제18회 임베디드SW경진대회 개발완료보고서

[Smart Things]

□ 개발 요약

팀 명	BOOGIE

그림 1. 작품 사진

작품명	스마트 물류 시스템 Logi Boogie (로기부기)
작품설명 (요약)	- 운송장을 스캔하여 택배의 배송지와 크기에 따라 최적의 적재 순서를 계산 - 주행 로봇과 로봇팔을 이용하여 택배를 분류하고 전달 - 택배/물류 뿐 아니라 스마트 팩토리, 대형 창고 및 물품 보관 서비스 등에도 활용 가능
소스코드	https://github.com/Kkuma99/Boogie_emeddedSW_2020
시연동영상	https://youtu.be/anbjpytdbVc

□ 개발 개요

○ 개발 작품 개요

- 물류센터에서 차량에 물건 적재 시 로봇에 탑재된 프로그램을 통해 크기와 배송지에 따라 최적의 적재 순서를 빠르게 계산하는 스마트 물류 시스템을 고안하였다.
- 이 시스템을 통해 물류 업무의 노동/시간적 효율을 높일 수 있으며, 차량의 빈 공간을 최소화하는 방향으로 계산하기 때문에 한 번에 더 많은 물류를 배송할 수 있다.

- 시스템 프로세스

※ 주요 영상처리와 적재 순서 계산 알고리즘을 수행하는 Jetson(Jetson TX2)과 로봇팔을 직접 제어하는 Rpi(Raspberry Pi), 서버 역할을 하고 Rpi에 명령을 전달하는 Host PC로 두 주요 보드와 마스터 PC를 구분한다.

1. Jetson에서 영상처리(카메라1)를 통해 컨베이어 벨트 위를 지나는 상자의 바코드를 스캔하여 물류 정보(배송 지역, 상자 번호)를 얻고, 상자의 크기를 측정하여 지역별로 분류하여 저장한다.
   1-1. ROS 통신을 이용하여 상자 정보를 Jetson에서 서버에 전송한다.
   1-2. HostPC가 서버에서 상자 정보를 받아와 저장하고, 영상처리(카메라2)를 통해 상자가 로봇팔 앞에 온 것을 인식하면 전송받은 상자의 배송지 정보에 따라 로봇팔을 이용하여 지역별로 분류한다.
2. 적재할 상자들에 대한 정보 수집과 분류가 끝나면, 저장된 물류 정보와 크기 정보를 이용하여 모든 상자에 대해 최적의 적재 순서를 계산한다.
3. 상자를 적재해야 하는 순서대로 3D 이미지 시각화하는 동시에 해당 상자의 정보를 터미널 창에 출력한다.

○ 개발 목표

- 택배 기사들의 효율적인 업무를 위해 로봇 및 로봇팔을 이용하는 스마트 물류 시스템을 마련하는 것이 목표이다. 이에 따른 세부 목표는 다음과 같다.
  - 영상처리를 이용한 바코드 스캔
  - 영상처리를 이용한 상자 크기 측정
  - ROS 통신을 이용한 두 PC 간의 연속적인 데이터 송수신
  - 로봇팔 제어를 통한 상자 분류
  - 저장된 상자 정보로 적재 순서를 계산하는 알고리즘 개발
  - 계산된 순서대로 상자의 정보를 표시하고 3D 시각화하는 알고리즘 개발

○ 기대 효과

- 현재 물류 업무는 컨베이어 벨트를 통해 물류가 전달되면 각 차량에 해당하는 것을 사람이 직접 분류하여 차량 쪽으로 전달하는 방식이다. 이를 로봇을 이용하여 자동화함으로써 노동력을 절약할 수 있다.

- 택배 기사가 택배를 차량에 적재할 때 무작위로 적재하면 배송할 때마다 안쪽에 있는 택배를 찾은 뒤 다시 적재해야 하는 불편함을 겪고 있다. 적재하는 순서를 프로그램이 계산해줌으로써 종사자들의 업무 강도를 개선할 수 있다.

○ 개발 작품의 활용 가능성

- 택배 회사의 물류 상하차에 활용

  이 시스템의 주목적은 물류센터의 업무 효율을 높이는 것이다. 빠른 적재 순서 계산을 통하여 여러 번 상하차를 진행할 필요 없이 알고리즘의 결과에 따라 물류를 적재하면 되기 때문에 편리하고, 각 지역에 배송하는 과정에서도 시간 절약이 가능하다.

- 스마트 팩토리의 물건 분류와 재고 관리에 활용 가능

  스마트 팩토리에서 물건을 분류하거나 출고 전의 물품을 관리하는 데에 활용할 수 있다.

- 대형 창고 및 물품 보관 서비스 업체에서 활용 가능

  많은 물품을 보관해야 하는 대형 창고나 물품 보관 서비스 업체에서는 제한되어있는 규모의 공간에 최대한 많은 물품을 보관하는 것이 좋다. 크기를 고려하여 최적의 위치를 결정해주는 시스템을 여기에 적용할 수 있다.

○ 개발 작품의 추후 개선 방향

- AR Marker를 활용한 상자의 위치 인식과 로봇팔의 그리퍼 활용

  AR Marker는 물체의 축 정보를 받아 위치와 방향 등을 알 수 있는 수단이다. Ubuntu 16.04의 개발 환경에서는 AR Marker와 관련된 오픈소스를 이용할 수 없어 현재는 영상처리만을 이용하여 상자를 인식하고 있다.

  추후 Ubuntu 18.04로 업그레이드하여 AR Marker를 사용할 수 있게 된다면 로봇팔로 상자를 분류하는 것뿐만 아니라 그리퍼로 상자를 잡아 트럭까지 전달할 수 있도록 할 예정이다.

- 여러 대의 로봇 사용으로 더 효율적인 작업

  현재 로봇의 본체와 주요 보드들을 하나씩만 보유하고 있기 때문에, 상자의 정보를 얻고 분류하는 작업을 하나의 로봇에서 수행하고 있다.

  여러 대의 로봇이 있다면 각 작업을 분배하여 유동적인 운용이 가능할 것이다. 또한 로봇이 상자를 차량 앞까지 계산된 순서대로 직접 전달하여 사람의 노동량을 감소시킬 수 있을 것이다.

- 작업 속도와 컨베이어 벨트 이동 속도의 동기화

  현재 수행 환경에서는 소형 컨베이어 벨트를 수동으로 작동시키고 있으나, 현장에서는 모터로 작동하는 자동 컨베이어 벨트를 사용한다.

  사용하고 있는 소형 컨베이어 벨트를 개조하여 모터로 작동하도록 한 뒤, 카메라에서 상자를 인식하거나 로봇팔이 상자를 분류하는 등의 작업 상태를 모터의 구동 속도와 동기화할 예정이다.

- 세분화된 DB 구축과 이동 경로를 적용한 적재 순서 계산

  테스트 시에는 3개의 배송지만으로 알고리즘을 수행하였으나, 실제처럼 배송지를 더 세분화하여 Database를 구축하고, 해당 지역의 지리 정보를 적용하여 배송 차량의 이동 경로에 따라 적재 순서를 계산한다면 더 활용도 높은 기술이 될 것이다.

□ 개발 환경 설명

○ Hardware 구성

- 주요 Hardware 구성은 다음과 같다.

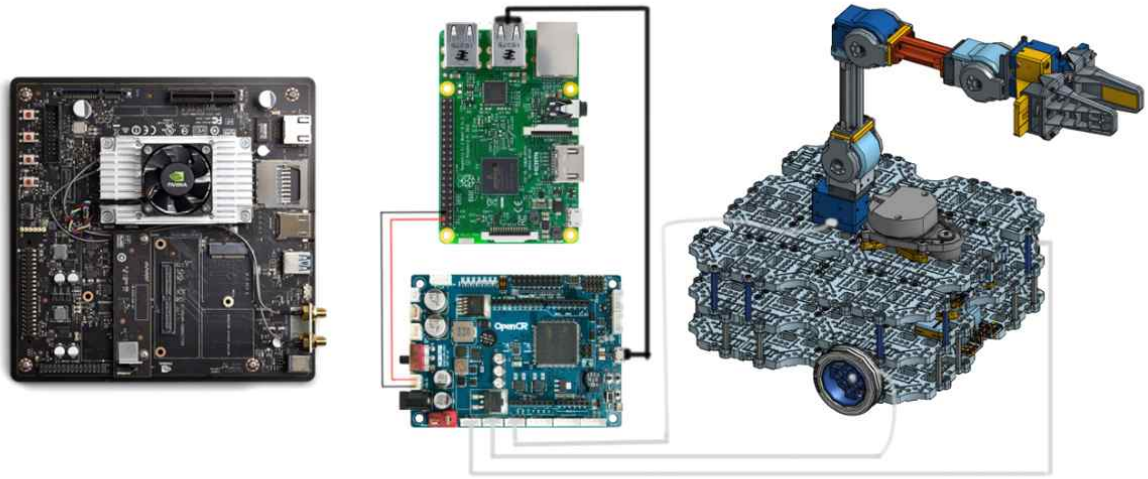

그림 2. Hardware 회로도

- Jetson TX2 (SSD로 용량 확장)

: 상자 인식, 바코드 스캔 등의 영상처리와 적재 순서 계산 알고리즘을 수행하고, 물류 정보를 관리하는 NVIDIA사의 개발 보드이다.

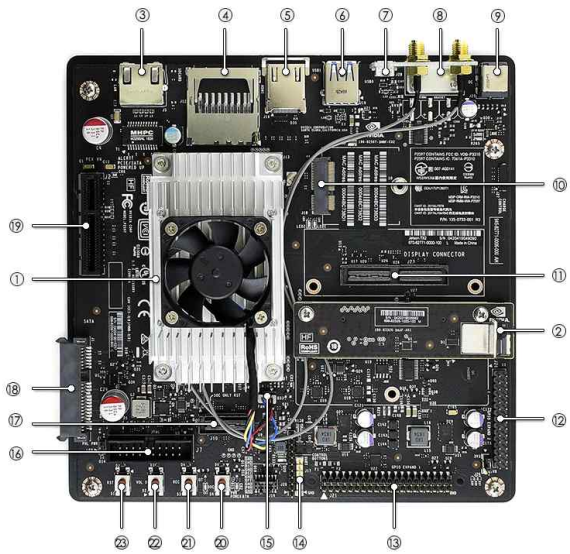

그림 3-a. Jetson TX2 (Top view)

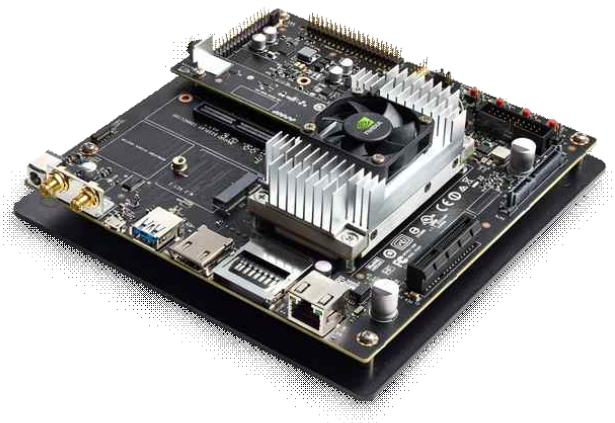

그림 3-b. Jetson TX2 (Side view)

- SSD

: 500GB의 SSD를 Jetson TX2([그림 3-a]의 ⑱번 위치)에 연결하여 용량을 증축하였다. System Root를 이동하여 메인 메모리로 사용하였다.

