와 시간적, 비용적 부담을 줄여주는 해결책이 될것이다. 따라서 홉저버는 보완 인력을 최소화하는 방식으로 경제적 부담을 줄여주는 경비 시스템이 될 것이다.

5) Hobserver 의 범용적인 사용에 대한 기대성

홉저버는 항만이 아니더라도 항공, 도시와 같은 주거 단지, 공장 단지 등 경비가 필요한 공간에서도 활용될 수 있기에 범용성이 높은 시스템이다.

□ 개발 일정

No	내용	2020 年				
		6 月	7 月	8 月	9 月	10 月
1	기능 설계 및 분석					
2	openCV, ROS 이해					
3	보드와 로봇의 제어 이해					
4	센서제어 테스트					
5	센서 결과값 처리 및 분석					
6	데모 맵 제작					
7	서버 구축, 통신					
8	OTP 개발					
9	사람 인식 알고리즘 구현					
10	주행 알고리즘 구현					
11	드론의 비행, 맵핑 알고리즘 구현					
12	추가 웹 UI					
13	시험 평가 및 테스트					

□ 팀 업무 분장

No	구분	성명	참여인원의 업무 분장
1	팀장	정현성	• ROS 터틀봇 주행 • 서버, OTP, 터틀봇 연동 • 하드웨어 구성 • TURTLEBOT 센서 제어
2	팀원	조혜영	• 서버 및 통신 구축 • 맵 이미지 영상처리 및 경로 알고리즘 제작 • 드론 이미지 영상처리 및 객체 추적,판단 알고리즘 • 터틀봇 이미지 영상처리 및 OTP 인증 알고리즘
3	팀원	박민정	• ROS 터틀봇 주행 • 서버, 터틀봇 연동 • 하드웨어 구성 • 웹 UI 제작
4	팀원	배한빈	• Drone Control • Yolo dataset 생성 및 학습 • Object Detection • OTP module 제작

[1] e-나라지표(HOME> 청단위기관> 해양경찰청> 최근 5년간 밀입국자, 밀출국자 단속 현황), 2020.09.25

[2] 노준철, 감천항서 베트남 선원 4명 사라져…"밀입국 추정",
<KBS News>, https://news.kbs.co.kr/news/view.do?ncd=4488427, 2020.07.07

[3] 울산항, 항만보안 이렇게 허술해선 안된다, 울산신문,
https://www.ulsanpress.net/news/articleView.html?idxno=354360

[4] Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao, 2020, YOLOv4: Optimal Speed and Accuracy of Object Detection