

LIMO

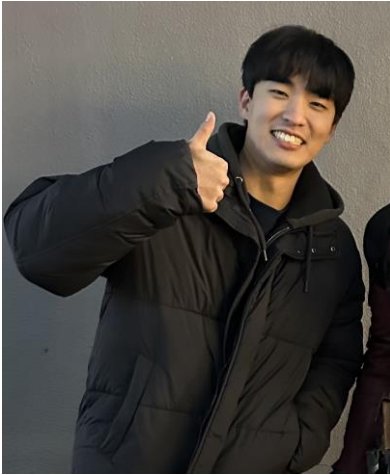
LIMO 사용법

- Hardware Manual -

Department of Electrical Engineering, Incheon National University
Cheolmin Jeong, Youngkeun Kim

2024.06.11.

강사 소개



■ 김영근 (Youngkeun Kim)

■ ~2024: 인천대학교 전기공학과 학사

■ 2024~현재: 인천대학교 전기공학과 석박사 통합과정 재학 중

■ 무인지능 시스템제어 연구실 소속

(지도교수: 강창묵 교수, <https://uniconlab.wixsite.com/main>)

■ 관심분야: 고장진단, 자율주행, SLAM, 인공지능

■ 적용플랫폼: 자율주행차량, 모바일로봇, 로봇 팔, 4족보행로봇 등

■ Projects:

- CNN 기반의 wavelet 이미지 분류
- 인천공항 폐기물 이송 로봇 개발
- ROS 기반 시뮬레이션 환경 구성
- ...

Contents

1. 소개

- 1) 구성품
- 2) 각 부분 명칭

2. 기본 사항

- 1) LIMO 기능
- 2) Drive mode

3. 로봇 사용

- 1) 사용 및 작동
- 2) 컨트롤러

Appendix



소개

1. 소개

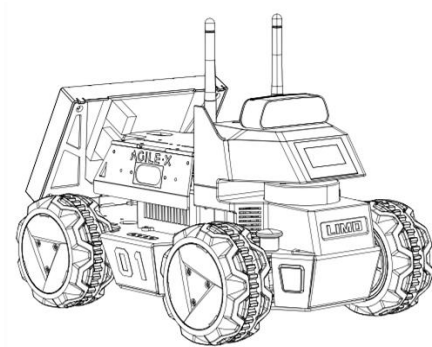
1) 구성품

이름	수
LIMO	X 1
배터리	X 1
충전기	X 1
Mecanum wheel	X 4
Track	X 2
Cross screwdriver	X 1
screw	M3x12mm : x 3 M3x5mm : x 20

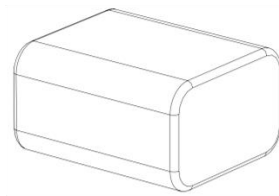
- Jetson Nano, EAI X2L LiDAR, Depth Camera 장착
- SLAM mapping, Path Planning, Obstacle Avoidance 기능 제공

1. 소개

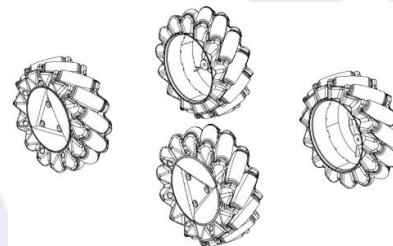
1) 구성품



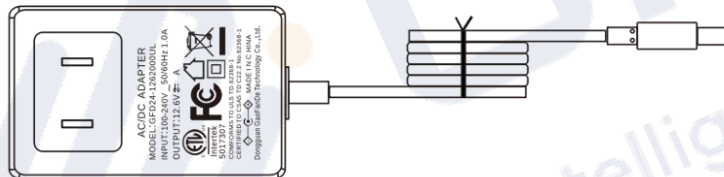
LIMO high-end body *1 (install off-road wheel *4)



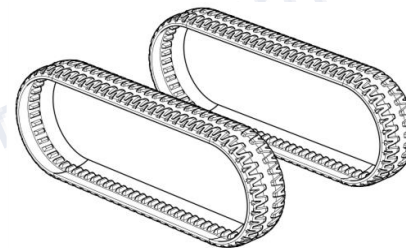
Battery *1



Mecanum wheel *4



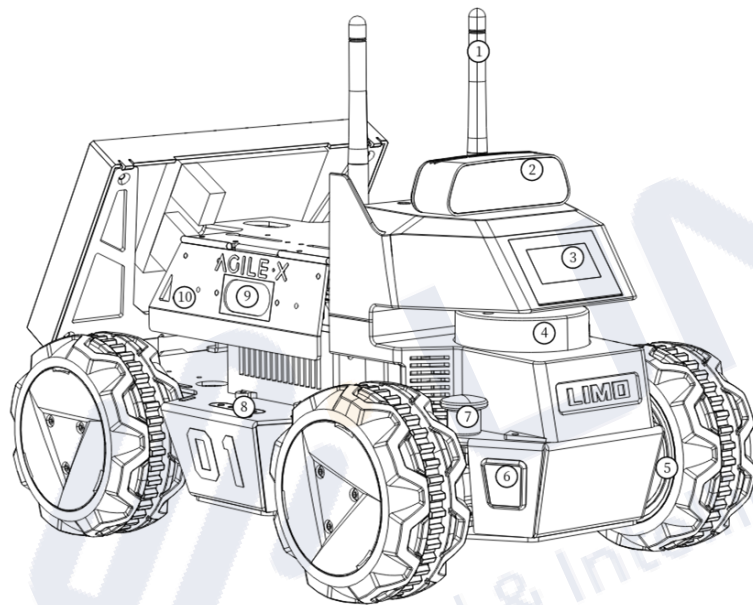
Charger *1



Track *2

1. 소개

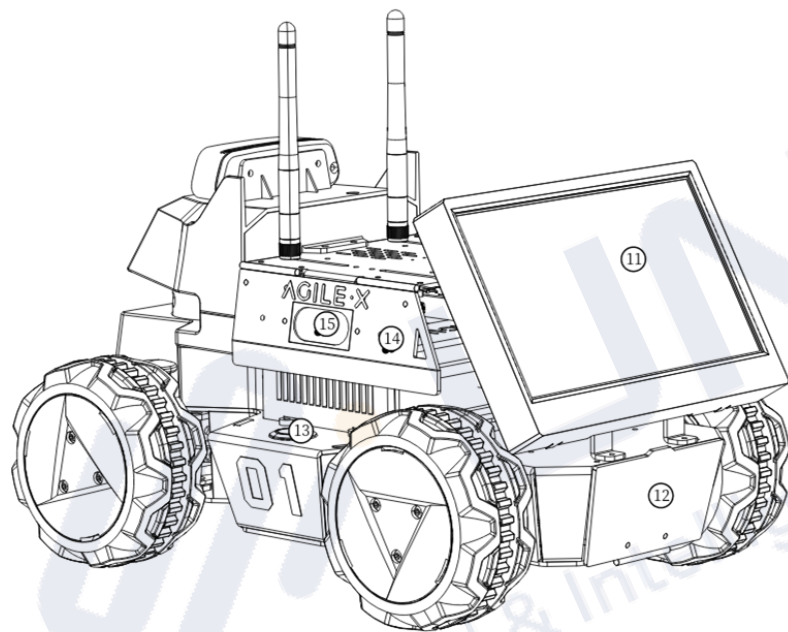
2) 각 부분 명칭



1	WiFi / Bluetooth antenna
2	Depth Camera
3	Front Display
4	EAI X2L LiDAR
5	Hub motor
6	RGB light
7	Four-wheel differential / Ackermann mode switching latch
8	Power display
9	Left speaker
10	Left seagull door

1. 소개

2) 각 부분 명칭



11	Rear display
12	Battery door
13	Switch
14	Right seagull door
15	Right speaker

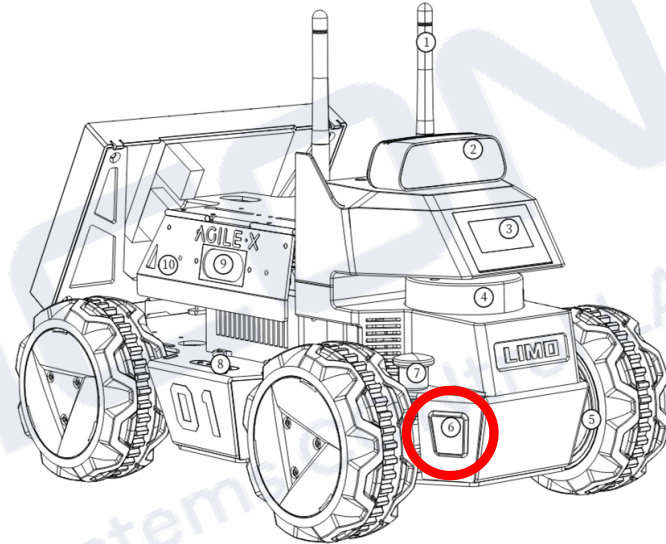
기본 사항

2. 기본 사항

1) LIMO 기능

- LED

- 전방 LED는 5개의 색상으로 현재의 상태를 표시
- 적색 점멸, 배터리 부족
- 적색, software 정지
- 녹색, Ackermann mode
- 황색, differential or track mode
- 청색, mecanum wheel mode