그림 4. SSD

- USB 3.0 허브

: Jetson TX2에는 USB 포트가 한 개이므로 카메라 및 키보드, 마우스 연결에 사용한다.

그림 5. USB 3.0 허브

- Raspberry Pi 3 B+

: 로봇팔의 제어부로 사용되는 임베디드 보드이다.

OpenCR로부터 GPIO 핀을 통해 전원을 공급받으며 USB–Micro 5Pin 케이블을 이용하여 OpenCR과 데이터를 주고받는다.

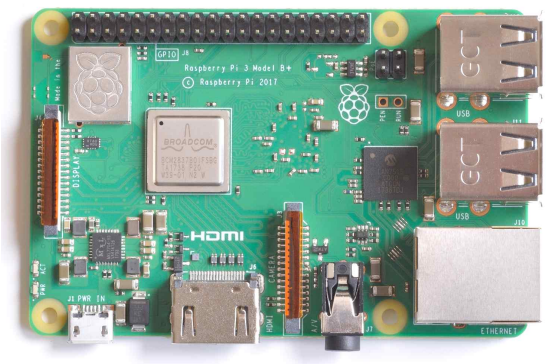

그림 6. Raspberry Pi 3 B+

- OpenCR 1.0

: 로봇의 모터(Dynamixel)를 제어하기 위한 컨트롤 보드이다. ROS를 지원하기 때문에 Open Manipulator와 Turtlebot wheel의 전반적인 제어에 사용한다.

[그림 7]의 GND Power out에 케이블을 연결하여 Rpi에 전원을 공급한다.

[그림 7]의 TTL 포트(GND VDD DATA)에 로봇 케이블을 연결하여 Open Manipulator와 Turtlebot wheel의 모터를 제어한다.

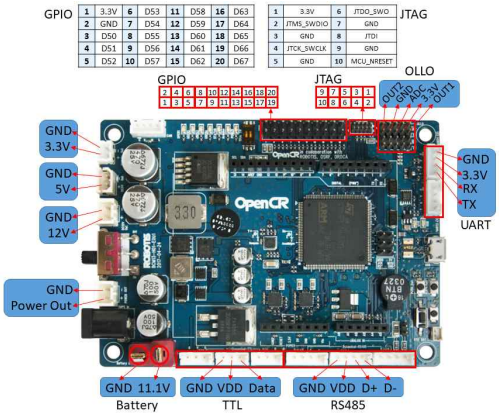

그림 7. OpenCR 1.0

- Open Manipulator-X

: ROS 기반의 소형 오픈소스 로봇으로, 여러 개의 Dynamixel로 구성된 로봇팔이다.

각 Dynamixel들은 서로 연결되어 있으며, [그림 7]의 OpenCR에서 TTL 포트에 연결하여 제어한다.

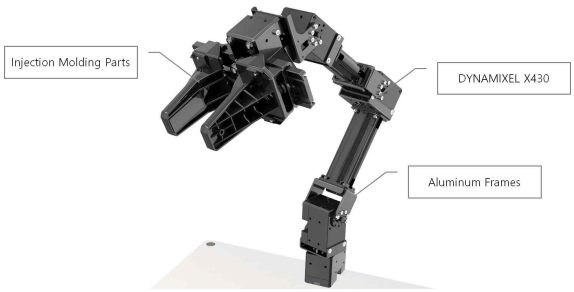

그림 8. Open Manipulator-X

- Turtlebot3 Waffle Pi

: ROS 기반의 모바일 로봇으로 Dynamixel을 이용한 wheel과 LiDAR 등이 사용되며, 주로 OpenCR과 RaspberryPi를 통해 제어한다.

기본 제공되는 형태를 개조하여 Jetson TX2를 탑재하고, Open Manipulator를 부착하여 사용하였다.

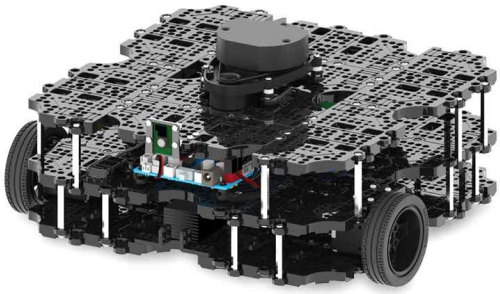

그림 9. Turtlebot3 Waffle Pi

- 웹 카메라

: USB 포트에 연결하여 영상처리에 사용한다.

총 두 대중 한 대는 Jetson TX2에 연결하여 상자 인식, 바코드 스캔, 크기 측정에 사용하고, 다른 한 대는 Host PC에 연결하여 로봇팔의 동작을 위한 상자 인식에 사용한다.

그림 10. 웹 카메라

- 노트북(Laptop)

: Host PC로 사용된다. ROS의 서버 역할을 하며, Raspberry Pi에 원격접속하여 Turtlebot을 제어한다.

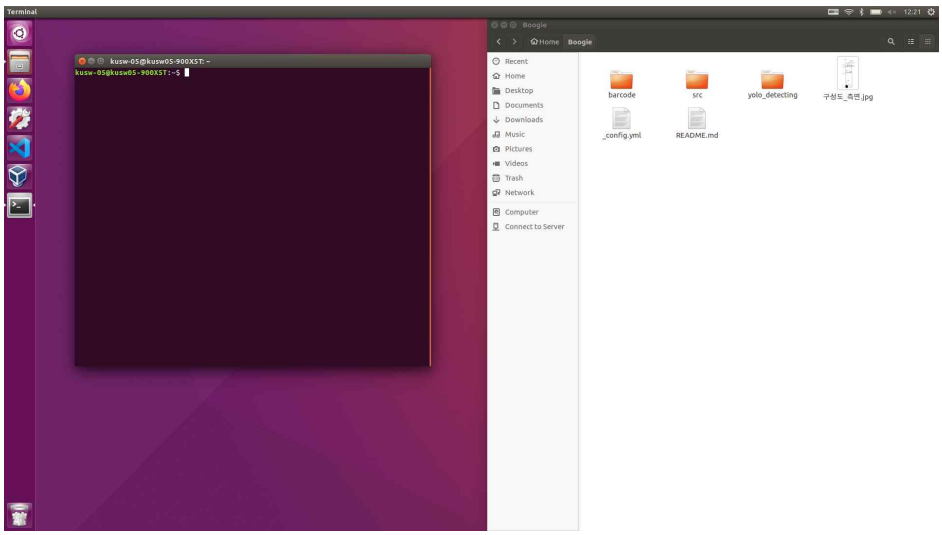

그림 11. Host PC 화면

○ Software 구성

- 주요 Software 구성은 다음과 같다.

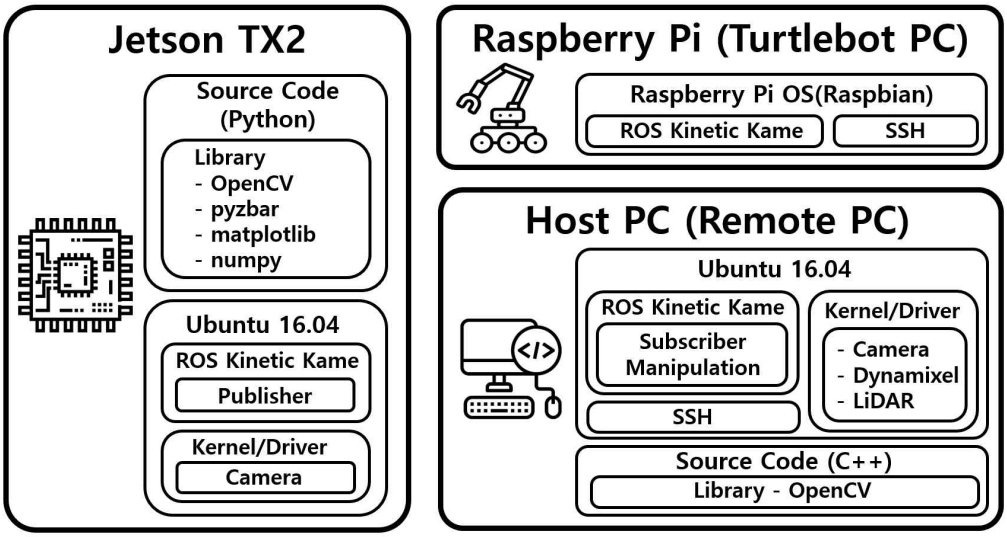

그림 12. Software 구성도

- 사용한 Software의 종류

  - Ubuntu 16.04 LTS: 리눅스 커널을 기반으로 한 리눅스 배포판이다. ROS Kinetic 버전을 사용하는 데에 호환성이 좋고 오픈소스가 많이 공개되어 있어 용이하다.
  - ROS Kinetic Kame: ROS는 로봇 미들웨어로 운영체제는 아니지만, 이기종 컴퓨터 클러스터용으로 설계된 서비스를 제공한다. 실시간으로 코드를 통합할 수 있다는 장점이 있다.

    ROS Kinetic Kame은 ROS의 10번째 배포판이며 2016년에 발매되었고 주요 ROS 클라이언트 라이브러리는 주로 대규모 오픈 소스 소프트웨어 종속성 모음에 의존하기 때문에 Unix 계열 시스템에 맞춰져 있어 Ubuntu 16.04 릴리즈 버전을 사용한다.
  - NVIDIA Jetpack 3.4: NVIDIA에서 제공하는 SDK로, AI 응용 프로그램을 구축할 때 사용하는 포괄적인 솔루션이다. 이를 이용하면 주로 사용되는 라이브러리와 API, 샘플 및 문서를 한 번에 간편히 설치할 수 있다.
  - Python: 인터프리터 언어로 다양한 오픈소스와 라이브러리들이 존재하기 때문에 이를 활용하기 위해 Python을 사용한다. Jetson TX2에서 OpenCV, pyzbar, matplotlib 등의 라이브러리를 활용하였다.
  - C++: 클래스를 사용하는 객체 지향 언어로 Host PC의 메인 소스 코드에서 사용한다.
  - OpenCV 3.4.6: Open Source Computer Vision의 약자로 다양한 영상/동영상 처리에 사용할 수 있는 오픈소스 라이브러리이다. 상자를 인식하고 크기를 측정하는 데에 사용한다.
  - matplotlib: 파이썬 라이브러리 중에서 플롯을 그릴 때 주로 사용되는 2D, 3D 플로팅 패키지 모듈이고 현재 기준으로 3.3.2가 최신 릴리즈 버전이다.
  - pyzbar: 파이썬 라이브러리로 zbar 라이브러리를 이용하여 1차원 바코드 및 QR 코드를 읽는다. Python 2.7 및 Python 3.4 ~ 3.6에서 사용할 수 있다.
  - numpy: 배열 작업에 사용되는 Python 라이브러리로 기존 Python에서의 리스트보다 최대 50배 빠른 배열 객체를 제공할 수 있다.

    numpy는 오픈소스를 제공하며 https://github.com/numpy/numpy에 공개되어 있다.

- SSH: Secure Shell Protocol, 네트워크 프로토콜 중 하나로 컴퓨터와 컴퓨터가 인터넷과 같은 Public Network를 통해 서로 통신을 할 때 안전하게 통신하기 위해 사용하는 프로토콜이다.
  대표적으로 데이터 전송 혹은 원격 제어를 할 때 사용한다.

