표지판 학습 및 차선 인식을 통한 자율주행 알고리즘 개발

표지판 학습 및 차선 인식을 통한 자율주행 알고리즘 개발

https://github.com/wangjinhoon/Object-Detection-Autonomous-Driving

표지판 학습 및 차선 인식을 통한 자율주행 알고리즘 개발_리뷰.pptx

사용 언어

- 파이썬

개발 환경

- Ubuntu10.04
- ROS melodic
- 자이카 TX2 보드

사용 툴

- Pytorch
- OpenCV

*Pytorch란? 파이썬을 위한 오픈소스 딥러닝 라이브러리.

▼ Pytorch와 비슷한 딥러닝 프레임워크

1. Pytorch
   - 주로 자연어 처리.
   - 익히기 쉽고 간결.
   - 구현이 쉬움.
   - 최적화가 빠름.
   - 그래프를 만들면서 동시에 값을 할당하여 코드를 간결하게 할 수 있음.
   - GPU 연산 가능.
   - 학습 예제를 구하기 어려움.
   - 디테일한 모델링 어려움.
2. TensorFlow
   - 구글 오픈소스 소프트웨어 라이브러리.
   - 노드와 엣지로 방향성 그래프 표현.
   - 이미지 인식 등 신경망 학습에 주로 사용.
   - 대규모 예측 모델 구성에 적합.
   - 메모리 비효율적 사용.
   - 기능이 유연하지 못함.
3. 케라스
   - 텐서플로의 문제를 해결하기 위해 단순화됨.
   - API 사용 가능.
   - 학습 및 구축이 쉬움.
   - 오류가 발생했을 때 문제 파악이 어려움.
   - 문서화가 부족해 이용자 수가 적음.

프로젝트 개관

- 표지판 학습 모델을 차량에 적용하여 차선 인식을 통해 자율주행.
- 갈래길에서 만나는 표지판의 지시에 따라 좌, 우 조향.
- 주행 도중 일시 정지 표지판, 횡단 보드 표지판, 신호등을 만나면 그에 맞는 동작 실행.
- 이때 제시되는 신호등 위치는 고정, 신호는 랜덤.