- Drive Mode



Four-wheel
differential
steering



Ackermann



Omni-
directional
steering



Tracked
steering

2. 기본 사항

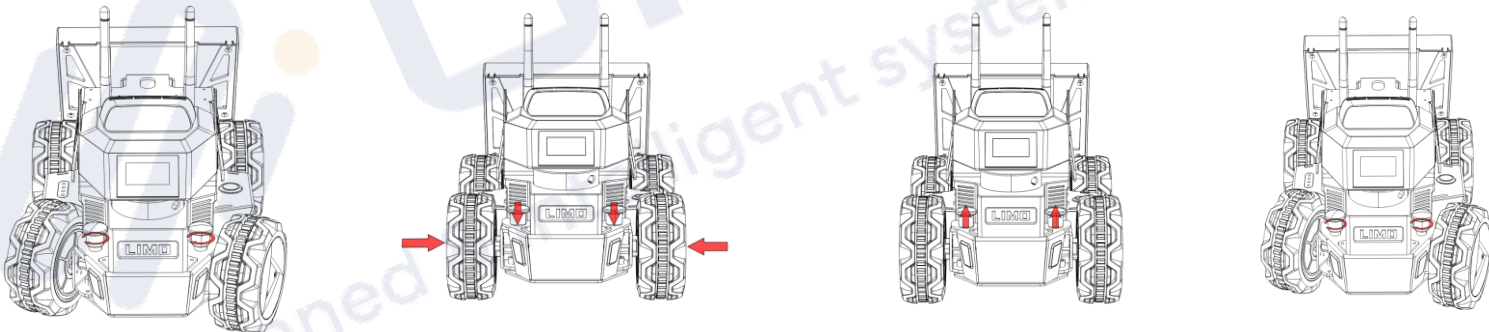
2) Drive mode

- Differential steering

- 좌우로 구분된 두 개의 바퀴를 사용.
- 직진 상태에서는 두 바퀴가 동일한 속도로 회전하여 차량이 직진, 회전 시에는 좌우 바퀴의 회전 속도를 조절하여 차량의 회전 제어

- Ackermann Drive

- 차량의 조향 각도에 따라서 바퀴의 회전 속도를 제어하여 움직임 조절.



- 전방에 위치한 빨간색 걸쇠를 올리고 내려서 Drive mode 전환
- 내릴 때는, 양쪽 바퀴를 정면을 보도록 하고 내려야 함

로봇 사용

3. 로봇 사용

1) 사용 및 작동

- 전원 켜기

- 좌측의 전원 버튼을 길게 누름 (뽁 소리가 날 때까지)
- 오른쪽의 전력 상태를 확인하고, 부족할 경우 충전하거나 다른 배터리로 교체

- 전원 끄기

- 좌측의 전원 버튼을 길게 누름

- UBUNTU 로그인

- ID: agilex
- PW: agx

- 충전

- 경고음이 울리면, 배터리 잔량이 부족한 것이므로 배터리 충전 해야함
- 모니터 하단의 덮개를 아래로 내리면 있는 충전 단자를 통해 충전

3. 로봇 사용

2) 컨트롤러

- 스위치

이름	SW A	SW B	SW C	SW D
기능	없음	상단 / 코드 가운데 / 수동 하단 / 정지	없음	Differential Mecanum 변경



3. 로봇 사용

3) 자율 주행

- map base

- ✓ 맵을 알고 있을 때, 맵을 기준으로 주행.
- ✓ 맵 상에서 자신의 위치를 정확히 파악하는 것이 관건

- sensor base

- ✓ 매 순간 자신이 가야할 곳을 결정
- ✓ 맵을 모르기 때문에 localization을 할 필요가 없음

실습 준비

1. git clone https://github.com/0-keun/limo_tutorials.git
2. limo_ws/src 경로에 limo_tutorials 폴더 옮기기
3. limo_tutorials/scripts 폴더 내의 모든 python 파일에 chmod 이용해서 권한 부여하기 (sudo chmod 777 *.py) 반드시 경로 확인할 것

LiDAR

4. LiDAR

1) SLAM

- mapping
 - roslaunch limo_bringup limo_start.launch
 - roslaunch limo_bringup limo_gmapping.launch
- map 저장
 - rosrun map_server map_saver -f test_map

2) map 기반 주행

- lidar 기반 주행 준비
 - roslaunch limo_bringup limo_start.launch

launch 파일 수정
 limo_bringup/launch 경로에서
 limo_navigation_diff.launch 파일 수정
 map -> test_map으로 변경

- roslaunch limo_bringup limo_navigation_diff.launch

```

<!-- ***** map server ***** -->
<node pkg="map_server" type="map_server" name="map_server"
test_map.yaml" output="screen" >
<param name="frame_id" value="map" />
</node>
<!-- ***** Navigation ***** -->
<node pkg="move_base" type="move_base" respawn="false"
  <roscparam file="$(find limo_bringup)/param/diff
command="load" ns="global_costmap" />
  <roscparam file="$(find limo_bringup)/param/diff
command="load" ns="local_costmap" />
  <roscparam file="$(find limo_bringup)/param/diff
command="load" />
  <roscparam file="$(find limo_bringup)/param/diff
command="load" />
  <roscparam file="$(find limo_bringup)/param/diff

```

4. LiDAR

2) map 기반 주행

- lidar 기반 주행 방법

- move_base/goal publish 해보기

```
roslaunch limo_tutorials 3_goal_publish.py
```

- 좌표 기록하기

```
roslaunch limo_tutorials 4_footprint.py
```



1. map을 만들고 2. footprint를 적절히 수정해서 경로의 좌표들을 얻어낸 후 3. 해당 경로로 goal을 순차적으로 publish 해준다.

ROS openv

4. ROS opencv

1) image topic

- topic publish
 - `roslaunch astra_camera dabai_u3.launch`
- topic subscribe and imshow
 - `roslaunch limo_tutorials 1_ROS_camera.py`