- 각 PC(보드)에서 Software의 전반적인 역할
  - Host PC: ROS에서 master로써, roscore를 실행하여 서버로 이용한다. Turtlebot 및 Open Manipulator를 제어하는 main 코드는 C++(.cpp 파일)로 작성하였으며, GCC를 통해 Cross Compile한다.
  - Raspberry Pi: SSH를 통해 Host PC에서 원격접속하여 사용한다. Host PC에서 실행되는 main 코드의 명령을 수행한다. OpenCR에 연결하여 Dynamixel을 제어한다.
  - Jetson TX2: OpenCV를 사용하여 바코드 스캔, 상자 크기 측정 등의 전반적인 영상처리와 적재 순서 계산 알고리즘을 수행한다. ROS 통신을 이용하여 Publisher로써 Subscriber인 Host PC에 데이터를 전송한다. 코드는 Python(.py 파일)으로 작성하였다.

○ Software 설계도
- State Diagram은 다음과 같다.

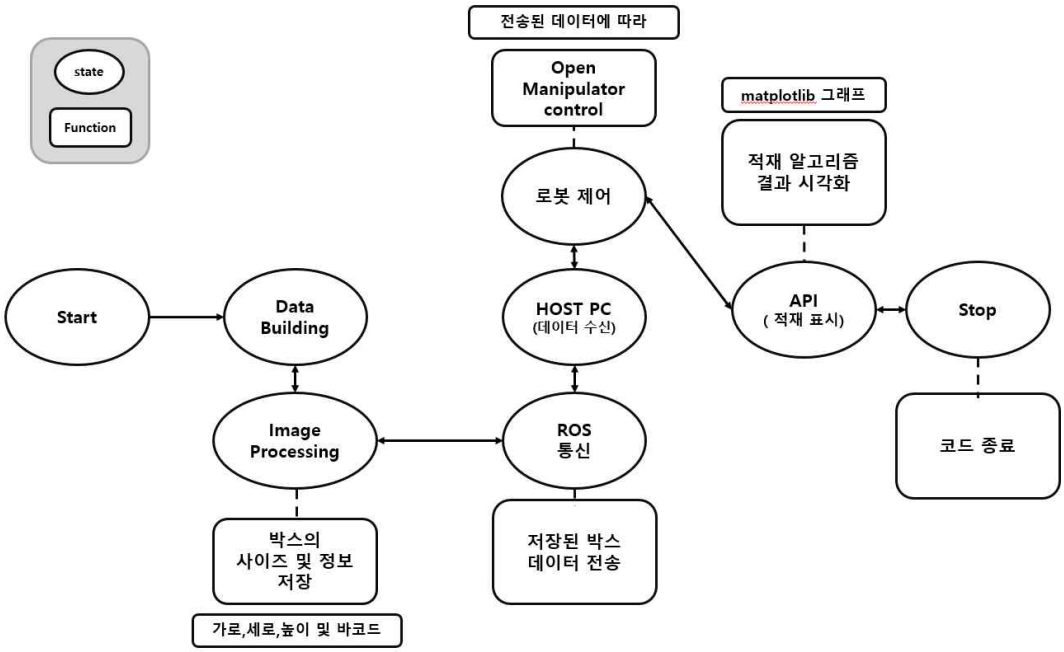

그림 13. State Diagram

- state로는 크게 Start(시작) 및 Data Building(데이터 처리), Stop(종료)으로 볼 수 있다. 처음에 대기 상태에서 데이터가 들어오기 시작하면 Start하여 Data Building 상태로 들어가게 된다.
- Image Processing을 하는 동시에 ROS 통신으로 데이터가 전달되고 Control system을 통하여 로봇을 제어한다.
- 이후 종료 state에 들어간 것이 확인되면 Open Manipulator는 구동을 멈추고 적재 알고리즘에 따라 적재된 상자를 시각화하여 사용자에게 공유한다.

- 주요 소스 파일의 Flow Chart는 다음과 같다.

• smart_logi_system_jetson.py

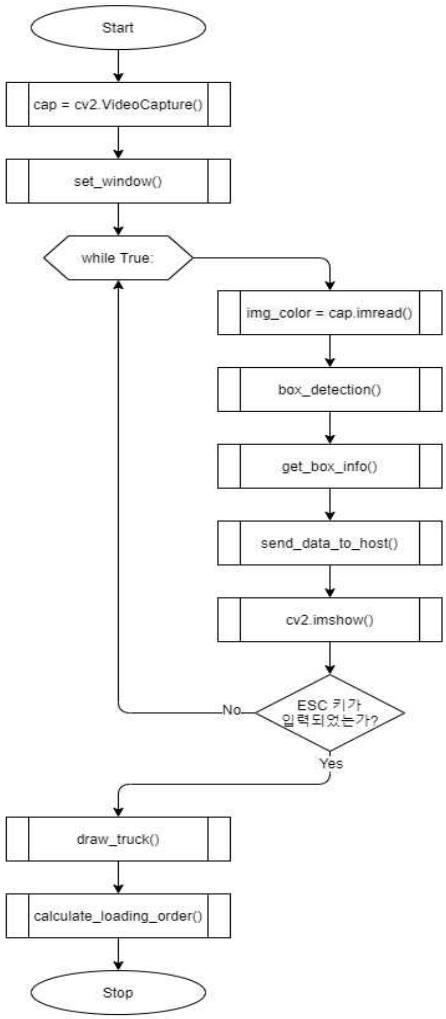

그림 14. Flow Chart (1)

• main.cpp

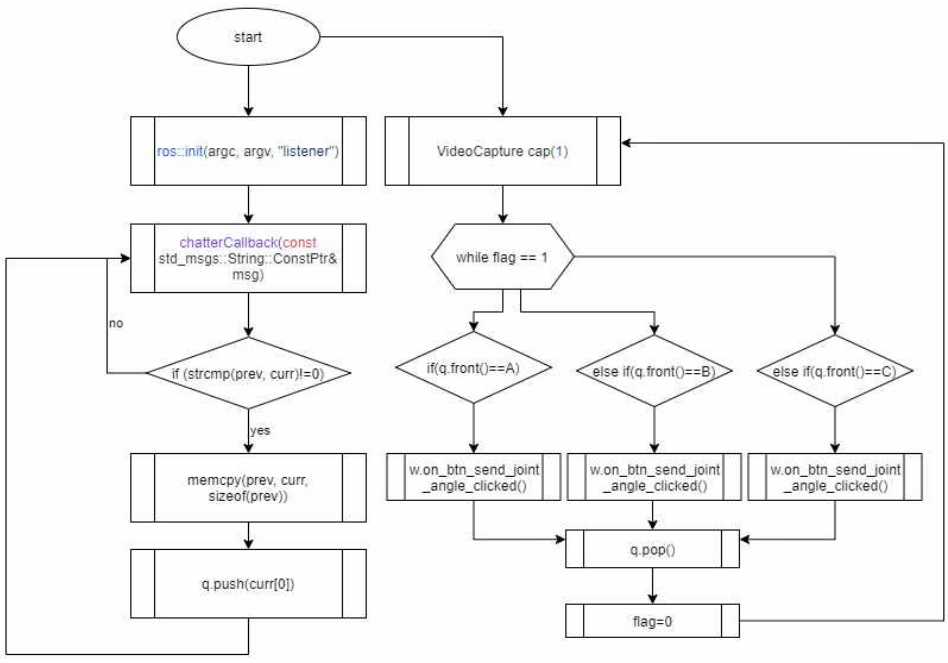

그림 15. Flow Chart (2)

○ 프로그램 사용법

- Jetson TX2

아래 명령을 통해 python3으로 python 프로그램을 실행한다. 하나의 .py 파일로 Jetson 에서 수행해야 하는 모든 작업을 수행할 수 있다.

```
$ python3 smart_logi_system_jetson.py
```

```
nvidia@tegra-ubuntu:~$ python3 smart_logi_system_jetson.py

[INFO] [1602325557.997203]: XXX
[INFO] [1602325558.208215]: XXX
[INFO] [1602325558.391593]: XXX
```