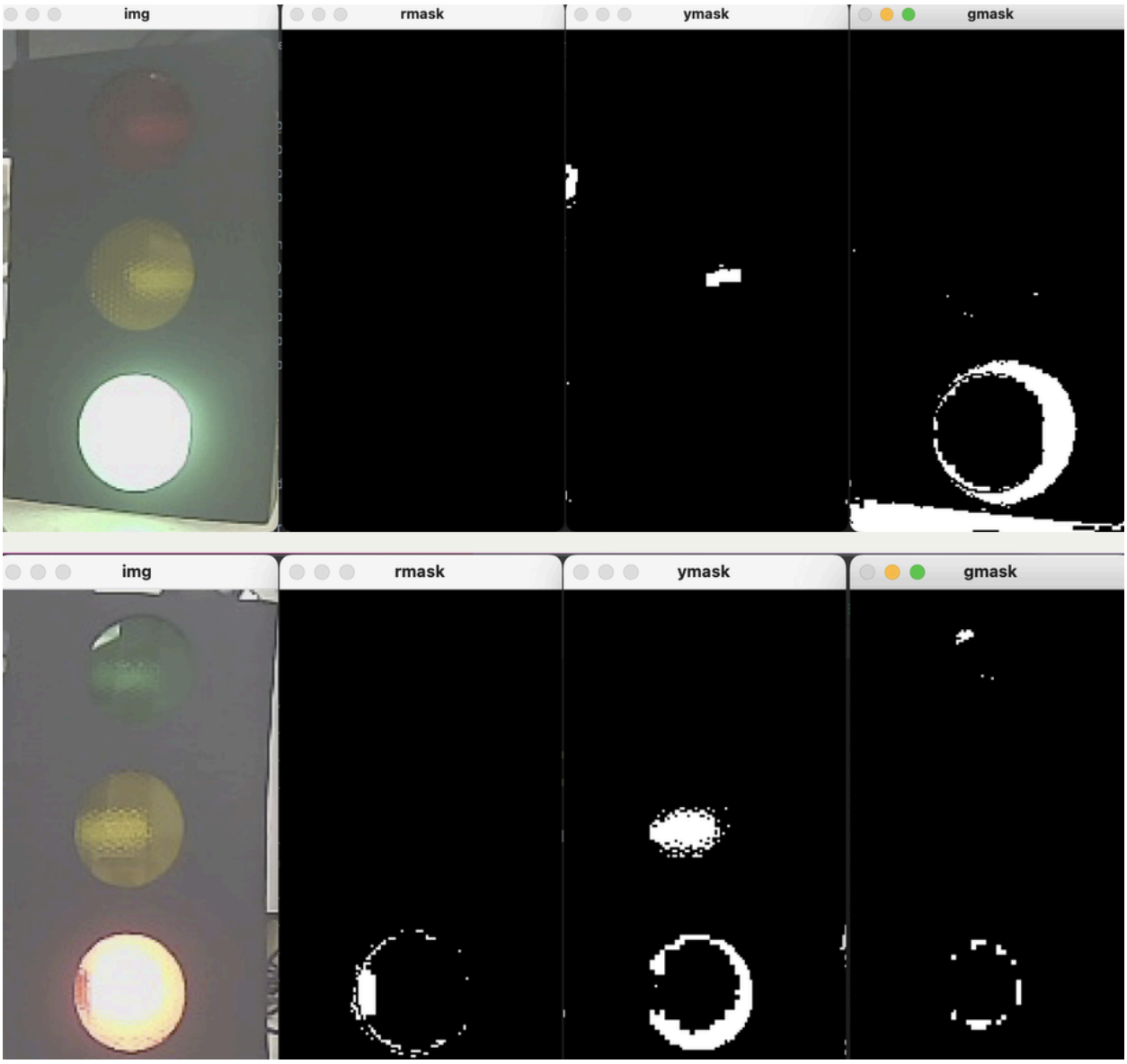
프로젝트 피드백 사항

- 신호등 밝기가 어두워 색 추출이 어려움.
- 따라서 명도차를 이용해 밝고 어두움으로 신호를 감지하도록 대체.

이미지 학습

- 좌회전 표지판 735개.
- 우회전 표지판 327개.
- 일시정지 513개.
- 횡단보도 408개.
- 신호등 240개.