2) 응용

- 차선 검출
 - `roslaunch limo_tutorials 2_HSV_with_Hough.py`

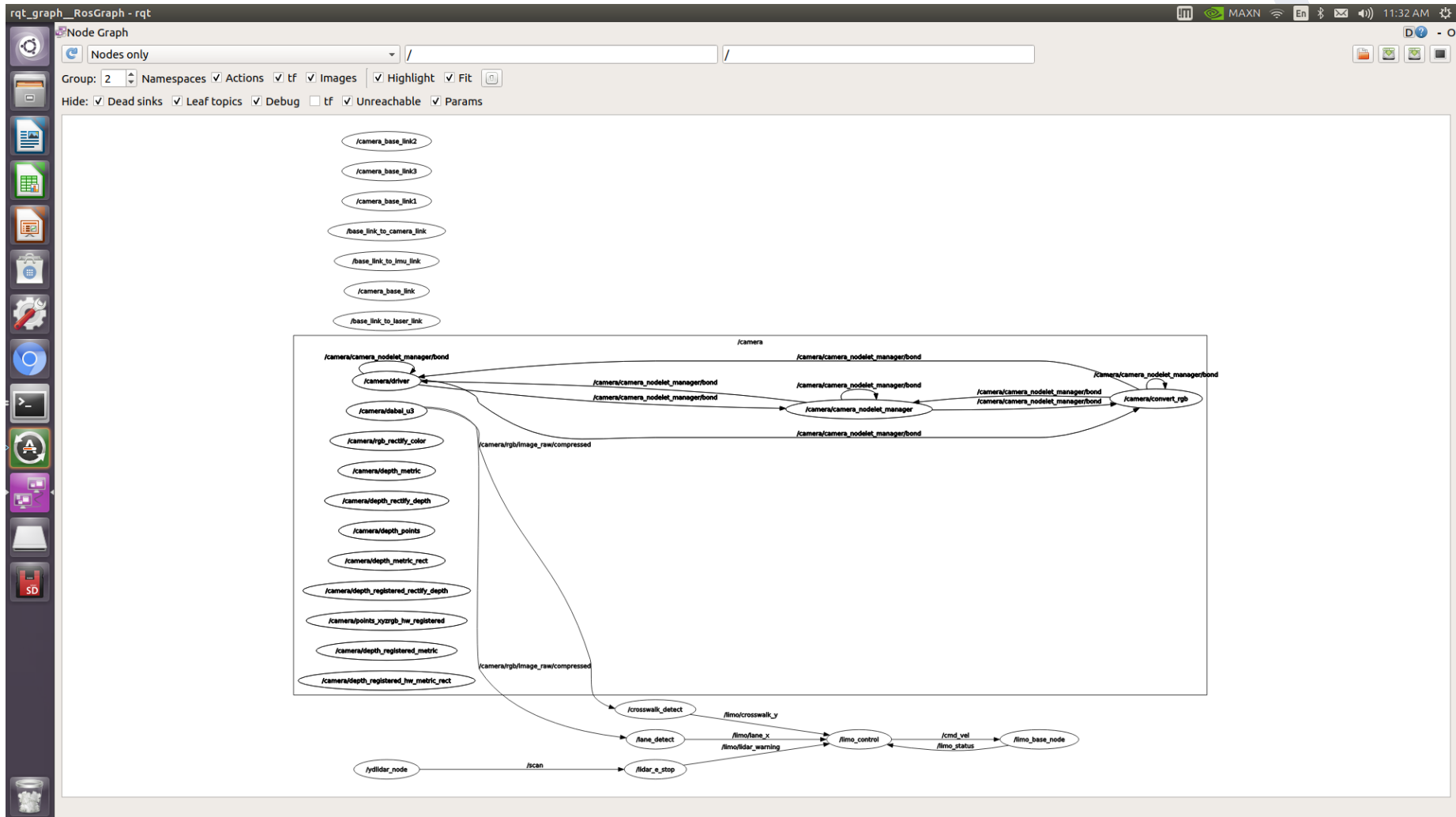


opencv tutorial 과정에서 배운 내용을 합쳐서 성능 개선 (ex BEV)