그림 16-a. smart_logi_system_jetson.py 실행

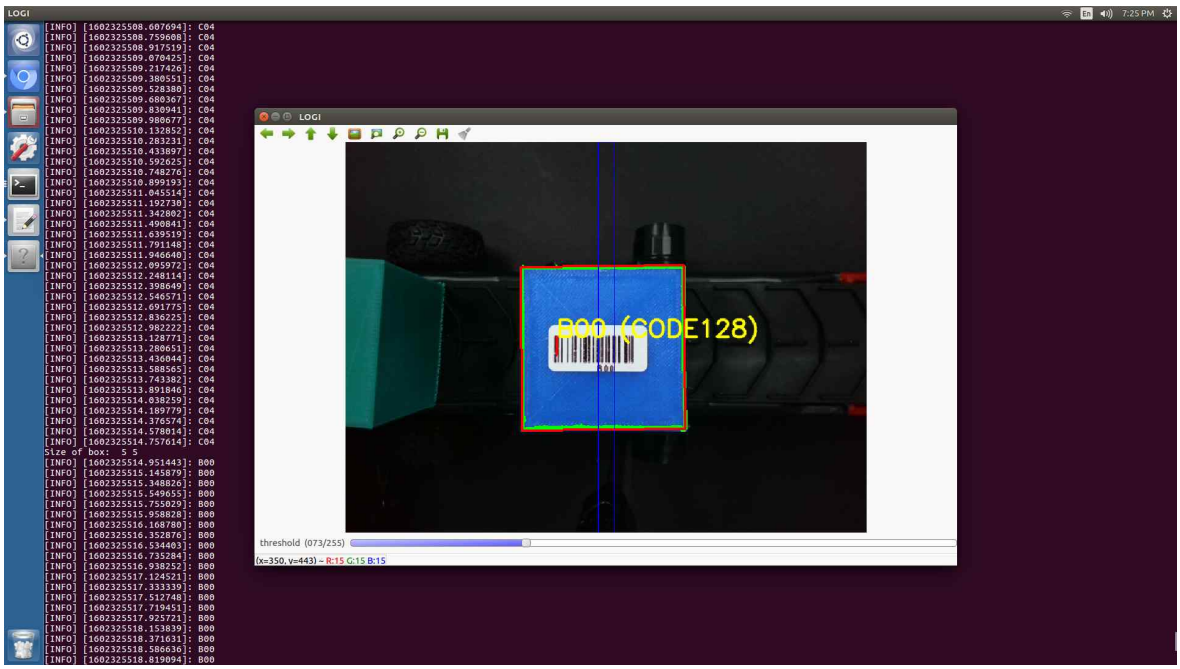

그림 16-b. smart_logi_system_jetson.py 실행 중 바코드를 스캔 중인 모습

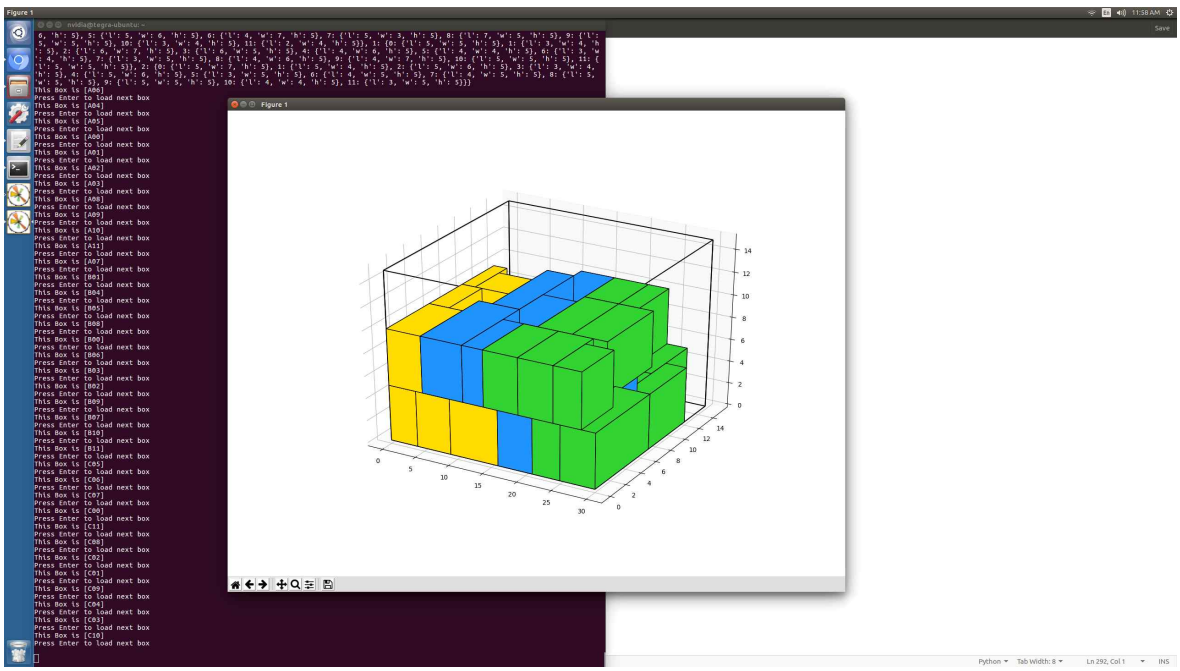

그림 16-c. smart_logi_system_jetson.py 실행 중 상자 적재가 완료된 모습

- Host PC

• roscore 실행

아래 명령을 통해 ROS의 node들이 실행되고 서로 통신할 수 있도록 roscore를 실행시킨다. 이를 통해 Host PC는 master로써 중심 서버 역할을 한다.

```
$ roscore
```

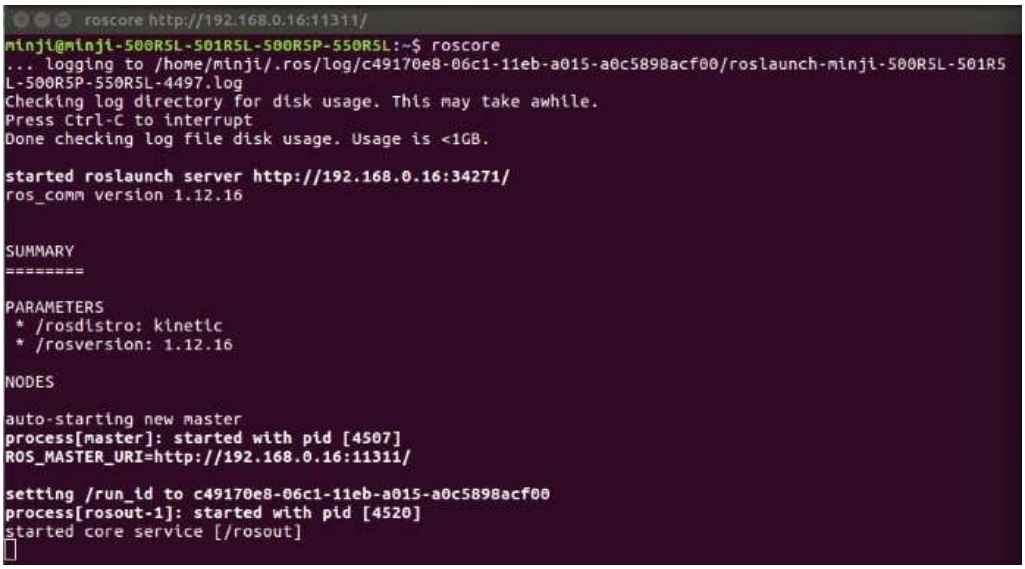

그림 17. roscore 실행 화면

• manipulator bringup 실행

아래 명령을 통해 open-manipulator를 제어할 수 있는 기본적인 조작 기능이 담긴 소스코드를 launch한다.

```
$ roslaunch turtlebot3_manipulation_bringup turtlebot3_manipulation_bringup.launch
```

그림 18. manipulator bringup 실행 화면

- MoveIt 실행

MoveIt은 로봇의 조작을 위해 사용되는 소프트웨어이며 모션계획, 조작, 3d 인식, 운동학, 제어 및 내비게이션을 통합하여 로봇에서 사용된다. 매니퓰레이터를 작동시키기 위해 실행한다.

```
$ roslaunch turtlebot3_manipulation_moveit_config move_group.launch
```

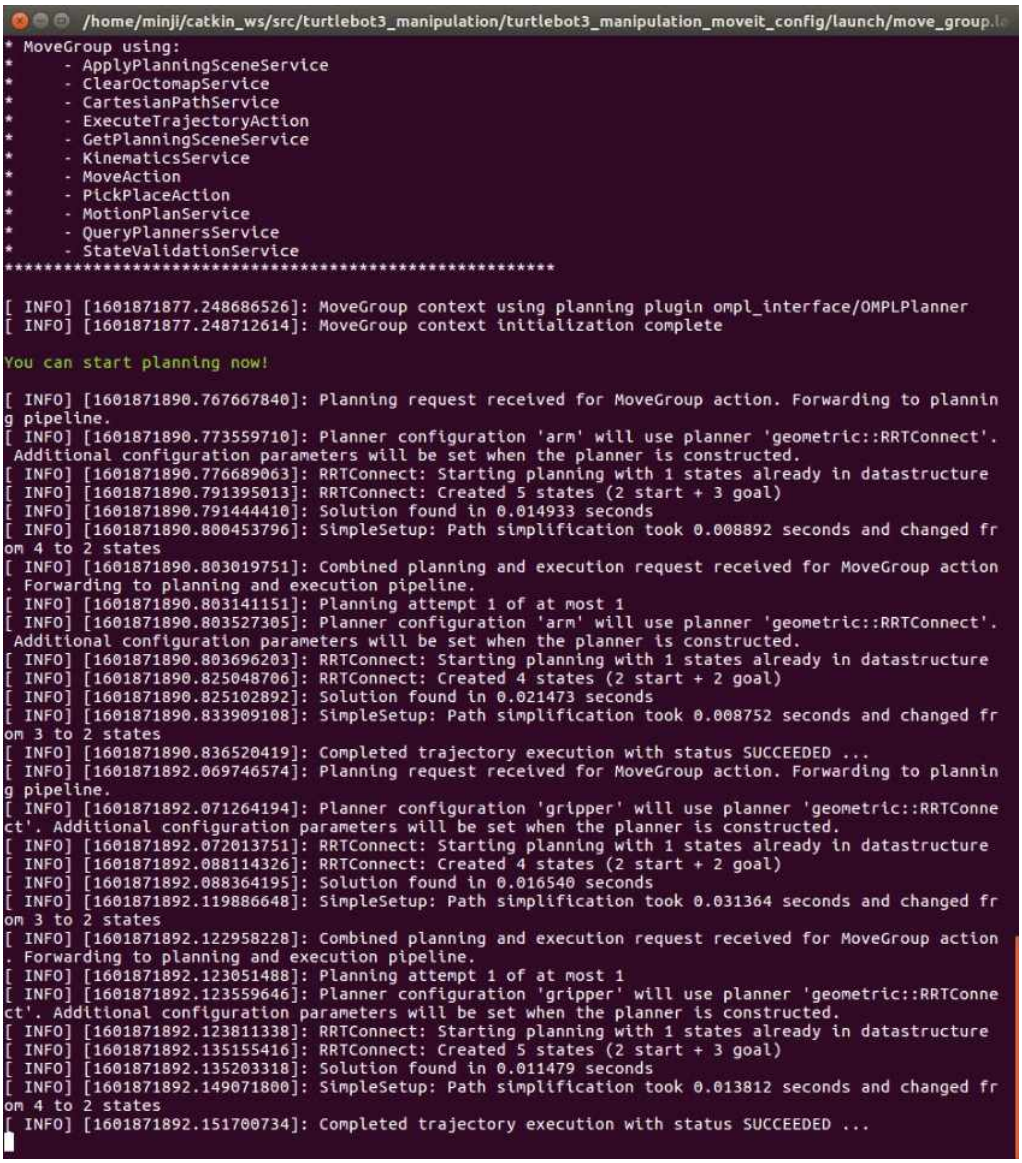

그림 19. MoveIt 실행 화면

- 로봇팔 제어 코드 실행

아래 명령어를 통하여 터틀봇의 오픈 매니퓰레이터를 제어하는 메인 코드를 실행한다. Jetson으로부터 전달되는 데이터를 받아서 오픈 매니퓰레이터의 동작을 제어한다.

```
$ roslaunch turtlebot3_manipulation_gui turtlebot3_manipulation_gui.launch
```