신호등 검출

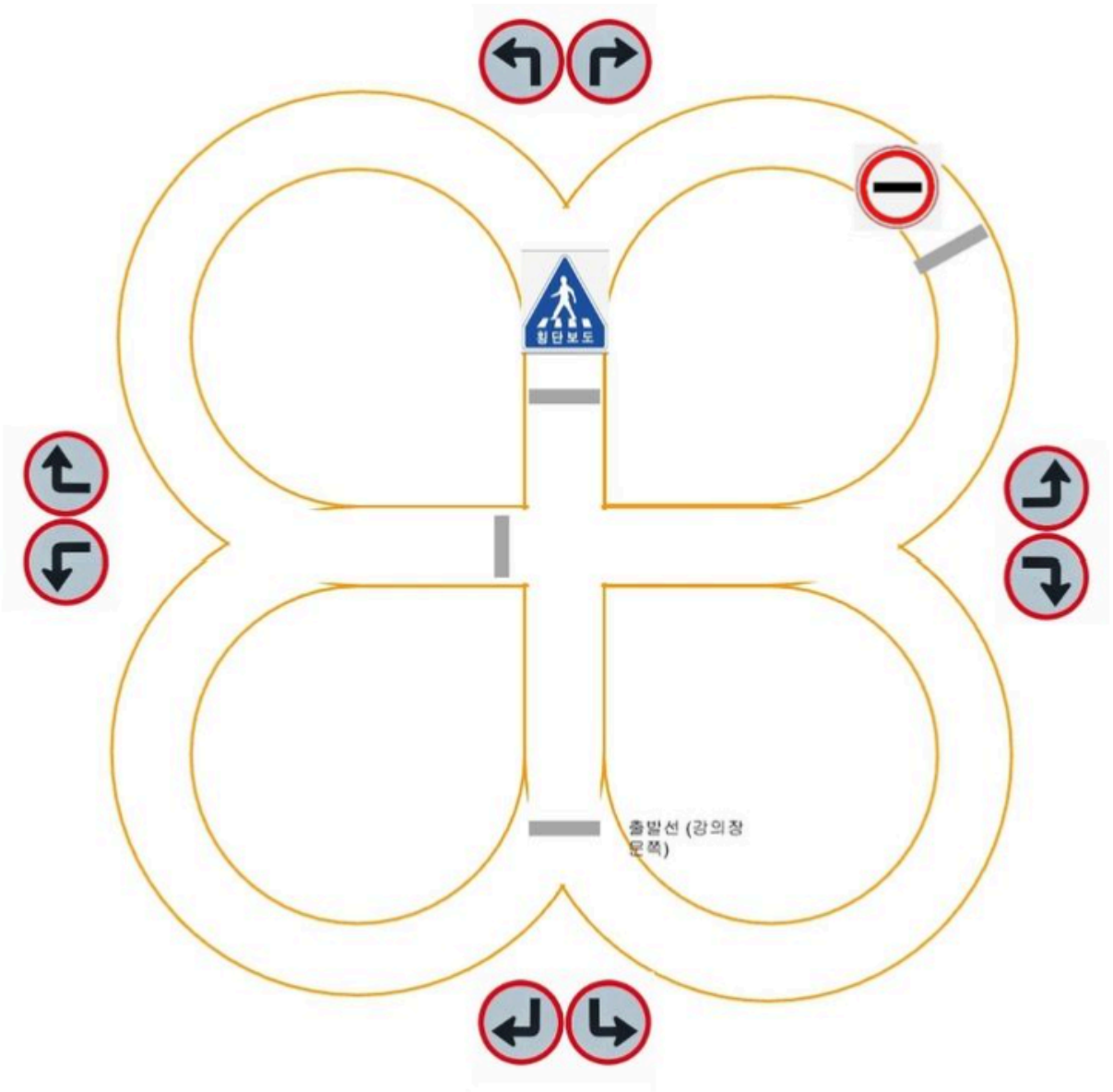


- 1차
  - HSV 색 검출.
  - 카메라 화질, 조명의 문제로 색 구분은 미약하고 밝기만 차이가 남.
- 2차
  - 명도 차이.
  - 이미지를 2진화 시킨 후, 신호등 객체를 3등분하여 초록불, 노란불, 빨간불 영역 구분.
  - 하얀색 픽셀의 개수가 가장 많은 부분을 검출.

차선 검출

- 직진 - 일반 허프 변환.
- 좌우 회전 - 표지판 객체 검출 후 회전 대기 상태.
- 정지선 - 따로 검출하지 않고 객체 인식으로 처리.

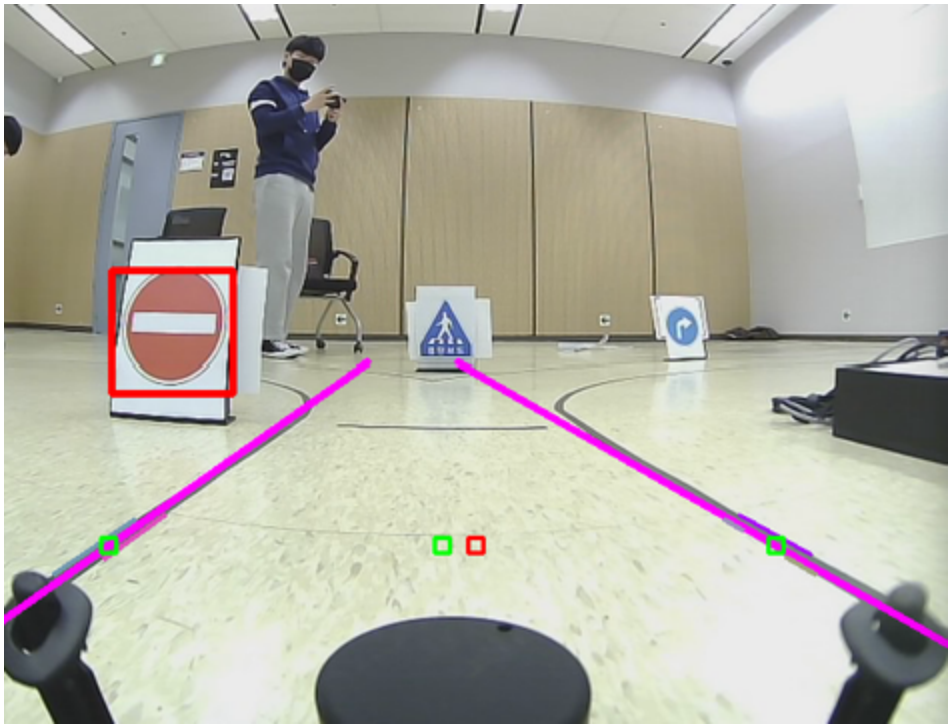
주행



- 직진
  - 일반 직진 차선, 양쪽 다 차선이 없는 사거리, 한쪽 선(=정지선)은 있는 사거리로 구분.
  - 초록불이 검출되면 몇 프레임 동안 직진.
  - 사거리의 경우 양쪽 차선이 검출되지 않으면 직진.
- 코너링
  - 회전 대기 상태에서 좌우 차선의 거리가 멀어지면 회전.
  - 반대 회전 명령이 들어올 경우 무시.