자율주행 알고리즘

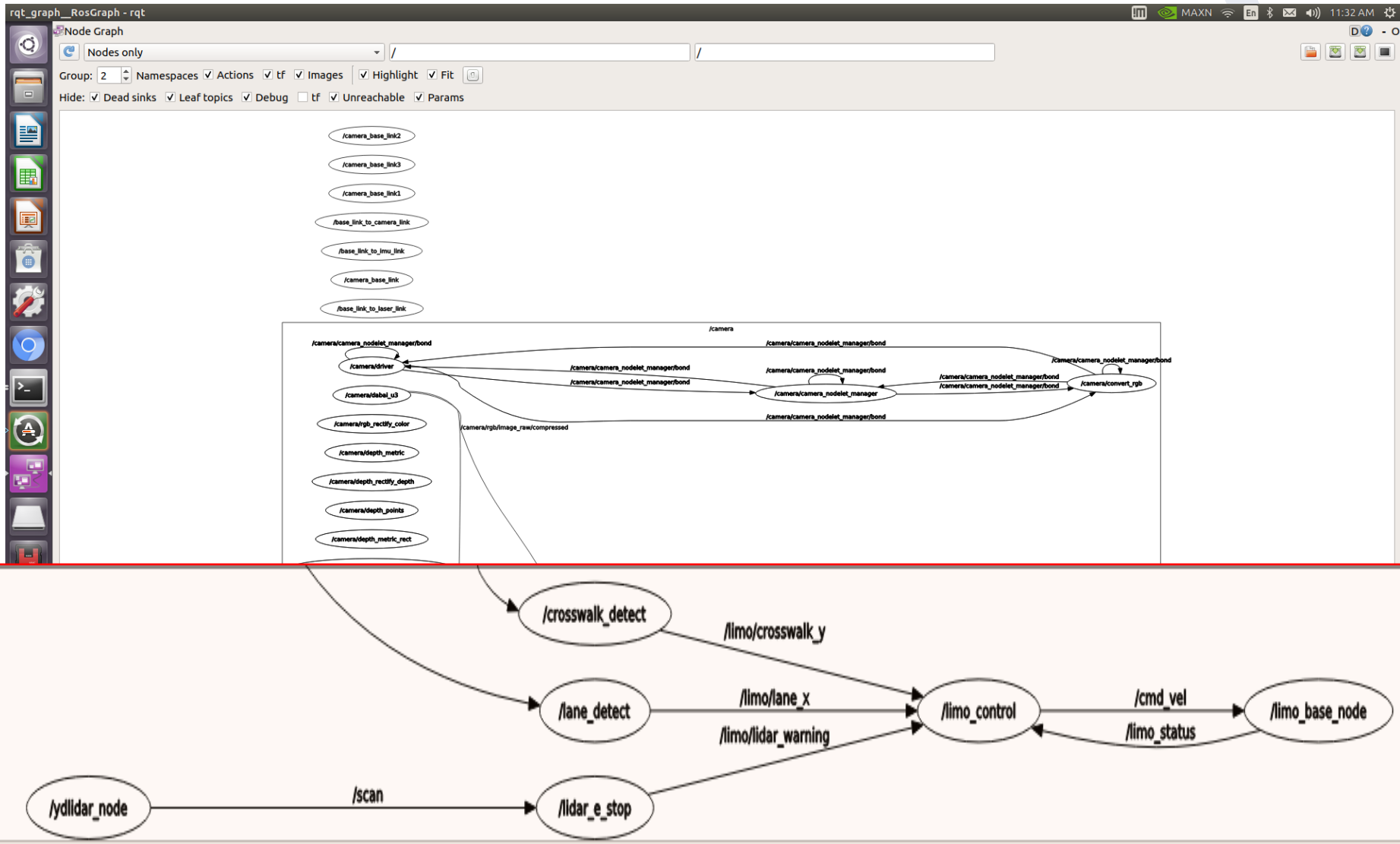
5. 자율주행 알고리즘

1) rqt_graph



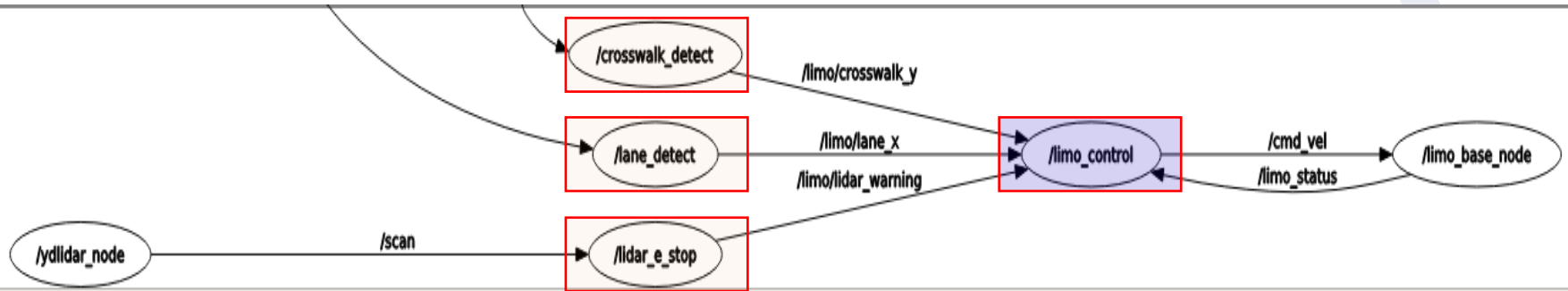
5. 자율주행 알고리즘

1) rqt_graph


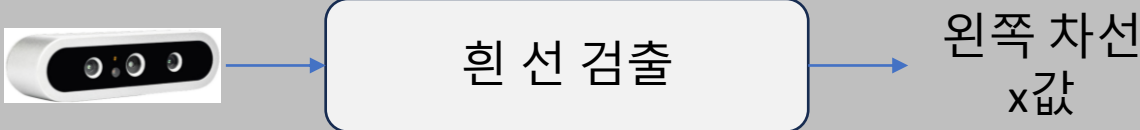

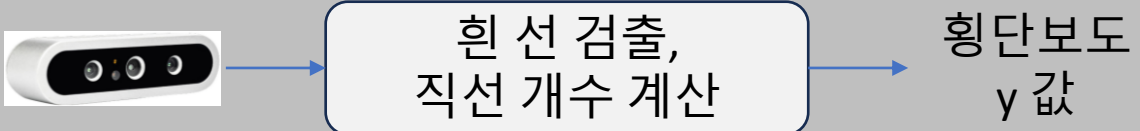

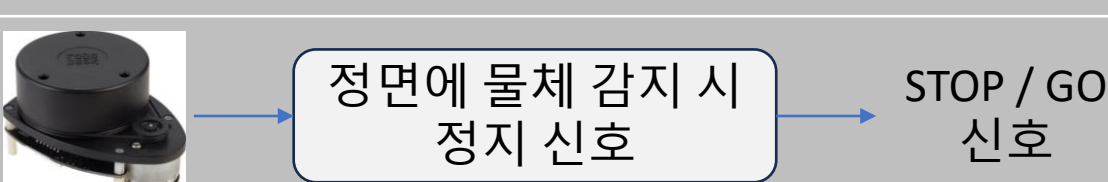


5. 자율주행 알고리즘

1) rqt_graph



● nodes

노드 명	기능
lane_detect	 
crosswalk_detect	 
Lidar_e_stop	 

5. 자율주행 알고리즘

2) lane_detection

```
90
91     def image_topic_callback(self, img):
92         ...
93         실제 이미지를 입력 받아서 동작하는 부분
94         CompressedImage --> OpenCV Type Image 변경 (compressed_imgmsg_to_cv2)
95         차선 영역만 ROI 지정 (imageCrop)
96         ROI 영역에서 차선 색 영역만 검출 (colorDetect)
97         검출된 차선을 기반으로 거리 계산 (calcLaneDistance)
98         최종 검출된 값을 기반으로 카메라 좌표계 기준 차선 무게중심 점의 x좌표 Publish
99         ...
100     self.frame = self.cvbridge.compressed_imgmsg_to_cv2(img, "bgr8")
101     self.cropped_image = self.imageCrop(self.frame)
102     self.thresholded_image = self.colorDetect(self.cropped_image)
103     self.left_distance = self.calcLaneDistance(self.thresholded_image)
104     self.distance_pub.publish(self.left_distance)
105
106     # visualization
107     if self.viz:
108         self.visResult()
```

5. 자율주행 알고리즘

2) lane_detection

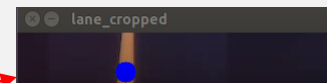
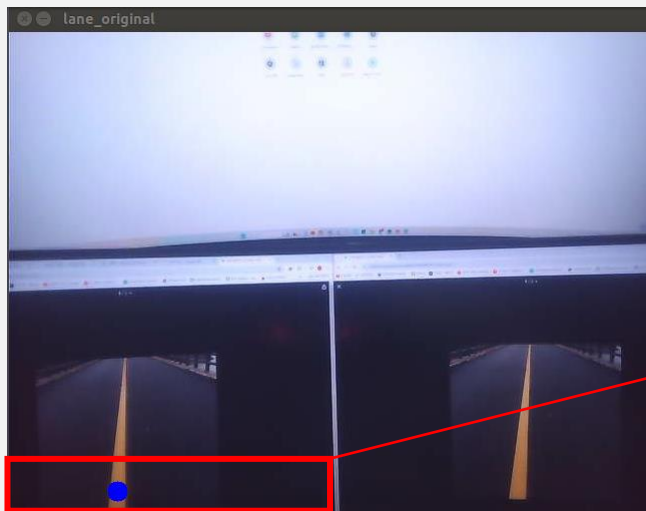
```
33 # return np.ndarray (opencv image type)
34 def imageCrop(self, _img=np.ndarray(shape=(480, 640))):
35     ...
36     원하는 이미지 영역 검출
37     ...
38     return _img[420:480, 0:320]
39
40 # return np.ndarray (opencv image type)
41 def colorDetect(self, _img=np.ndarray(shape=(480, 640))):
42     ...
43     특정 색 영역만 추출 (Dynamic Reconfigure를 통해, 값 변경 가능)
44     ...
45     hls = cv2.cvtColor(_img, cv2.COLOR_BGR2HLS)
46     mask_yellow = cv2.inRange(hls, self.YELLOW_LANE_LOW_TH, self.YELLOW_LANE_HIGH_TH)
47     return mask_yellow
48
49 # return Int
50 def calcLaneDistance(self, _img=np.ndarray(shape=(480, 640))):
51     ...
52     최종 검출된 이미지를 이용하여 차선의 모멘트 계산
53     모멘트의 x, y 좌표 중 차량과의 거리에 해당하는 x를 반환
54     ...
55     try:
56         M = cv2.moments(_img)
57         self.x = int(M['m10']/M['m00'])
58         self.y = int(M['m01']/M['m00'])
59     except:
60         self.x = -1
61         self.y = -1
62     # print("x, y = {}, {}".format(x, y))
63     return self.x
```

5. 자율주행 알고리즘

2) lane_detection

```
33 # return np.ndarray (opencv image type)
34 def imageCrop(self, _img=np.ndarray(shape=(480, 640))):
35     ...
36     원하는 이미지 영역 검출
37     ...
38     return _img[420:480, 0:320]
39
```

imageCrop()



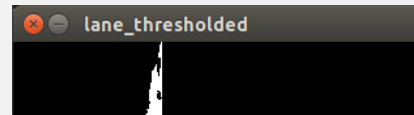
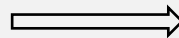
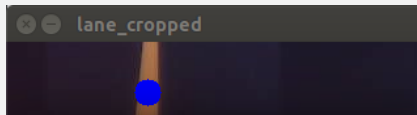
```
61 self.y = -1
62 # print("x, y = {}, {}".format(x, y))
63 return self.x
```

5. 자율주행 알고리즘

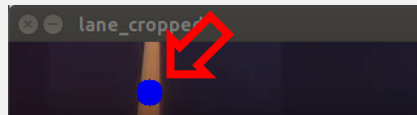
2) lane_detection

```
33 # return np.ndarray (opencv image type)
34 def imageCrop(self, _img=np.ndarray(shape=(480, 640))):
35     ...
36     원하는 이미지 영역 검출
37     ...
38     return _img[420:480, 0:320]
```

colorDetect()



calcLaneDistance()



```
56 M = cv2.moments(_img)
57 self.x = int(M['m10']/M['m00'])
58 self.y = int(M['m01']/M['m00'])
59 except:
60     self.x = -1
61     self.y = -1
62 # print("x, y = {}, {}".format(x, y))
63 return self.x
```

5. 자율주행 알고리즘

2) lane_detection

```

33 # return np.ndarray (opencv image type)
34 def imageCrop(self, _img=np.ndarray(shape=(480, 640))):
35     ...
36     원하는 이미지 영역 검출
37     ...
38     return _img[420:480, 0:320]
39
40 # return np.ndarray (opencv image type)
41 def colorDetect(self, _img=np.ndarray(shape=(480, 640))):
42     ...
43     특정 색 영역만 추출 (Dynamic Reconfigure를 통해, 값 변경 가능)
44     ...
45     hls = cv2.cvtColor(_img, cv2.COLOR_BGR2HLS)
46     mask_yellow = cv2.inRange(hls, self.YELLOW_LANE_LOW_TH, self.YELLOW_LANE_HIGH_TH)
47     return mask_yellow
48
49 # return Int
50 def calcLaneDistance(self, _img=np.ndarray(shape=(480, 640))):
51     ...
52     최종 검출된 이미지를 이용하여 차선의 모멘트 계산
53     모멘트의 x, y 좌표 중 차량과의 거리에 해당하는 x를 반환
54     ...
55     try:
56         M = cv2.moments(_img)
57         self.x = int(M['m10']/M['m00'])
58         self.y = int(M['m01']/M['m00'])
59     except:
60         self.x = -1
61         self.y = -1
62     # print("x, y = {}, {}".format(x, y))
63     return self.x
  
```

} : 희망 영역

} : 희망 색상

5. 자율주행 알고리즘

2) lane_detection

```

65 def visResult(self):
66     ...
67     최종 결과가 추가된 원본 이미지 (lane_original)
68     차선 영역만 ROI로 잘라낸 이미지 (lane_cropped)
69     ROI 내부 중 특정 색 영역만 검출한 이미지 (lane_thresholded)
70     ...
71     cv2.circle(self.cropped_image, (self.x, self.y), 10, 255, -1)
72     cv2.imshow("lane_original", self.frame)
73     cv2.imshow("lane_cropped", self.cropped_image)
74     cv2.imshow("lane_thresholded", self.thresholded_image)
75     cv2.waitKey(1)
76
77 # =====
78 #           Callback Functions
79 # =====
80
81 def reconfigure_callback(self, config, level):
82     ...
83     Dynamic_Reconfigure를 활용하여, 차선 검출을 위한 색 영역 지정
84     HLS Color Space를 기반으로 검출
85     노란색 차선을 검출을 위한 Threshold 설정
86     ...
87     self.YELLOW_LANE_LOW_TH = np.array([config.yellow_h_low, config.yellow_l_low, config.yellow_s_low])
88     self.YELLOW_LANE_HIGH_TH = np.array([config.yellow_h_high, config.yellow_l_high, config.yellow_s_high])
89     return config
90
91 def image_topic_callback(self, img):
92     ...
93     실제 이미지를 입력 받아서 동작하는 부분
94     CompressedImage --> OpenCV Type Image 변경 (compressed_imgmsg_to_cv2)
95     차선 영역만 ROI 지정 (imageCrop)
96     ROI 영역에서 차선 색 영역만 검출 (colorDetect)
97     검출된 차선을 기반으로 거리 계산 (calcLaneDistance)
98     최종 검출된 값을 기반으로 카메라 좌표계 기준 차선 무게중심 점의 x좌표 Publish
99     ...
100     self.frame = self.cvbridge.compressed_imgmsg_to_cv2(img, "bgr8")
101     self.cropped_image = self.imageCrop(self.frame)
102     self.thresholded_image = self.colorDetect(self.cropped_image)
103     self.left_distance = self.calcLaneDistance(self.thresholded_image)
104     self.distance_pub.publish(self.left_distance)
105
106 # visualization
107 if self.viz:
108     self.visResult()
  