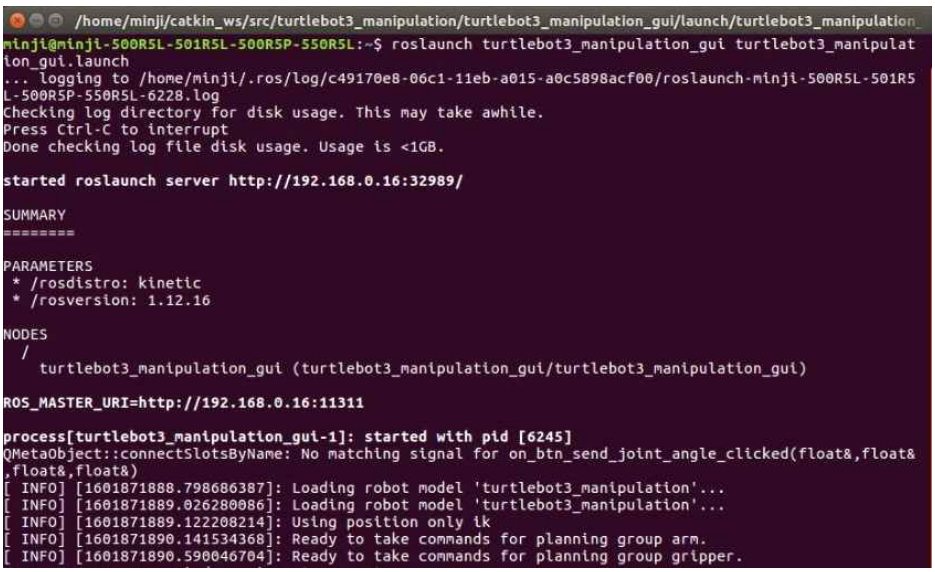

그림 20. 로봇팔 제어 코드 실행 화면

- Raspberry Pi

- Turtlebot bringup 실행

터틀봇 기본 기능을 시작하기 위한 roslaunch 스크립트를 실행한다.

```
$ roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

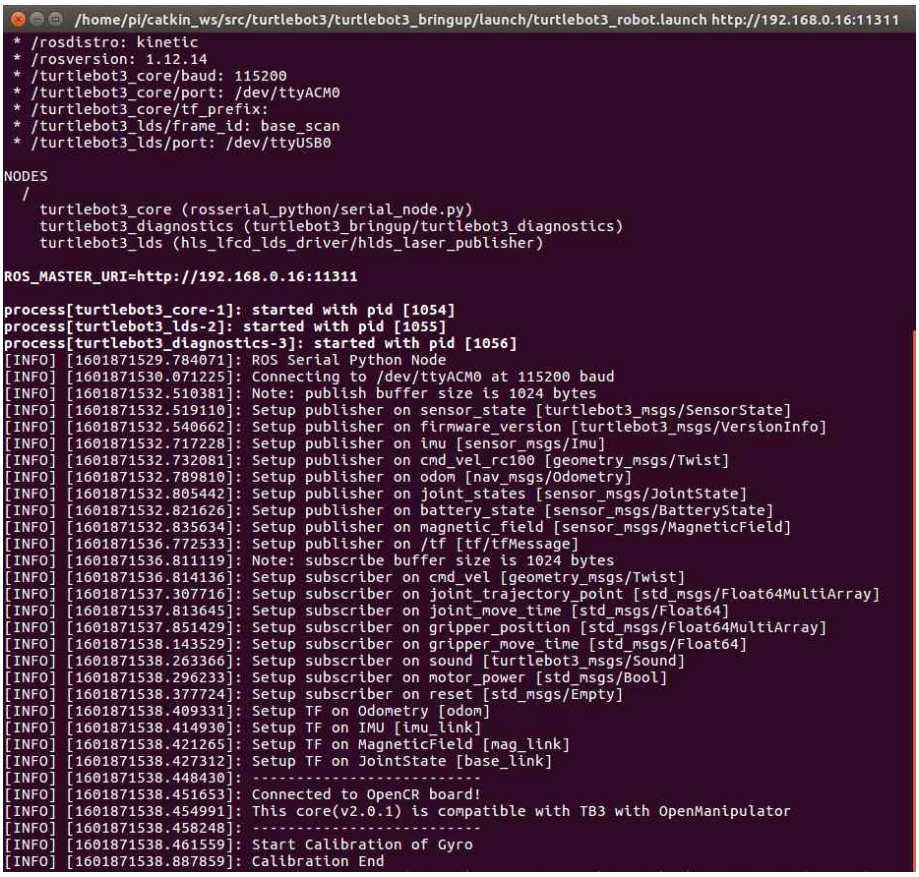

그림 21. Rpi Turtlebot bringup 실행 화면

○ Software 기능

- 적재 순서 계산 알고리즘(smart_logi_system_jetson.py)

: Jetson TX2 상에서 사용되는 알고리즘이다. 상자의 정보를 수집하고 적재 순서를 계산하는 역할을 한다. 이 알고리즘이 수행하는 작업은 아래와 같다.

① 먼저, 배송해야 하는 배송지의 개수와 트럭의 크기를 상수로 지정하고, 바코드 데이터를 담을 변수를 초기화한다.

② VideoCapture()를 통해 웹 카메라의 화면을 받아온 뒤, 프레임의 너비와 높이, 프레임 속도를 조절한다.

③ set_window() 함수를 호출하여 카메라 화면을 display할 준비를 한다. window의 이름을 지정하고, 뒤에서 검출할 컨투어의 threshold를 쉽게 변경할 수 있도록 트랙바를 추가한다. 이 threshold는 작업을 수행하는 환경의 조도 상황에 따라 조정하여 사용한다.

④ while문에 진입하여 지속적인 영상처리를 시작한다. while문 내부에서는 상자 인식, 상자 정보 수집, 바코드 데이터 전송 등의 작업을 수행한다.

우선 카메라의 프레임을 읽어와 img_color 객체에 저장한다. box_detection() 함수를 호출하여 현재 프레임에서 상자를 인식한다. box_detection()에서 수행하는 작업은 다음과 같다.

㉠ 상자를 검출할 이미지 img_color를 인수로 받아온다.

㉡ 노이즈를 제거하기 위해 가우시안 블러를 적용한 뒤, 그레이스케일로 변환한다.

㉢ 트랙바에 설정되어 있는 threshold값을 받아오고, 이를 그레이스케일 이미지에 적용하여 바이너리 이미지를 생성한다.

㉣ findContours()를 통해 해당 이미지에서 컨투어를 검출한다.

㉤ 이미지에서 가장 큰 면적을 차지하는 컨투어 객체가 상자에 해당하므로, 검출된 모든 컨투어에 대해서 for문을 돌며 상자의 컨투어를 찾아 인덱스를 저장한다.

㉥ 원본 이미지에 검출된 상자의 컨투어를 초록색으로 표시한다.

㉦ 상자의 컨투어를 둘러싸는 가장 작은 사각형을 그려 빨간색으로 표시한다. 이 때 상자는 회전되어 있어도 무방하다. 이 사각형의 꼭짓점 좌표를 box에 저장한다.

㉧ 후에 이미지의 display와 바코드 스캔, 상자의 크기 정보 추출을 위하여 필요한 값들을 반환한다.

⑤ 다음으로 get_box_info() 함수를 호출하여 상자의 정보를 수집한다. get_box_info()에서 수행하는 작업은 다음과 같다.

㉠ 먼저, 바코드 스캔 함수를 사용하기 위해 입력 이미지를 그레이스케일로 변환한다.

㉡ 상자의 정확한 크기 측정을 위해 상자가 화면의 중앙에 왔을 때 크기를 측정해야 하므로, 이 중앙 영역을 구분하기 쉽도록 파란색으로 표시한다. 상자는 화면에서 컨베이어 벨트를 따라 x축 방향으로 이동하므로, 상자 중심의 x좌표를 계산하여 변수 center에 저장한다.

㉢ if문을 사용하여 center의 값이 전체 화면 중심의 ±10 픽셀 이내에 위치할 때에만 작업을 수행하도록 한다.

㉣ if문에 진입하면 pyzbar 라이브러리의 decode() 함수를 통해 바코드를 인식한다. 바코드가 성공적으로 인식되면 decoded에 바코드 객체가 생성된다.

㉤ for문에 진입하면 바코드 영역의 좌표를 x, y, w, h에 저장하고 화면에 사각형으

로 표시한다. 바코드 객체에서 data 멤버의 decode() 함수를 실행하여 인식된 바코드가 갖고 있는 정보를 얻어 barcode_data에 저장하고, type 멤버에 접근하여 그 값을 barcode_type에 저장한다. 그리고 barcode_data와 barcode_type을 이미지 위에 삽입한다.

㉤ 바코드 정보를 처리한 뒤에는 상자의 크기를 측정한다. 우선 화면에서 상자가 차지하는 픽셀 단위 크기를 구한다. 이는 앞에서 box에 저장된 꼭짓점 좌표를 이용한다. 상자가 회전되어 있는 경우도 고려하여 피타고라스 정리를 이용해 box_l_pixel과 box_w_pixel 값을 계산한다. 그리고 상자의 실제 크기와 픽셀 단위 크기의 비율을 구하여 box_l과 box_w 값을 계산한다. 우리의 수행 환경에서 비율은 1:40.8 이었으며, 이는 카메라의 해상도나 상자와 카메라 사이의 거리에 따라 바뀌므로 환경에 따라 조정해 주어야 한다.

㉥ 상자 크기를 측정한 뒤에는 앞에서 수집한 바코드 정보와 상자 크기를 데이터화해야 한다. 자료형이 딕셔너리인 변수 inputBox에 해당 상자 정보를 저장한다. inputBox는 이중 딕셔너리이며, 1중 key는 배송지, 2중 key는 각 상자의 인덱스이다. 바코드 정보가 '알파벳 대문자(배송지), 숫자(상자인덱스 십의자리), 숫자(상자인덱스 일의자리)'의 형태(ex. A00, A01, … , B00, B01, …)로 되어 있으므로 문자를 아스키 코드로 변환해주는 함수 ord()를 사용하여 바코드 데이터의 맨 앞글자를 추출해 inputBox의 key 값으로 사용한다. if문을 사용하여 상자의 정보가 처음 들어올 때만 inputBox에 저장하고, 상자의 개수를 증가시킨다.

㉦ 후에 바코드 데이터를 Host PC에 전송하기 위해 반환한다.

⑥ send_data_to_host() 함수를 호출하여 바코드 데이터를 Host PC에 전송한다. 이 함수는 ROS 통신 중 단방향 통신인 메시지 통신을 수행한다. 데이터를 전송하는 publisher 역할이며, 문자열 데이터를 전송한다.

⑦ imshow()를 통해 컨투어, 바코드 등의 정보가 삽입된 이미지를 화면에 띄운다. waitKey()를 사용하여 키보드로 ESC 키가 입력되면 while문에서 탈출한다. ESC 키는 모든 상자의 입력이 끝났을 때 사용한다.

⑧ while문에서 탈출한 뒤에는 OpenCV, 즉 영상처리에 사용된 모든 메모리를 해제한다.

⑨ matplotlib를 이용하여 트럭의 내부 모습을 시각화하기 위해 plot의 형태를 3D로, 양상을 auto로 설정한다. colors는 각 배송지별 상자의 색이다.

⑩ draw_truck() 함수를 호출하여 트럭의 전체 프레임을 생성한다. numpy의 meshgrid() 함수를 사용하여 격자를 생성하고, 검정색 선으로 표현한다.

⑪ calculate_loading_order() 함수를 호출하여 입력된 모든 상자들에 대해 최적의 적재 순서를 계산한다. calculate_loading_order()에서 수행하는 작업은 다음과 같다. (트럭의 상태를 나타내는 truck 변수는 1㎥ 단위로, 빈 공간이면 0, 적재되어 있는 상자의 배송지가 A이면 3, B이면 4, C이면 5로 표현한다. 2는 실제로는 빈 공간이긴 하나, 상자를 가장 효율적으로 적재하기 위해 알고리즘에서는 비어 있지 않은 것으로 가정한다. 2로 채워진 영역은 매우 작으므로, 실제로 상자를 적재하는 데에는 어려움이 없다.)