- 20 프레임 동안 조향각의 평균값이 임계값 미만이면 직진.

- 표지판 인식



- 특정 size 이상부터 처리.

코드 리뷰

▼ ego_controller.py

```
#!/usr/bin/env python

import rospy
from xycar_msgs.msg import xycar_motor


class egoController:
    def __init__(self):
        self.pub = rospy.Publisher('xycar_motor', xycar_motor)
```

```
    def stop(self):
        # print("stop")
        msg = xycar_motor()
        msg.speed = 0
        msg.angle = 0
        self.pub.publish(msg)

    def go(self, angle):
        # print("go")
        msg = xycar_motor()
        msg.speed = 5
        msg.angle = angle
        self.pub.publish(msg)
```

- ROS 패키지로, 주행 제어를 담당하는 클래스.
- 자이카 제어를 위해 `xycar_msgs.msg` 라는 메시지 유형 중 `xycar_motor` 메시지 사용.
- `xycar_motor` 메시지는 주행 속도와 핸들 각도를 포함함.
- `def __init__ (self)`
  - ROS 노드 초기화.
  - `xycar_motor` 메시지를 발행할 퍼블리셔 설정.
- `def stop (self)`
  - 자이카를 정지시키는 역할.
  - 주행 속도와 핸들 각도를 0으로 설정하여 정지 시킴.
- `def go (self, angle)`
  - 로봇을 주행시키는 역할.
  - 주행 속도를 설정하고 인자로 전달된 핸들 각도를 사용하여 자이카의 핸들을 회전.

▼ liner.py

```
import time
from abc import abstractmethod
```

```
import rospy
from sensor_msgs.msg import Image
from yolov3_trt_ros.msg import BoundingBox, BoundingBoxes
from cv_bridge import CvBridge

from PID import PID
from ego_controller import egoController


class Liner:
    def __init__(self, node_name):
        rospy.init_node(node_name)
        self.sub = rospy.Subscriber('/usb_cam/image_raw',
                                    Image, self.callback, queue_size=1)

        self.bridge = CvBridge()
        self.pid = PID(0.5, 0.0005, 0.00005)
        self.controller = egoController()
        self.controller.stop()
        time.sleep(5)

    def imgmsg2numpy(self, msg):
        return self.bridge.imgmsg_to_cv2(msg, desired_encoding='passthrough

    @abstractmethod
    def callback(self, msg):
        pass

    @abstractmethod
    def callback_itrpt(self, msg):
        pass

    @staticmethod
    def run():
        rospy.spin()
```

- 주행 제어를 위한 이미지를 처리하고 주행을 수행하는 클래스.
- 추상 클래스와 메소드를 정의하기 위해 `abstractmethod` (=abc) 라이브러리 사용.
- ROS에서 센서 데이터를 전송하기 위한 메시지 형식 `sensor_msgs.msg` 패키지를 사용하여 Image 메시지를 전송할 수 있도록 함.
- OpenCV 이미지 데이터와 ROS 이미지 메시지 간의 변환을 제공하는 CvBridge 라이브러리 사용.
- `def __init__ (self, node_name)`
  - ROS 노드 초기화.
  - 카메라 이미지를 구독하는 구독자 설정.
  - 이미지 변환을 위한 CvBridge, PID 제어기, 주행 제어기 객체를 초기화.
  - 멈춘 상태로 초기화 후, 5초간 대기하도록 함.
- `def imgmsg2numpy (self, msg)`
  - ROS 이미지 메시지를 OpenCV 이미지 형식으로 변환.
- `def callback (self, msg)`
  - 추상 메소드.
  - 이미지 메시지를 처리하는 콜백 메소드.
- `def callback_itrpt (self, msg)`
  - 추상 메소드.
  - 인터럽트를 처리하는 콜백 메소드.
- `def run ()`
  - ROS 노드를 실행.
  - 노드가 종료될 때까지 실행을 유지.
  - ROS 시스템에게 노드의 실행을 알리고, 종료되기 전까지 프로그램이 실행되도록 함.