```

: 영상 표시

: 색상 영역 조절

: 영상 표시 on/off

5. 자율주행 알고리즘

2) lane_detection

- 실행 (각 터미널에 별도로 실행)

- bringup

```
agilex@agilex:~$ roslaunch limo_bringup limo_start.launch
```

LIMO의 바퀴와 LiDAR 센서를 ROS 시스템에 연결

- camera

```
agilex@agilex:~$ roslaunch astra_camera dabal u3.launch
```

camera 센서를 ROS 시스템에 연결

- control

```
agilex@agilex:~$ roslaunch limo_application lane_detection.launch
```

센서 데이터를 기반으로 하는 자율주행 알고리즘 실행

5. 자율주행 알고리즘

2) lane_detection

- visResult() 실행하기
 - lane_detect.launch 파일 수정

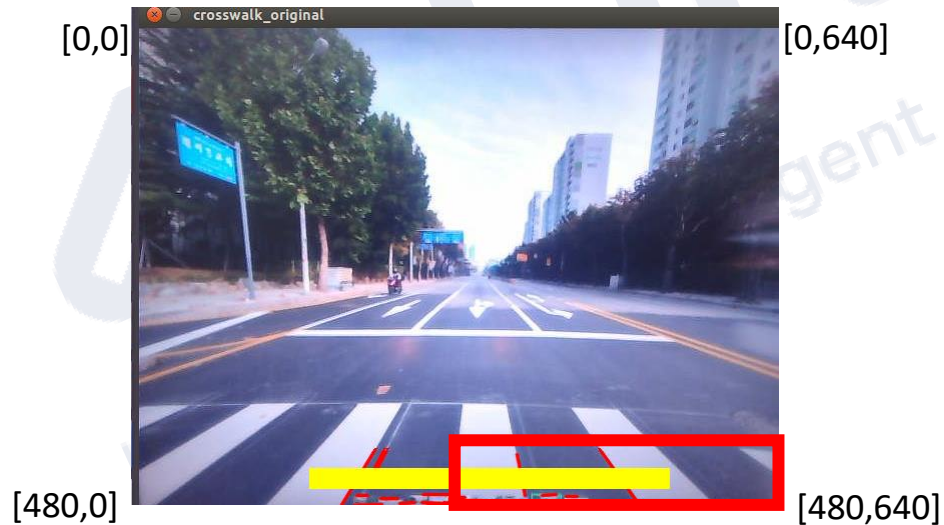
```
1  <?xml version="1.0"?>
2  <launch>
3      <node name="lane_detect" pkg="limo_application" type="lane_detect.py" output="log">
4          <rosparam file="$(find limo_application)/params/lane_detection/lane_detection.yaml" command="load" />
5          <param name="image_topic_name" value="/camera/rgb/image_raw/compressed"/>
6          <param name="visualization" value="True"/>
7      </node>
8      <node name="crosswalk_detect" pkg="limo_application" type="crosswalk_detect.py" output="screen">
9          <rosparam file="$(find limo_application)/params/lane_detection/crosswalk.yaml" command="load" />
10         <param name="image_topic_name" value="/camera/rgb/image_raw/compressed"/>
11         <param name="visualization" value="True"/>
12     </node>
13     <node name="limo_control" pkg="limo_application" type="control.py" output="screen">
14         <rosparam file="$(find limo_application)/params/lane_detection/control.yaml" command="load" />
15         <param name="control_topic_name" value="/cmd_vel"/>
16     </node>
17     <node name="lidar_e_stop" pkg="limo_application" type="e_stop.py" output="log">
18         <rosparam file="$(find limo_application)/params/lane_detection/e_stop.yaml" command="load" />
19         <param name="lidar_topic_name" value="/scan"/>
20     </node>
21 </launch>
22
```


5. 자율주행 알고리즘

2) lane_detection

- crop 영역 변경
 - lane_detect.py 파일 수정

```
def imageCrop(self, _img=np.ndarray(shape=(480, 640))):  
    ...  
    원하는 이미지 영역 검출  
    ...  
    return _img[420:480, 0:320]
```