㉠ 각 상자가 적재되었는지 여부를 판단하기 위한 리스트 check을 생성한다.

㉡ for문을 사용하여 각 배송지 단위로 반복하며, 반복인자가 작은 숫자일수록 트럭의 안쪽에 적재된다. 가장 아래에서부터 적재하므로 다음 지역으로 넘어갈 때마다 floor를 0으로 초기화하며, 각 지역별 상자의 개수를 고려하여 적재할 수 있는 공간을 제한하기 위해 sum_num_box에 상자의 개수를 더한다.

㉣ 상자를 적재할 층수를 고려해야 한다. 길이 방향(x축 방향)으로 상자가 가장 적게 적재되어 있는 지점을 찾기 위해 빈 공간(0)이 나올 때까지 count_L을 증가시켜 가며 측정하는 것을 너비 방향(y축 방향)으로 반복하여 진행한다. 가장 적게 적재된 곳에 상자가 적재된 공간의 길이를 endOfL에 저장한다. endOfL이 해당 배송지역 그룹에 할당된 길이 이상이거나, 트럭의 길이 이상이면 다음 층에 적재를 시작한다. 단, 트럭의 높이 이상이 된 경우에는 다시 가장 아래층에 적재한다.

㉤ 상자를 적재할 최적의 위치를 찾는다. 길이 방향으로 진행하며 빈 공간이 나오면 측정 모드로 전환하고 빈 공간의 너비를 측정한다. 이 공간에 상자를 적재하기 위함이다. 이 때 x, y, z 좌표를 저장한다. 좌표는 직육면체 기준으로 항상 원점과 가장 가까운 지점으로 정한다.

㉥ 적재할 좌표와 빈 공간의 너비를 측정했으면 이 공간에 들어갈 최적의 상자를 선별하여 상자의 인덱스를 저장한다. 이 때 5가지의 조건을 모두 만족해야 한다.

1. 아직 적재하지 않은 상자이고, 너비가 count_W 이하여야 한다.
2. 빈 공간에 들어갈 수 있는 상자 중 최대 너비를 가진 상자여야 한다.
3. 해당 위치에 상자를 적재했을 때 트럭 높이를 넘지 않아야 한다.
4. 해당 위치에 상자를 적재했을 때 트럭 길이를 넘지 않아야 한다.
5. 적재할 상자의 아래가 빈 공간이 아니어야 한다.

㉦ 선별된 상자를 해당 위치에 적재한다(else). truck을 배송지역에 따라 3, 4, 또는 5로 채운다. 적재에 성공하면 적재 완료된 상자의 수를 갱신하고 check에 표시한다. 그리고 해당 상자를 matplotlib를 이용하여 각 배송지역에 해당하는 색으로 시각화하면서 사용자가 상자를 올바르게 찾아 적재할 수 있도록 화면에 상자의 정보를 출력한다. Enter 키를 입력할 때마다 다음 상자를 적재하고 마찬가지로 시각화하는 것을 반복한다.

만약 해당 위치에 적재할 수 있는 상자가 없는 경우(if) 그 공간을 2로 채운다.

㉧ 적재가 모두 끝나면 종료 메시지를 출력하고 마지막으로 60초간 적재가 완료된 트럭의 모습을 화면에 띄워준다.

- 로봇팔 제어 알고리즘 (main.cpp)

① ROS 메시지 통신을 이용하여 Jetson으로부터 바코드 데이터를 전송받는다.

② 이때 데이터가 계속해서 들어오는 경우 중복을 방지하기 위해서 중복 검사를 한다. 주소지의 형태로 예를 들어 A00이면 'A','0','0' 이런 식으로 string 데이터가 들어오게 되는데 각각의 문자열을 원래 가지고 있던 배열과 비교한다. 만약 각각의 문자열 중 하나라도 다르다면 다른 상자이므로 그 데이터를 배열에 저장하고 Queue에 해당 데이터를 push한다. 즉 Queue에서는 중복되는 데이터 없이 배송지를 저장하여 추후 로봇팔 제어에 사용하게 된다.

③ 기존에 GUI에서의 입력을 통하여 로봇팔을 제어하던 알고리즘을 수정하여 지정된 위치로 로봇팔을 제어할 수 있도록 함수를 사용한다. 원본 함수의 경우에는 GUI 창이 틀어지면서 home pose / init pose / gripper open / gripper close 외에 나머지는 x 좌표, y좌표, z좌표의 데이터 혹은 각각의 조인트의 angle값을 숫자로 입력하여 그 값을 전달해주는 방식으로 로봇팔을 조종한다. 현재 알고리즘에서는 해당 함수를 변경하여 GUI에서 입력하는 것이 아닌 코드상에서 미리 인자를 지정하여 사용하는 형식으로 변경하였다.

각 배송지별로 분류하기 위해 로봇팔이 이동해야 하는 위치를 사전에 확인하여 이를 배열로 저장하고, 그 값을 배송지 조건에 따라 인자로 전달하여 함수가 실행되도록 한다. 예를 들어 float a[4] = {0.550, 0.500, -0.100, 0.500}; 와 같이 저장해둔 후, 함수 실행 시 각각의 조인트를 해당 배열값으로 조정한다.

④ subscriber를 통하여 주소지 데이터를 연속적으로 전달받고 중복되지 않는 값이 Queue에 쌓이는 동안에, 로봇팔도 연속적으로 움직이는 것이 아니라 상자가 로봇팔이 작업할 수 있는 위치에 도달하였을 때만 분류 작업을 수행한다.

1차적으로 상자가 Jetson에서 인식되고, 추출한 상자 정보를 서버로 전송하면 main.cpp에서 정보를 받아 저장해둔다. 2차적으로 상자가 로봇팔 앞에 있다는 것이 인식되면 그 순간에 로봇팔을 움직이도록 하였다. 여기서도 OpenCV를 이용하였으며, 상자의 컨투어가 특정 좌표에 위치했을 때 플래그를 바꾸고 로봇팔 제어 함수를 호출한다.

○ 개발환경 (언어, Tool, 사용시스템 등)

- 개발에 사용된 언어
  - C++ : 리눅스 환경에서 사용되어 gcc 컴파일러를 사용
  - Python

- 각 PC(보드)의 운영체제
  - Host PC: Ubuntu 16.04
  - Raspberry Pi(Turtlebot PC): Raspberry Pi OS(Raspbian)
  - Jetson TX2: Ubuntu 16.04

- 각 PC(보드)에서 개발에 사용된 주요 툴
  - Host PC: ROS Kinetic, GCC, OpenCV 3.4.6, SSH
  - Raspberry Pi(Turtlebot PC): ROS Kinetic, SSH
  - Jetson TX2: ROS Kinetic, NVIDIA JetPack 3.4, OpenCV 3.4.6, pyzbar, matplotlib

- 기타 사용한 툴
  - 코드 에디터: Visual Studio code, Vim, Gedit
  - 프로젝트 관리: Github

□ 개발 프로그램 설명

○ 파일 구성

- smart_logi_system_jetson.py
  : Jetson TX2에서 상자 인식, 바코드 스캔, 상자 크기 측정 등의 영상처리와 ROS 메시지 통신을 이용한 바코드 정보 전송, 상자 적재 순서 계산 및 시각화 등 핵심 알고리즘을 수행하는 파일이다. 본 파일 하나만으로 위의 모든 작업을 수행할 수 있다.

- turtlebot_manipulation_bringup.launch
  : Raspberry Pi에서 터틀봇 로봇을 구동하기 위하여 실행하는 launch 파일로 메인 코드인 turtlebot3_manipulation_bringup.cpp를 실행시킨다.

- turtlebot_manipulation_gui.launch
  : 마스터에서 터틀봇 로봇팔을 구동하기 위하여 사용하는 파일로 main.cpp를 실행시킨다.

○ 함수별 기능

- smart_logi_system_jetson.py

(1) 자체적으로 제작한 함수

- send_data_to_host(boxData)
  : ROS 통신으로 Host PC에 boxData를 전송(Subscribe)하는 함수이다.
- box_detection(img_color)
  : img_color에서 상자를 인식하는 함수이다.