▼ traffic_light.py

```
import cv2
from cv2 import waitKey
from cv2 import threshold

def traffic_light(gray, ract): # ract = (minx, miny, width, height)

    w_ratio = 640/416.
    h_ratio = 480/416.

    darkness = -100
    threshold_min = 70
    threshold_max = 255

    x1, y1, x2, y2 = int(ract[0] * w_ratio), int(ract[1] * h_ratio), int((ract[0]+ra
ct[2]) * w_ratio), int((ract[1] + ract[3]) * h_ratio)

    gray_dark = cv2.add(gray, darkness)

    _, gray_the = cv2.threshold(gray_dark, threshold_min, threshold_max, cv
2.THRESH_BINARY)
    bin_origin = gray_the[y1:y2, x1:x2]
    bin = gray_the[y1+int((y2-y1)*0.1):y2-int((y2-y1)*0.1), x1+int((x2-x1)*0.2):
x2-int((x2-x1)*0.2)]

    #cv2.imshow("bin", bin)
    #cv2.imshow("bin_origin", bin_origin)
    countr = 0
    county = 0
    countg = 1

    w = (x2-x1) - int((x2-x1)*0.4)
    h = (y2-y1) - int((y2-y1)*0.2)

    for y in range(0, h//3):
        for x in range(0, w):
```

```
            if bin[y,x] == 255:
                countr += 1
    for y in range(h//3, h//3 * 2):
        for x in range(0, w):
            if bin[y,x] == 255:
                county += 1
    for y in range(h//3 * 2, h):
        for x in range(0, w):
            if bin[y,x] == 255:
                countg += 1
    M = max(countr, county, countg)
    if(M == countr) : return 5
    elif(M == county) : return 6
    else : return 7
```

- 주어진 이미지에서 교통 신호등의 색을 인식하여 값을 반환.
- OpenCV에서 waitKey와 threshold 함수를 import.
- waitKey는 키보드 입력을 기다리는 함수. threshold는 이미지 이진화를 수행하는 함수.
- 기본 설정
  - `w_ratio`, `h_ratio` - 이미지 크기 변환를 위해 가로 세로 비율을 계산.
  - `darkness` - 이미지를 어둡기 하기 위한 값.
  - `threshold_min`, `threshold_max` - 이미지 이진화를 위한 임계값의 최솟값과 최댓값.
  - `x1`, `y1`, `x2`, `y2` - ROI 관심 영역 설정.
- 이미지 전처리
  - `gray_dark` - 이미지에 `darkness` 값을 적용하여 어둡게 만듦.
  - `gray_the` - 임계값을 적용하여 이진화된 이미지 생성.
  - `bin_origin`, `bin` - ROI 영역에서 얻은 원본 이진 이미지와 ROI 영역에서 잘린 이진 이미지 생성.
- 픽셀 계산

- `countr`, `county`, `countg` - 빨간색, 노란색, 초록색 픽셀의 개수 저장.
- `w`, `h` - ROI 영역의 너비와 높이 계산.
- `for문` - ROI 영역을 탐색하며 각 색상에 해당하는 픽셀 수 계산.
- `M` - 가장 많은 픽셀 수를 가지는 색상 판별 후 값(임의의 값 5, 6, 7)을 반환.