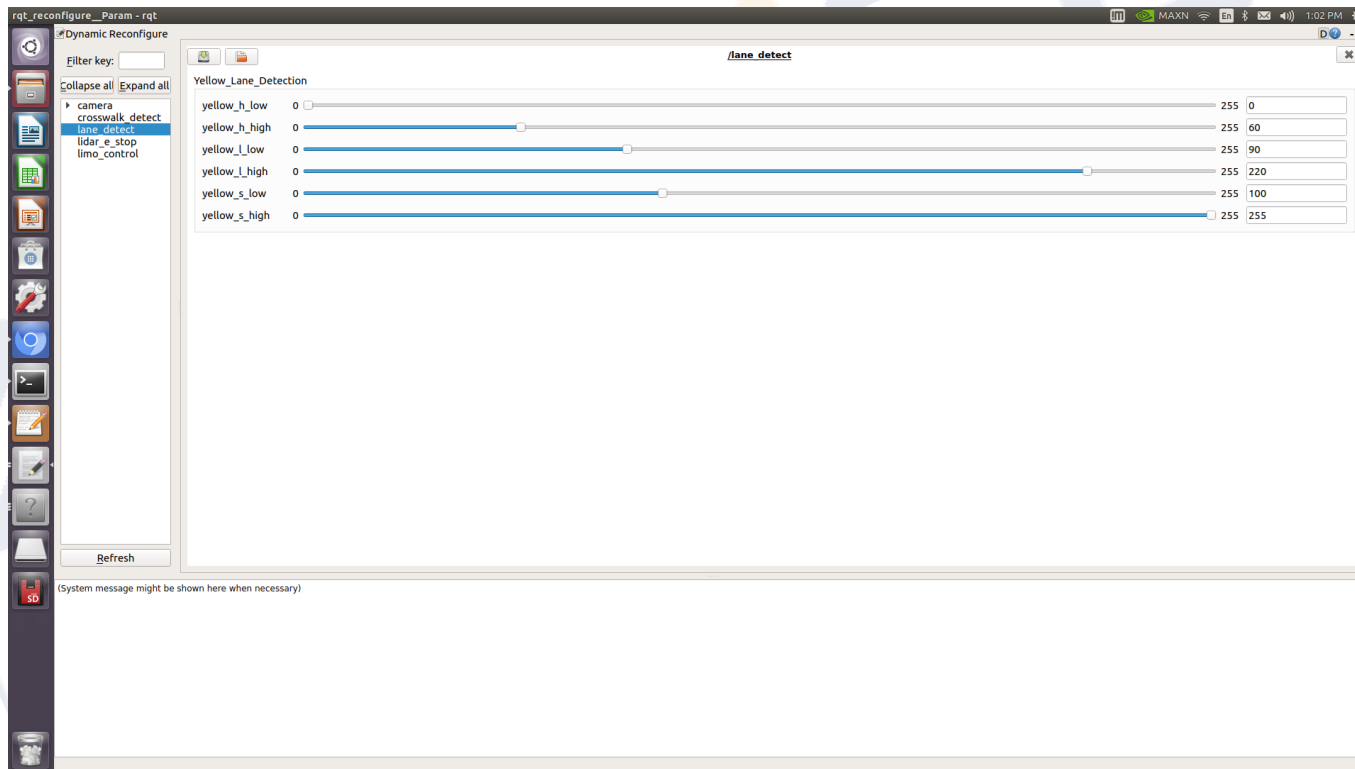


5. 자율주행 알고리즘

2) lane_detection

- 색상 변경

- parameter 수정
- 터미널에 “ `roslaunch rqt_reconfigure rqt_reconfigure` ” 명령어 입력

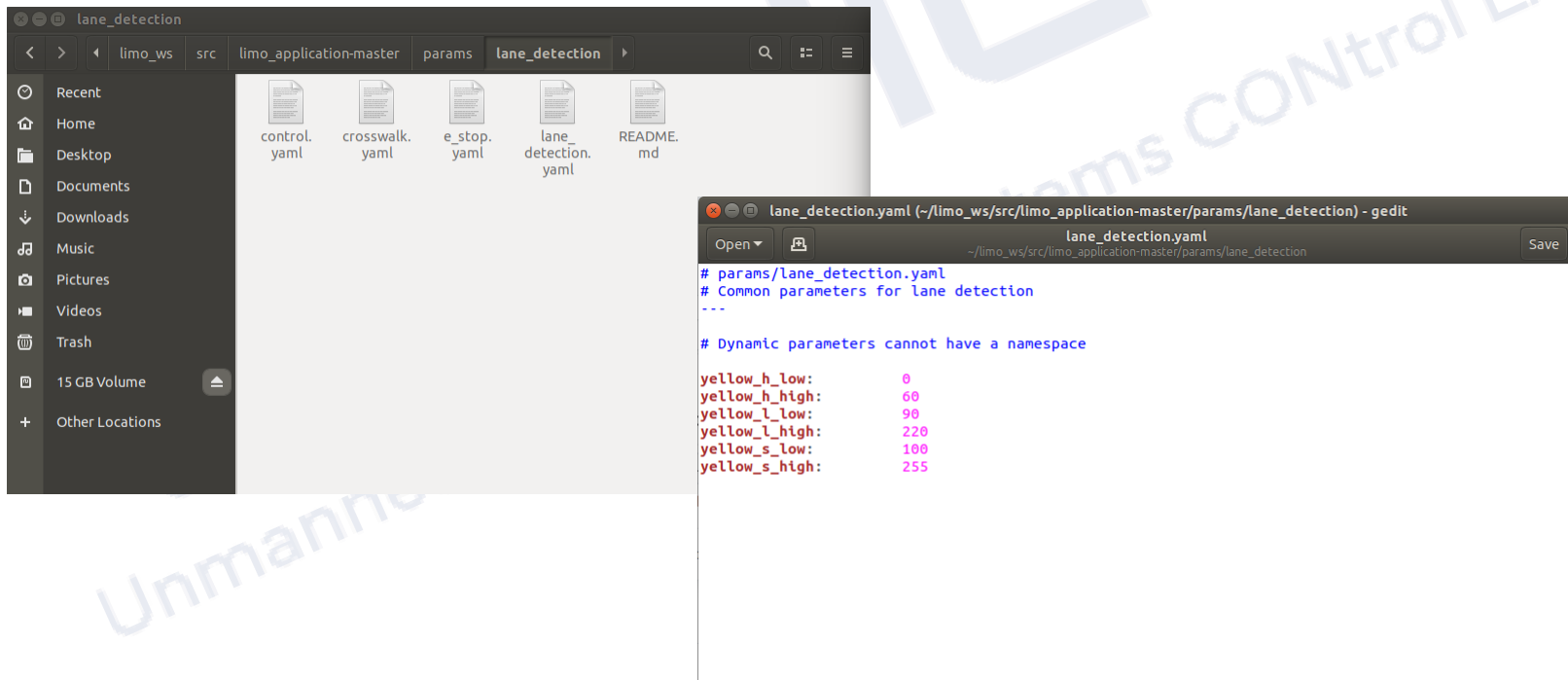


5. 자율주행 알고리즘

2) lane_detection

● 변경된 파라미터 고정

- rqt_reconfigure 를 이용한 파라미터 변경은 프로그램 종료 시 초기화
- params/lane_detection 에서 원하는 파라미터를 수정해서 default값 설정



5. 자율주행 알고리즘

3) crosswalk_detect

```
135 def Image_CB(self, img):
136     ...
137     실제 이미지를 받아서 동작하는 부분
138     CompressedImage --> OpenCV Type Image 변경 (compressed_imgmsg_to_cv2)
139     횡단 보도 영역만 ROI 지정 (imageCrop)
140     ROI 영역에서 횡단보도 색 영역만 검출(colorDetect)
141     검출된 횡단 보도의 경계면 검출 (edgeDetect)
142     검출된 경계선을 이용하여 직선 검출 (houghLineDetect)
143     최종 검출된 직선의 개수를 비교하여, 횡단 보도 유무 확인
144     횡단 보도가 있을 경우, 흰색으로 검출된 영역의 무게 중심의 카메라 좌표계 기준 좌표 publish
145     ...
146     self.frame = self.cvbridge.compressed_imgmsg_to_cv2(img, "bgr8")
147     self.cropped_image = self.imageCrop(self.frame)
148     self.thresholded_image = self.colorDetect(self.cropped_image)
149     self.edge_image = self.edgeDetect(self.thresholded_image)
150     self.houghLineDetect(self.edge_image)
151     self.crosswalk_distance = self.calcCrossWalkDistance(self.thresholded_image)
152     self.distance_pub.publish(self.crosswalk_distance)
153
154     # visualization
155     if self.viz:
156         self.visResult()
```

5. 자율주행 알고리즘

3) crosswalk_detect

```

71 # return opencv Image type
72 def imageCrop(self, _img=np.ndarray(shape=(480, 640))):
73     ...
74     원하는 이미지 영역 검출
75     ...
76     self.crop_size_x = 360
77     self.crop_size_y = 60
78     return _img[420:480, 170:530]
79
80 # return opencv Image type
81 def edgeDetect(self, _img=np.ndarray(shape=(480, 640))):
82     ...
83     이미지의 경계면 검출
84     ...
85     return cv2.Canny(_img, 0, 360)
86
87 def houghLineDetect(self, _img=np.ndarray(shape=(480, 640))):
88     ...
89     이미지의 직선 검출
90     ...
91     new_img = _img.copy()
92     self.lines = cv2.HoughLinesP(new_img, self.RHO, self.THETA * np.pi / 180, self.THRESHOLD, minLineLength=10, maxLineGap=5)
93
94     if self.lines is None:
95         # print("length is 0")
96         self.line_num = 0
97     else:
98         # print("length is {}".format(len(self.lines)))
99         self.line_num = len(self.lines)
100         for i in range(self.lines.shape[0]):
101             pt1 = (self.lines[i][0][0], self.lines[i][0][1])
102             pt2 = (self.lines[i][0][2], self.lines[i][0][3])
103             cv2.line(self.cropped_image, pt1, pt2, (0, 0, 255), 2, cv2.LINE_AA)
104
105 # return opencv Image type
106 def colorDetect(self, _img=np.ndarray(shape=(480, 640))):
107     ...
108     원하는 색 영역만 추출 (Dynamic Reconfigure를 통해, 값 변경 가능)
109     ...
110
111     hls = cv2.cvtColor(_img, cv2.COLOR_BGR2HLS)
112     mask_white = cv2.inRange(hls, self.WHITE_LANE_LOW, self.WHITE_LANE_HIGH)
113
114     return mask_white
  
```

: 경계면 검출

: 직선 검출

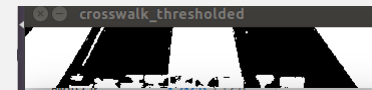
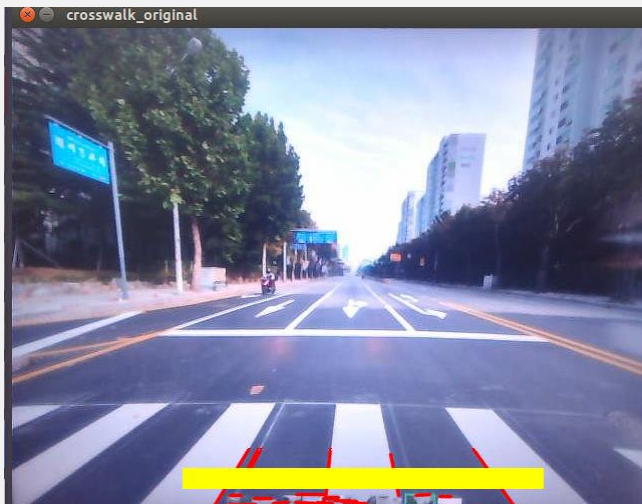
5. 자율주행 알고리즘