- get_box_info(img_color, gray_barcode, box, barcode_data, inputBox, NUM_BOX)
  : 인식된 상자의 바코드 정보, 크기 정보 등을 추출하는 함수이다.
- calculate_loading_order(NUM_LOCAL, NUM_BOX, TRUCK_L, TRUCK_W, TRUCK_H, inputBox, truck)
  : inputBox에 저장된 상자 데이터를 이용하여 최적의 적재 순서를 계산하는 함수이다.

(2) OpenCV 내장 함수

- cv2.VideoCapture()
  : 전달된 인자(카메라, 동영상 파일 등)로부터 비디오 객체를 생성한다.
- read()
  : 비디오 객체로부터 frame을 읽어온다.
- cv2.GaussianBlur()
  : 입력 이미지에 가우시안 블러를 적용한다.
- cv2.cvtColor()
  : 입력 이미지의 색상 모드를 지정하는 flag에 따라 변환한다.
  우리는 cv2.COLOR_BGR2GRAY를 사용하여 컬러 이미지를 그레이스케일 이미지로 변환하였다.
- cv2.threshold()
  : 임계값(threshold)을 지정하여 입력 이미지를 바이너리 이미지로 변환한다. 우리는

트랙바에서 설정한 임계값에 따라 모든 픽셀들이 0 또는 255의 값을 갖도록 하였다.

- cv2.findContours()
  : 입력 이미지(바이너리)에서 엣지를 검출하여 그 정보를 계층 또는 리스트로 저장한다. 우리는 리스트로 저장하였다.
- cv2.drawContours()
  : 컨투어가 저장된 리스트와 컨투어를 그리고자 하는 이미지를 입력으로 넣어주면 해당 이미지에 특정 인덱스를 가진 컨투어를 표시한다. 표시할 선의 색과 굵기를 설정할 수 있다.
- cv2.rectangle()
  : 입력 이미지 위에 사각형을 그리는 함수이다. 사각형의 꼭짓점 좌표와 선의 색과 굵기를 설정할 수 있다.

(3) pyzbar 내장 함수

- pyzbar.decode()
  : 입력 이미지(그레이스케일)에서 바코드를 인식하여 해당 바코드 정보를 가지는 객체를 생성한다.

(4) numpy 내장 함수

- np.sqrt()
  : square root, 즉 제곱근을 구하는 함수이다.
- np.meshgrid()
  : x축, y축의 값을 입력받아 좌표 행렬을 생성하여 반환하는 함수이다. 우리는 트럭의 벽면을 표현하기 위해 사용하였다.
- np.zeros()
  : 0으로 초기화되어있는 배열을 생성하는 함수이다. 우리는 트럭의 내부 상태를 관리하기 위해 3차원 배열을 선언하여 사용하였다.

- turtlebot_manipulation_bringup.launch (turtlebot3_manipulation_bringup.cpp)
  https://github.com/ROBOTIS-GIT/turtlebot3_manipulation/tree/master/turtlebot3_manipulation_bringup 오픈 소스 활용
  - Turtlebot3ManipulationBringup::armActionCallback
    : 매니퓰레이터에 대한 콜백함수로, 매니퓰레이터 액션에 대한 행동을 서버에 전달한다.
  - Turtlebot3ManipulationBringup::gripperActionCallback
    : 매니퓰레이터 그리퍼에 대한 콜백함수로, 그리퍼 액션에 대한 행동을 서버에 전달한다.

- turtlebot_manipulation_gui.launch (main.cpp)
  https://github.com/ROBOTIS-GIT/turtlebot3_manipulation/blob/master/turtlebot3_manipulation_gui/src/main_window.cpp 오픈 소스 활용
  - chatterCallback(const std_msgs::String::ConstPtr& msg)
    : 메시지 통신에 대한 콜백함수로, subscriber 역할을 한다. 전달받은 메시지의 주소값

을 배열에 저장한 뒤 큐에 push하여 사용한다.

- on_btn_timer_start_clicked()
  : GUI를 통하여 데이터를 받았을 때 실행하는 함수로 오픈 매니퓰레이터에 대한 타이머를 시작한다.
- on_btn_init_pose_clicked()
  : GUI를 통하여 데이터를 받았을 때 실행하는 함수로 오픈 매니퓰레이터를 초기 위치로 이동시킨다.
- on_btn_gripper_close_clicked()
  : GUI를 통하여 데이터를 받았을 때 실행하는 함수로 오픈 매니퓰레이터의 그리퍼를 닫는다.
- on_btn_send_joint_angle_clicked(float a, float b, float c, float d)
  : 원래의 기능은 GUI를 통하여 데이터를 받아 오픈 매니퓰레이터의 위치를 전달하는 것이며, 전달할 데이터에 맞게 코드를 수정하였다.
  현재 변경된 방식은 ROS 통신을 통하여 데이터를 전달받은 후 상자가 컨베이어 벨트의 끝부분(지정된 위치)에 도달한 것이 인식된다면 플래그를 0에서 1로 바꾸고 이 함수를 실행시킨다.
  각 주소지 별로 오픈 매니퓰레이터가 박스를 전달해야 하는 위치를 설정해두었으며 이 함수에서 float형 a, b, c, d를 인자로 받아들인다.
  이후 전달받은 a, b, c, d 값에 따라 현재 state에서 각각의 조인트 앵글을 변경한다. 로봇팔의 작업 수행에 소요되는 시간을 고려해야 하므로 본 함수의 각 실행 사이에 딜레이를 넣어주었다.

○ 주요 함수의 흐름도

- box_detection()

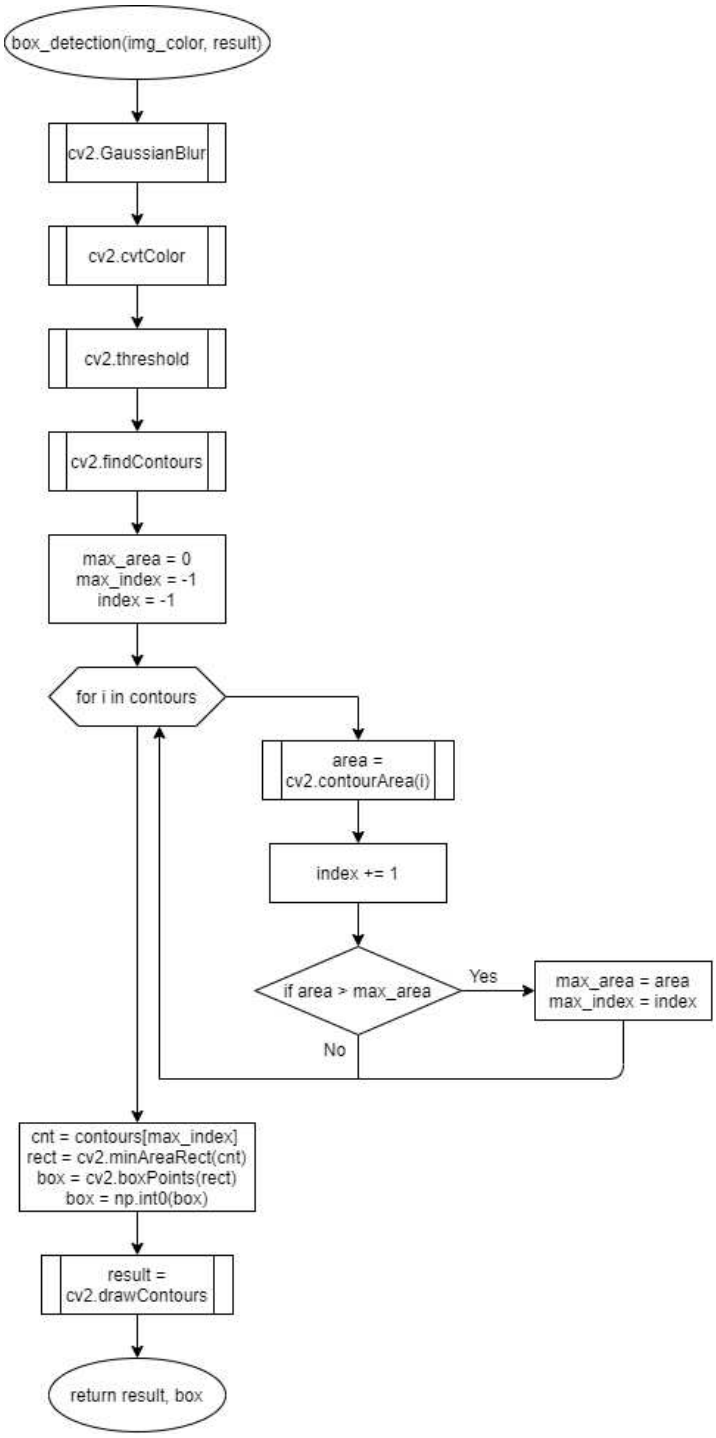

그림 22. box_detection()의 흐름도

- get_box_info()

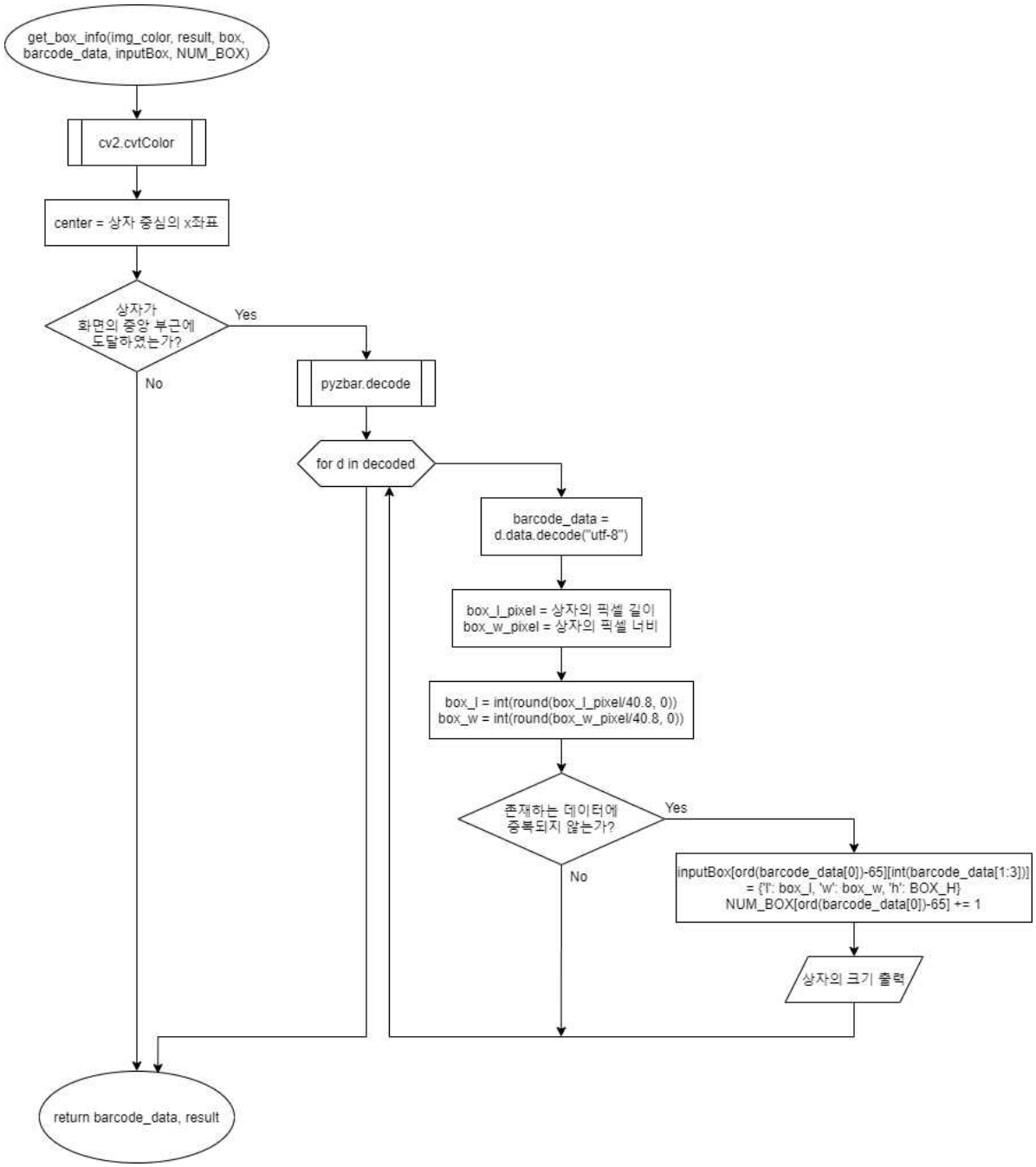

그림 23. get_box_info()의 흐름도

- calculate_loading_order()

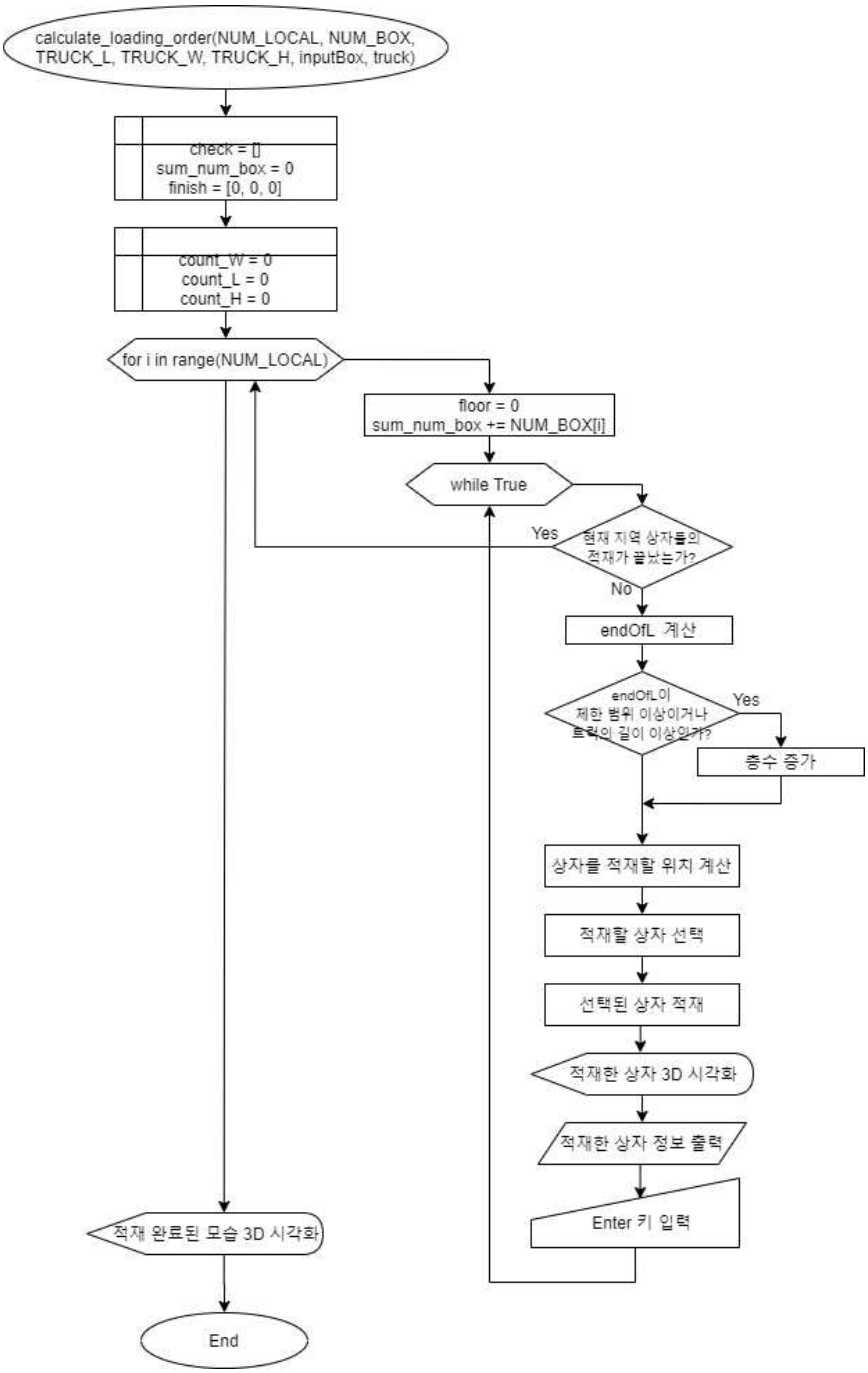

그림 24. calculate_loading_order()의 흐름도

- main.cpp

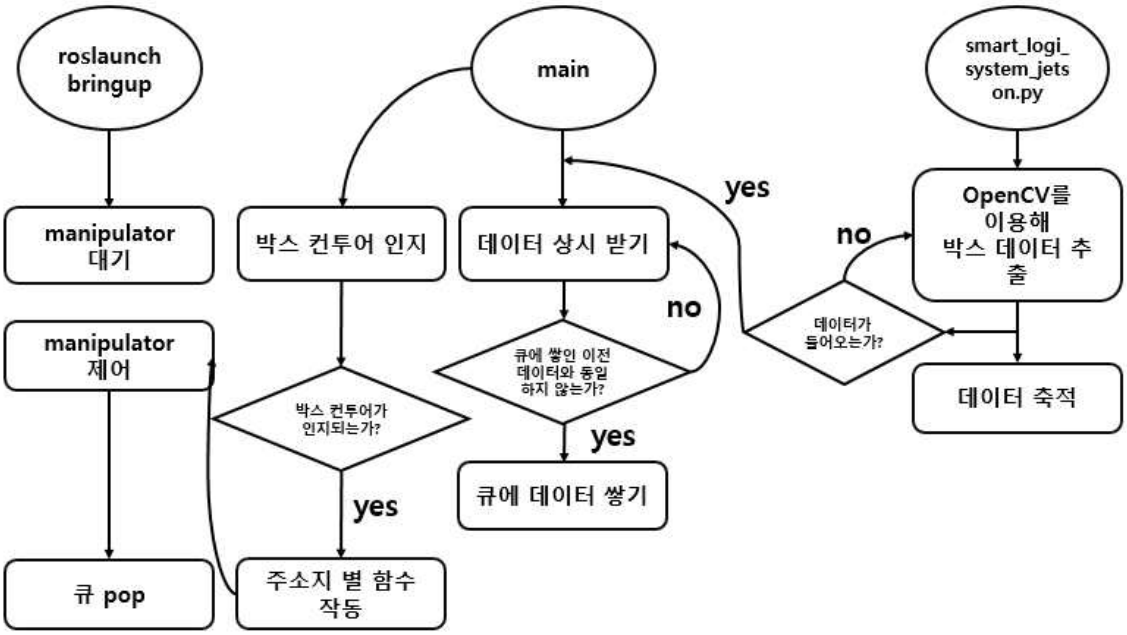

그림 25. main.cpp의 흐름도

- HostPC에서 roscore를 실행하여 서버를 열어둔 후 시작한다.
- smart_logi_system_jetson.py와 turtlebot_manipulation_gui.launch를 통한 main.cpp를 실행하여 각 PC와 보드를 구동 상태로 만든다.
- smart_logi_system_jetson.py에서 카메라 앞을 지나가는 상자들을 OpenCV로 처리하여 상자의 정보를 추출한다. 이 정보들 중 바코드 데이터를 ROS 메시지 통신을 사용하여 서버에 전송하면 main.cpp에서 데이터를 받아와 큐에 저장한다.
- main.cpp에서도 OpenCV를 이용하여 manipulator가 닿을 수 있는 거리에 상자가 도달하였는지 인식한다. 도달하였다면, 큐에 저장된 데이터를 이용하여 각 배송지에 맞게 분류하도록 manipulator를 제어한 후 해당 데이터를 큐에서 삭제한다.

○ 기술적 차별성

- smart_logi_system_jetson.py
  - 하나의 소스 파일에 모든 기능을 구현하여 이 파일만 실행하면 되므로 편리하다.
  - 여러 오픈소스 라이브러리를 활용하였다.