▼ trt_detection.py

```
#!/usr/bin/env python2
#
# Copyright 1993-2019 NVIDIA Corporation.  All rights reserved.
#
# NOTICE TO LICENSEE:
#
# This source code and/or documentation ("Licensed Deliverables") are
# subject to NVIDIA intellectual property rights under U.S. and
# international Copyright laws.
#
# These Licensed Deliverables contained herein is PROPRIETARY and
# CONFIDENTIAL to NVIDIA and is being provided under the terms and
# conditions of a form of NVIDIA software license agreement by and
# between NVIDIA and Licensee ("License Agreement") or electronically
# accepted by Licensee.  Notwithstanding any terms or conditions to
# the contrary in the License Agreement, reproduction or disclosure
# of the Licensed Deliverables to any third party without the express
# written consent of NVIDIA is prohibited.
#
# NOTWITHSTANDING ANY TERMS OR CONDITIONS TO THE CONTRARY
IN THE
# LICENSE AGREEMENT, NVIDIA MAKES NO REPRESENTATION ABOUT T
HE
# SUITABILITY OF THESE LICENSED DELIVERABLES FOR ANY PURPOSE.
IT IS
# PROVIDED "AS IS" WITHOUT EXPRESS OR IMPLIED WARRANTY OF AN
Y KIND.
# NVIDIA DISCLAIMS ALL WARRANTIES WITH REGARD TO THESE LICENS
```

```
ED
# DELIVERABLES, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANT
ABILITY,
# NONINFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.
# NOTWITHSTANDING ANY TERMS OR CONDITIONS TO THE CONTRARY
IN THE
# LICENSE AGREEMENT, IN NO EVENT SHALL NVIDIA BE LIABLE FOR AN
Y
# SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, OR
ANY
# DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PR
OFITS,
# WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TOR
TIOUS
# ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PER
FORMANCE
# OF THESE LICENSED DELIVERABLES.
#
# U.S. Government End Users.  These Licensed Deliverables are a
# "commercial item" as that term is defined at 48 C.F.R. 2.101 (OCT
# 1995), consisting of "commercial computer software" and "commercial
# computer software documentation" as such terms are used in 48
# C.F.R. 12.212 (SEPT 1995) and is provided to the U.S. Government
# only as a commercial end item.  Consistent with 48 C.F.R.12.212 and
# 48 C.F.R. 227.7202-1 through 227.7202-4 (JUNE 1995), all
# U.S. Government End Users acquire the Licensed Deliverables with
# only those rights set forth herein.
#
# Any use of the Licensed Deliverables in individual and commercial
# software must include, in the user documentation and internal
# comments to the code, the above Disclaimer and U.S. Government End
# Users Notice.
#

import sys, os
import time
```

```
import numpy as np
import cv2
import tensorrt as trt
from PIL import Image,ImageDraw
import rospy

from std_msgs.msg import String
from yolov3_trt_ros.msg import BoundingBox, BoundingBoxes

from cv_bridge import CvBridge
from sensor_msgs.msg import Image as Imageros

from data_processing import PreprocessYOLO, PostprocessYOLO, ALL_CA
TEGORIES
import common

from traffic_light import *

TRT_LOGGER = trt.Logger(trt.Logger.WARNING)

CFG = "/home/nvidia/xycar_ws/src/5b/src/yolov3-tiny_tstl_416.cfg"
TRT = '/home/nvidia/xycar_ws/src/5b/src/model_416_2400_epoch2450.tr
t'
NUM_CLASS = 6

bridge = CvBridge()
xycar_image = np.empty(shape=[0])
```

```
class yolov3_trt(object):
    def __init__(self):
        self.cfg_file_path = CFG
        self.num_class = NUM_CLASS
        width, height, masks, anchors = parse_cfg_wh(self.cfg_file_path)
        self.engine_file_path = TRT
        self.show_img = True
```

```
        # Two-dimensional tuple with the target network's (spatial) input reso
lution in HW ordered
        input_resolution_yolov3_WH = (width, height)
        # Create a pre-processor object by specifying the required input reso
lution for YOLOv3
        self.preprocessor = PreprocessYOLO(input_resolution_yolov3_WH)

        # Output shapes expected by the post-processor
        output_channels = (self.num_class + 5) * 3
        if len(masks) == 2:
            self.output_shapes = [(1, output_channels, height//32, width//32),
(1, output_channels, height//16, width//16)]
        else:
            self.output_shapes = [(1, output_channels, height//32, width//32),
(1, output_channels, height//16, width//16), (1, output_channels, height//8,
width//8)]

        postprocessor_args = {"yolo_masks": masks,                    # A list of 3 t
hree-dimensional tuples for the YOLO masks
                              "yolo_anchors": anchors,
                              "obj_threshold": 0.5,                               # Thres
hold for object coverage, float value between 0 and 1
                              "nms_threshold": 0.3,                                # Thre
shold for non-max suppression algorithm, float value between 0 and 1
                              "yolo_input_resolution": input_resolution_yolov3_WH,
                              "num_class": self.num_class}

        self.postprocessor = PostprocessYOLO(**postprocessor_args)