3) crosswalk_detect

```

71 # return opencv Image type
72 def imageCrop(self, _img=np.ndarray(shape=(480, 640))):
73     ...
74     원하는 이미지 영역 검출
75     ...
76     self.crop_size_x = 360
77     self.crop_size_y = 60
78     return _img[420:480, 170:530]
79
80 # return opencv Image type
81 def edgeDetect(self, _img=np.ndarray(shape=(480, 640))):
82     ...
83     이미지 에지 검출
84     ...
85     return cv2.cvtColor(_img, cv2.COLOR_BGR2GRAY)
86
87 def houghLineDetect(self, _img=np.ndarray(shape=(480, 640))):
88     ...
89     이미지 에지 검출
90     ...
91     new_img = cv2.cvtColor(_img, cv2.COLOR_BGR2GRAY)
92     self.lines = cv2.HoughLinesP(new_img, rho=1, theta=np.pi/180,
93                                   minLineLength=50, maxLineGap=5)
94     if self.lines is not None:
95         # print the lines
96         self.lines = self.lines[0:10]
97     else:
98         # print the error
99         self.lines = None
100     for i in range(len(self.lines)):
101         pt1 = self.lines[i][0]
102         pt2 = self.lines[i][2]
103         cv2.line(new_img, pt1, pt2, (0, 255, 0), 2)
104
105 # return opencv Image type
106 def colorDetect(self, _img=np.ndarray(shape=(480, 640))):
107     ...
108     원하는 색 영역만 추출 (Dynamic Reconfigure를 통해, 값 변경 가능)
109     ...
110
111     hls = cv2.cvtColor(_img, cv2.COLOR_BGR2HLS)
112     mask_white = cv2.inRange(hls, self.WHITE_LANE_LOW, self.WHITE_LANE_HIGH)
113
114     return mask_white
  
```

imageCrop(), colorDetect()



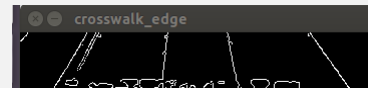
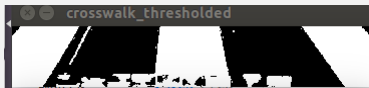
5. 자율주행 알고리즘

3) crosswalk_detect

```

71 # return opencv Image type
72 def imageCrop(self, _img=np.ndarray(shape=(480, 640))):
73     ...
74     원하는 이미지 영역 검출
75     ...
76     self.crop_size_x = 360
77     self.crop_size_y = 60
78     return _img[420:480, 170:530]
79
80 # return opencv Image type
81 def edgeDetect(self, _img=np.ndarray(shape=(480, 640))):
82     ...
83     이미지의 경계면 검출
84     ...
85     return cv2.Canny(_img, 0, 360)
86
87 def houghLineDet
88     ...
89     이미지의
90     ...
91     new_img = _i
92     self.lines =
93
94     if self.line
95         # print(
96         self.lin
97     else:
98         # print(
99         self.line_num = len(self.lines)
100         for i in range(self.lines.shape[0]):
101             pt1 = (self.lines[i][0][0], self.lines[i][0][1])
102             pt2 = (self.lines[i][0][2], self.lines[i][0][3])
103             cv2.line(self.cropped_image, pt1, pt2, (0, 0, 255), 2, cv2.LINE_AA)
104
105 # return opencv Image type
106 def colorDetect(self, _img=np.ndarray(shape=(480, 640))):
107     ...
108     원하는 색 영역만 추출 (Dynamic Reconfigure를 통해, 값 변경 가능)
109     ...
110
111     hls = cv2.cvtColor(_img, cv2.COLOR_BGR2HLS)
112     mask_white = cv2.inRange(hls, self.WHITE_LANE_LOW, self.WHITE_LANE_HIGH)
113
114     return mask_white
  
```

edgeDetect()



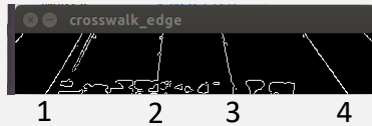
5. 자율주행 알고리즘

3) crosswalk_detect

```

71 # return opencv Image type
72 def imageCrop(self, _img=np.ndarray(shape=(480, 640))):
73     ...
74     원하는 이미지 영역 검출
75     ...
76     self.crop_size_x = 360
77     self.crop_size_y = 60
78     return _img[420:480, 170:530]
79
80 # return opencv Image type
81 def edgeDetect(self, _img=np.ndarray(shape=(480, 640))):
82     ...
83     이미지의 경계면 검출
84     ...
85     return cv2.Canny(_img, 0, 360)
86
87 def houghLineDet
88     ...
89     이미지의
90     ...
91     new_img = _i
92     self.lines =
93
94     if self.line
95         # print(
96         self.lin
97     else:
98         # print(
99         self.line_num = len(self.lines)
100         for i in range(self.lines.shape[0]):
101             pt1 = (self.lines[i][0][0], self.lines[i][0][1])
102             pt2 = (self.lines[i][0][2], self.lines[i][0][3])
103             cv2.line(self.cropped_image, pt1, pt2, (0, 0, 255), 2, cv2.LINE_AA)
104
105 # return opencv Image type
106 def colorDetect(self, _img=np.ndarray(shape=(480, 640))):
107     ...
108     원하는 색 영역만 추출 (Dynamic Reconfigure를 통해, 값 변경 가능)
109     ...
110
111     hls = cv2.cvtColor(_img, cv2.COLOR_BGR2HLS)
112     mask_white = cv2.inRange(hls, self.WHITE_LANE_LOW, self.WHITE_LANE_HIGH)
113
114     return mask_white
  
```

houghLineDetect()



self.line_num에
직선 개수 저장

5. 자율주행 알고리즘

3) crosswalk_detect

```
33     # return Int32
34     def calcCrossWalkDistance(self, _img):
35         ...
36         최종 검출된 Binary 이미지를 이용하여, 횡단 보도의 모멘트 계산
37         모멘트의 x, y 좌표 중 차량과의 거리에 해당하는 y를 반환
38         ...
39         if self.line_num >= self.CROSS_WALK_DETECT_TH:
40             try:
41                 M = cv2.moments(_img)
42                 self.x = int(M['m10']/M['m00'])
43                 self.y = int(M['m01']/M['m00'])
44             except:
45                 self.x = -1
46                 self.y = -1
47                 # print("x, y = {}, {}".format(x, y))
48                 return self.y
49         else:
50             self.x = 0
51             self.y = 0
52             return -1
```

5. 자율주행 알고리즘

3) crosswalk_detect

- crop 영역 변경
 - crosswalk_detect.py 파일 수정
- 색상 영역 변경
 - rqt_reconfigure 에서 수정 후 저장
- 직선 검출 민감도 조정
 - crosswalk_detect.py 파일의 self.line_num 조절

self.line_num	민감도
증가	낮음
감소	높음

5. 자율주행 알고리즘

4) e_stop

```

52 def lidar_callback(self, _data):
53     ...
54     실제 라이다 데이터를 받아서 동작하는 부분
55     라이다가 측정되는 각도 계산(Radian 및 Degree)
56     측정된 데이터 중, 위험 범위 안에 들어 있는 점의 수를 카운팅
57     카운팅된 점의 수가 기준 이상이면, 위험 메시지 전달
58     ...
59     cnt = 0
60     angle_rad = [_data.angle_min + i * _data.angle_increment for i, _ in enumerate(_data.ranges)]
61     angle_deg = [180 / math.pi * angle for angle in angle_rad]
62     for i, angle in enumerate(angle_deg):
63         if self.E_STOP_MIN_ANGLE_DEG <= angle <= self.E_STOP_MAX_ANGLE_DEG and 0.0 < _data.ranges[i] < self.E_STOP_DISTANCE_METER:
64             cnt += 1
65     if cnt >= self.E_STOP_COUNT:
66         if self.USE_LIFT:
67             if not self.Warning_Status:
68                 self.Warning_Status = True
69                 self.Warning_first_time = rospy.Time.now()
70                 self.warn_pub.publish("Warning")
71                 rospy.logdebug("Object Detected!! Warning!!")
72     else:
73         if self.USE_LIFT:
74             if self.Warning_Status:
75                 self.Warning_Status = False
76                 self.Safe_first_time = rospy.Time.now()
77                 self.Warning_first_time = rospy.Time.now()
78                 self.warn_pub.publish("Safe")
79                 rospy.logdebug("Safe!!")
80         if self.USE_LIFT:
81             self.ctrl_lift()
82
83 def ctrl_lift(self):
84     if rospy.Time.now().to_sec() - self.Warning_first_time.to_sec() > 5.0: # 5초 이상 정지하면 차단기 열기
85         self.lifter_ctrl_pub.publish(1)
86     elif 5.0 < rospy.Time.now().to_sec() - self.Safe_first_time.to_sec() < 6.0: # 5초 이상, 6초 이하 주행 중이면 차단기 닫기
87         self.lifter_ctrl_pub.publish(2)
  
```

: ang_deg 안의
detect 점의 개수
체크

5. 자율주행 알고리즘

3) crosswalk_detect

- self.E_STOP_COUNT
 - e_stop.py 파일 수정
- LiDAR 각도 영역 변경
 - rqt_reconfigure 에서 수정 후 저장