  - ROS 통신을 이용하여 간단하게 데이터를 전송한다.
  - OpenCV의 내장 함수인 findContours를 사용하여 상자를 쉽게 인식한다.
  - 바이너리 변환의 threshold를 트랙바로 조정할 수 있게 하여 조도 상황에 따라 값을 유동적으로 변경할 수 있다.
  - 비례식을 통해 2D 이미지만으로도 상자의 실제 크기를 측정할 수 있다.
  - 딕셔너리와 리스트 등 Python에만 존재하는 자료형을 경우에 맞게 활용하였다.
  - 트럭의 상태를 0(빈 공간)과 0이 아닌 정수(상자가 존재하는 공간)로 구분함으로써 트럭 내부 공간을 관리한다.
  - matplotlib를 활용하여 별도의 프로그램 설치 없이 라이브러리만 추가해도 3D 시각화를 확인할 수 있다.

- main.cpp
  - ROS 통신을 이용하여 간단하게 데이터를 수신한다.
  - 들어온 데이터가 처리되기 전에 또 다른 데이터가 들어오는 상황을 고려하여, 받아온 데이터를 Queue에 push하고 처리된 데이터는 pop하는 방식으로 데이터를 관리한다.
  - 데이터가 들어오면 직전에 들어온 데이터와 비교하여 중복된 데이터가 저장되지 않도록 한다.
  - 오픈소스를 활용하여 로봇팔을 원하는 방향으로 제어할 수 있도록 수정하였다.
  - 오픈소스에서 콜백 함수를 사용하기 때문에, ROS의 AsyncSpinner를 사용하였다. 함수가 실행된 후 호출되기를 기다리는 모든 콜백을 호출하게 하는 동시에, 로봇팔의 명령 수행 속도를 고려한 적절한 딜레이를 적용함으로써 별도의 입력 없이 코드가 매끄럽게 진행되도록 하였다.

□ 개발 중 장애요인과 해결방안

○ 개발 환경 내 버전 충돌

- 문제점: 모든 개발이 리눅스 환경에서 진행되었으며, Ubuntu 16.04를 사용하였다. 그 과정에서 버전에 대한 충돌이 나는 경우가 많았다.
- 해결 방법: 1차적으로, 충돌이 나는 부분에 대해서 사용하지 않는 것이라면 삭제하고 업그레이드나 다운그레이드가 필요하다면 따로 재설치를 진행하였다.
  2차적으로, 1차 시도에도 해결되지 않는 것에 대해서는 대체할 수 있는 방법을 모색하였다. 다른 알고리즘을 생각하거나 개발환경을 처음 상태로 리셋하였다.

○ 영상의 초점 문제

- 문제점: 영상처리를 이용하여 상자 정보 수집 작업을 수행할 때 웹카메라를 이용하여 frame을 받아와야 한다. 기존에 사용하던 카메라가 초점을 잡지 못해 바코드를 스캔하지 못하거나 상자의 크기를 정확하게 측정하지 못하는 문제가 있었다.
- 해결 방법: 빠른 Auto Focusing이 가능한 카메라를 구매하여 사용하였다.

○ 수행 환경의 조도 문제

- 문제점: 카메라 화면을 받아와 영상처리를 하는 과정에서 수행 환경의 조도 변화(낮과 밤, 조명의 개수, 수행 장소 등에 의함)에 의해 컨투어를 아예 찾지 못하여 프로그램이 종료되거나, 상자의 컨투어를 정확하게 추출하지 못하는 상황이 발생하였다.
- 해결 방법: 컨투어를 찾기 위한 바이너리 이미지 생성 과정에서 입력되는 threshold값을 상황에 따라 유동적으로 조정하기 위해 출력 화면에 트랙바를 추가하였다.

○ 통신환경

- 현재 상황: 유선 랜을 이용한 통신은 개발 목적과 맞지 않아 WIFI를 사용하는 방식으로 진행하였다.
- 문제점: 서버와 클라이언트 보드간에 sync가 맞지 않는 문제가 간혹 일어나는데 통신환경이 불안정하기 때문이다.
- 해결 방법: 라우터를 변경하였고 통신망이 조금 더 활발한 장소에서 진행하였다.

○ AR Marker

- 원래 로봇팔의 그리퍼를 이용하여 상자를 잡을 때 상자의 위치, 공간 정보를 인식하기 위해 사용하고자 했던 AR Marker를 활용한 오픈소스 알고리즘을 현재 개발환경인 Ubuntu 16.04에서 사용할 수 없다. 버전의 호환성 문제인 것으로 확인되었다.
- 해결 방법: 상자의 위치를 OpenCV를 이용한 영상처리를 통해 인식하고 로봇팔의 이동 위치를 직접 계산하여 코드로 작성하는 방식으로 알고리즘을 수정하였다. 이후 18.04로 버전을 업그레이드하고 난 후에 AR Marker 알고리즘을 다시 적용해볼 예정이다.

○ 물류 형태의 제한

- 문제점: 실제 물류업체에서 취급하는 물류들은 직육면체 뿐만이 아닌 다양한 형태로 존재하나, 현재 수준에서는 직육면체가 아닌 다른 형태까지 고려한 알고리즘을 구현하기 어렵다.
- 해결 방법: 우선 직육면체의 형태로만 제한하여 알고리즘을 구현하였다.

□ 개발결과물의 차별성

○ 현업 종사자를 배려한 사람 친화적인 기술

현재 물류 시장에서 점점 자동화에 관심을 가지고 발전시켜나가는 추세이다. 하지만 실제로 배송 직전에 물류를 적절한 순서로 차량에 적재하는 일은 온전히 택배기사의 개인적인 노하우와 노동을 필요로 한다. 노하우가 없는 신입 기사들은 배송할 때마다 큰 트럭 안에 무작위로 위치한 택배를 찾아야 하는 어려움을 겪기도 한다.

본 스마트 물류 시스템을 이용한다면 택배기사들의 시행착오와 불필요한 노동을 줄일 수 있다.

○ 배송의 효율성 증가

본 시스템은 프로그램을 통해 물류의 적재 순서를 계산하기 때문에 매우 빠르게 결과를 도출하고 사용자에게 시각적으로 공유함으로써 사람이 어떠한 고민도 할 필요 없이 바로 적재를 진행할 수 있다.

또한 알고리즘에서 트럭의 상태를 파악하고 해당 위치에 적합한 상자를 찾는 작업을 반복하기 때문에 빈 공간을 최소화하는 방향으로 적재 순서를 계산한다. 따라서 같은 크기의 공간에도 더 많은 상자를 적재하게 되어 한 대의 트럭이 기존보다 더 많은 택배를 배송할 수 있다.

이처럼 시간적인 효율성이 높아지므로 물류 업체에서 경제적인 이득을 볼 수 있다.

○ 광범위한 활용 분야

- 적재 순서 계산 알고리즘과 로봇팔 제어 알고리즘을 일부 수정하면 물류 업체뿐만 아니라 스마트 팩토리, 대형 창고, 물품 보관 서비스 등 다른 분야에서도 활용할 수 있다.
- 알고리즘이 함수화되어있기 때문에 트럭의 크기 대신 창고의 크기를 입력하거나 로봇팔의 이동 범위를 변경하는 등 함수에 전달하는 인자를 필요에 따라 수정하여 쉽게 사용할 수 있다.

□ 개발 일정

	내용	2020년							
		3월	4월	5월	6월	7월	8월	9월	10월
1	아이디어 구상								
2	실현 가능/타당성 검출								
	유사 사례 검토								
	기술 분석, 자료 조사								
3	기능 설계 및 분석								
4	전체 시스템 설계								
5	환경 구축								
6	적재 알고리즘 설계								
7	로봇팔 알고리즘 설계								
8	데이터 통신								
9	구현 알고리즘 통합								
10	시험 평가 및 테스트								
11	문서화, 영상 제작								

□ 팀 업무 분장

No	구분	성명	참여인원의 업무 분장
1	팀장	강민지	- 전송받은 상자의 데이터를 해석하여 상자를 각 배송지에 따라 분류하고 로봇팔을 제어하여 상자를 이동시키는 알고리즘 개발 - 전반적인 개발환경 구축 및 버전/호환성 문제 해결 - ROS 메시지 통신을 이용하여 문자열 데이터를 수신하는 기능 구현
2	팀원	권미경	- 영상처리를 통한 상자 인식, 상자 크기 측정, 바코드 스캔 등을 수행하는 알고리즘 개발 - 상자의 크기와 배송지를 고려하여 최적의 적재 순서를 계산하고 시각화하는 알고리즘 개발 - ROS 메시지 통신을 이용하여 문자열 데이터를 송신하는 기능 구현