5. 자율주행 알고리즘

5) control

```

133 def drive_callback(self, _event):
134     ...
135     입력된 데이터를 종합하여,
136     속도 및 조향을 조절하여 최종 cmd_vel에 Publish
137     ...
138 if self.yolo_object != "green" and self.calcTimeFromDetection(self.yolo_object_last_time) > 3.0:
139     self.yolo_object = "green"
140     self.bbox_size = [0, 0]
141
142 if self.calcTimeFromDetection(self.pedestrian_stop_last_time) > 20.0:
143     self.is_pedestrian_stop_available = True
144
145 drive_data = Twist()
146 drive_data.angular.z = self.distance_to_ref * self.LATERAL_GAIN
147 rospy.loginfo("OFF_CENTER, Lateral_Gain = {}, {}".format(self.distance_to_ref, self.LATERAL_GAIN))
148 rospy.loginfo("Bbox Size = {}, Bbox_width_min = {}".format(self.bbox_size, self.PEDE_STOP_WIDTH))
149
150 try:
151     if self.e_stop == "Warning":
152         drive_data.linear.x = 0.0
153         drive_data.angular.z = 0.0
154         rospy.logwarn("Obstacle Detected, Stop!")
155
156     # elif self.crosswalk_detected:
157     #     drive_data.linear.x = 0.0
158     #     rospy.logwarn("Crosswalk Detected, Stop!")
159
160     elif self.yolo_object == "yellow" or self.yolo_object == "red":
161         drive_data.linear.x = 0.0
162         drive_data.angular.z = 0.0
163         rospy.logwarn("Traffic light is Red or Yellow, Stop!")
164
165     # elif self.crosswalk_detected and (self.yolo_object == "yellow" or self.yolo_object == "red"):
166     #     drive_data.linear.x = 0.0
167     #     drive_data.angular.z = 0.0
168     #     rospy.logwarn("Traffic light is Red or Yellow, Crosswalk Detected, Stop!")
169
170     elif self.yolo_object == "slow":
171         drive_data.linear.x = self.BASE_SPEED / 2
172         rospy.logwarn("Slow Traffic Sign Detected, Slow Down!")
173
174     elif self.yolo_object == "pedestrian" and self.is_pedestrian_stop_available and self.bbox_size[0] > self.PEDE_STOP_WIDTH:
175         drive_data.linear.x = 0.0
176         drive_data.angular.z = 0.0
177         self.is_pedestrian_stop_available = False
178         self.pedestrian_stop_last_time = rospy.Time.now().to_sec()
179         rospy.logwarn("Pedestrian Traffic Sign Detected, Stop {} Seconds!".format(self.pedestrian_stop_time))
180         rospy.sleep(rospy.Duration(self.pedestrian_stop_time))
181
182     else:
183         drive_data.linear.x = self.BASE_SPEED
184         rospy.loginfo("All Clear, Just Drive!")
185
186 if self.limo_mode == "diff":
187     self.drive_pub.publish(drive_data)
188 elif self.limo_mode == "ackermann":
189     if drive_data.linear.x == 0:
190         drive_data.angular.z = 0
191     else:
192         drive_data.angular.z = \
193             math.tan(drive_data.angular.z / 2) * drive_data.linear.x / self.LIMO_WHEELBASE
194         # 2를 나눈 것은 Differential과 GAIN비율을 맞추기 위함
195         self.drive_pub.publish(drive_data)
196
  
```

5. 자율주행 알고리즘

5) control

```

150     try:
151         if self.e_stop == "Warning":
152             drive_data.linear.x = 0.0
153             drive_data.angular.z = 0.0
154             rospy.logwarn("Obstacle Detected, Stop!")
155
156         # elif self.crosswalk_detected:
157         #     drive_data.linear.x = 0.0
158         #     rospy.logwarn("Crosswalk Detected, Stop!")
159
160         elif self.yolo_object == "yellow" or self.yolo_object == "red":
161             drive_data.linear.x = 0.0
162             drive_data.angular.z = 0.0
163             rospy.logwarn("Traffic light is Red or Yellow, Stop!")
164
165         # elif self.crosswalk_detected and (self.yolo_object == "yellow" or self.yolo_object == "red"):
166         #     drive_data.linear.x = 0.0
167         #     drive_data.angular.z = 0.0
168         #     rospy.logwarn("Traffic light is Red or Yellow, Crosswalk Detected, Stop!")
169
170         elif self.yolo_object == "slow":
171             drive_data.linear.x = self.BASE_SPEED / 2
172             rospy.logwarn("Slow Traffic Sign Detected, Slow Down!")
173
174         elif self.yolo_object == "pedestrian" and self.is_pedestrian_stop_available and self.bbox_size[0] > self.PEDE_STOP_WIDTH:
175             drive_data.linear.x = 0.0
176             drive_data.angular.z = 0.0
177             self.is_pedestrian_stop_available = False
178             self.pedestrian_stop_last_time = rospy.Time.now().to_sec()
179             rospy.logwarn("Pedestrian Traffic Sign Detected, Stop {} Seconds!".format(self.pedestrian_stop_time))
180             rospy.sleep(rospy.Duration(self.pedestrian_stop_time))
181
182         else:
183             drive_data.linear.x = self.BASE_SPEED
184             rospy.loginfo("All Clear, Just Drive!")
185
186         if self.limo_mode == "diff":
187             self.drive_pub.publish(drive_data)
188         elif self.limo_mode == "ackermann":
189             if drive_data.linear.x == 0:
190                 drive_data.angular.z = 0
191             else:
192                 drive_data.angular.z = \
193                     math.tan(drive_data.angular.z / 2) * drive_data.linear.x / self.LIMO_WHEELBASE
194             # 2를 나눈 것은 Differential과 GAIN비율을 맞추기 위한
195             self.drive_pub.publish(drive_data)

```

} : 주석 해제
 } : 주석
 } : 주석
 } : 주석

5. 자율주행 알고리즘

5) control

```

150 try:
151     if self.e_stop == "Warning":
152         drive_data.linear.x = 0.0
153         drive_data.angular.z = 0.0
154         rospy.logwarn("Obstacle Detected, Stop!")
155
156     elif self.crosswalk_detected:
157         drive_data.linear.x = 0.0
158         rospy.logwarn("Crosswalk Detected, Stop!")
159
160     # elif self.yolo_object == "yellow" or self.yolo_object == "red":
161     #     drive_data.linear.x = 0.0
162     #     drive_data.angular.z = 0.0
163     #     rospy.logwarn("Traffic light is Red or Yellow, Stop!")
164
165     # elif self.crosswalk_detected and (self.yolo_object == "yellow" or self.yolo_object == "red"):
166     #     drive_data.linear.x = 0.0
167     #     drive_data.angular.z = 0.0
168     #     rospy.logwarn("Traffic light is Red or Yellow, Crosswalk Detected, Stop!")
169
170     # elif self.yolo_object == "slow":
171     #     drive_data.linear.x = self.BASE_SPEED / 2
172     #     rospy.logwarn("Slow Traffic Sign Detected, Slow Down!")
173
174     # elif self.yolo_object == "pedestrian" and self.is_pedestrian_stop_available and self.bbox_size[0] > self.PEDE_STOP_WIDTH:
175     #     drive_data.linear.x = 0.0
176     #     drive_data.angular.z = 0.0
177     #     self.is_pedestrian_stop_available = False
178     #     self.pedestrian_stop_last_time = rospy.Time.now().to_sec()
179     #     rospy.logwarn("Pedestrian Traffic Sign Detected, Stop {} Seconds!".format(self.pedestrian_stop_time))
180     #     rospy.sleep(rospy.Duration(self.pedestrian_stop_time))
181
182     else:
183         drive_data.linear.x = self.BASE_SPEED
184         rospy.loginfo("All Clear, Just Drive!")
185
186     if self.limo_mode == "diff":
187         self.drive_pub.publish(drive_data)
188     elif self.limo_mode == "ackermann":
189         if drive_data.linear.x == 0:
190             drive_data.angular.z = 0
191         else:
192             drive_data.angular.z = \
193                 math.tan(drive_data.angular.z / 2) * drive_data.linear.x / self.LIMO_WHEELBASE
194             # 2를 나눈 것은 Differential과 GAIN비율을 맞추기 위함
195             self.drive_pub.publish(drive_data)

```

} : 장애물
 } : 횡단보도

} : 기본 상태

5. 자율주행 알고리즘

6) 실행

- 실행 (각 터미널에 별도로 실행)

- bringup

```
agilex@agilex:~$ roslaunch limo_bringup limo_start.launch
```

LIMO의 바퀴와 LiDAR 센서를 ROS 시스템에 연결

- camera

```
agilex@agilex:~$ roslaunch astra_camera dabal u3.launch
```

camera 센서를 ROS 시스템에 연결

- control

```
agilex@agilex:~$ roslaunch limo_application lane_detection.launch
```

센서 데이터를 기반으로 하는 자율주행 알고리즘 실행

QnA

E-mail : cmin87394@gmail.com

Mobile : 010-3357-8739

Thank you.