        self.engine = get_engine(self.engine_file_path)

        self.context = self.engine.create_execution_context()
```

```
        self.detection_pub = rospy.Publisher('/yolov3_trt_ros/detections', Bou
ndingBoxes, queue_size=1)
```

```
def detect(self):
        rate = rospy.Rate(10)
        image_sub = rospy.Subscriber("/usb_cam/image_raw", Imageros, img
_callback)
        while not rospy.is_shutdown():
            rate.sleep()

            # Do inference with TensorRT

            inputs, outputs, bindings, stream = common.allocate_buffers(self.e
ngine)

            # if xycar_image is empty, skip inference
            if xycar_image.shape[0] == 0:
                continue

            if self.show_img:
                cv2.imshow("show_trt",xycar_image)
                cv2.waitKey(1)

            image = self.preprocessor.process(xycar_image)
            # Store the shape of the original input image in WH format, we will
need it for later
            shape_orig_WH = (image.shape[3], image.shape[2])
            # Set host input to the image. The common.do_inference function
will copy the input to the GPU before executing.
            start_time = time.time()
            inputs[0].host = image
            trt_outputs = common.do_inference(self.context, bindings=binding
s, inputs=inputs, outputs=outputs, stream=stream)

            # Before doing post-processing, we need to reshape the outputs a
```

```
s the common.do_inference will give us flat arrays.
        trt_outputs = [output.reshape(shape) for output, shape in zip(trt_ou
tputs, self.output_shapes)]
        # Run the post-processing algorithms on the TensorRT outputs and
get the bounding box details of detected objects
        boxes, classes, scores = self.postprocessor.process(trt_outputs, sh
ape_orig_WH)

        # print("classes", type(classes), classes)
        # exit()
        maximum_size = 0
        maxbox = []
        maximum_name = -1
        maxscore = 0
        if boxes is not None:
        #     for ((top, right, bottom, left), cls, score) in zip(boxes, classes, s
cores ):
        #         hh = bottom-top
        #         ww = right-left
        #         now = hh*ww
        #         if maximum_size < now:
        #             maximum_size = now
        #             maximum_name = cls
        #             maxscore = score
        #             t,r,b,l = top, right, bottom, left
        #     #print(maximum_size)

            # Traffic Light Classificiation
            for i in range(len(boxes)):
                if classes[i] and int(classes[i]) == 5:
                    # Get image
                    #img = np.array(np.transpose(image[0], (1,2,0)) * 255, dtyp
e=np.uint8)
                    #rgb_img = np.array(Image.fromarray(img))
                    traffic_light_gray = cv2.cvtColor(xycar_image, cv2.COLOR_
BGR2GRAY)
```

```
                # 640 × 480 → 416 × 416

                # Get Traffic Light Bounding box
                #minx, miny, width, height = box

                # Traffic Light ROI
                #traffic_light_roi = traffic_light_gray[miny:height, minx:widt
h]
                traffic_id = traffic_light(traffic_light_gray, boxes[i])
                classes[i] = traffic_id

            # #boxes =  np.ndarray(maxbox)
            # print(maximum_name)
            # if maximum_name == 0 and maximum_size > 100: #left
            #     maxbox.append(t)
            #     maxbox.append(r)
            #     maxbox.append(b)
            #     maxbox.append(l)
            #     print("publish")
            #     self.publisher(maxbox, maxscore, maximum_name)

            # elif maximum_name == 1 and maximum_size > 100: #right
            #     maxbox.append(t)
            #     maxbox.append(r)
            #     maxbox.append(b)
            #     maxbox.append(l)
            #     print("publish")
            #     self.publisher(maxbox, maxscore, maximum_name)

            # elif maximum_name != 5 and maximum_size > 100: #stop
            #     maxbox.append(t)
            #     maxbox.append(r)
            #     maxbox.append(b)
            #     maxbox.append(l)
            #     self.stop_recognition = True
            #     print("publish")
```

```
            #     self.publisher(maxbox, maxscore, maximum_name)

            # elif maximum_name == 5 and maximum_size > 100: #light
            #     maxbox.append(t)
            #     maxbox.append(r)
            #     maxbox.append(b)
            #     maxbox.append(l)
            #     print("publish")
            #     self.publisher(maxbox, maxscore, maximum_name)

        latency = time.time() - start_time
        fps = 1 / latency

        #publish detected objects boxes and classes
        self.publisher(boxes, scores, classes)

        # Draw the bounding boxes onto the original input image and save i
t as a PNG file
        # print(boxes, classes, scores)
        if self.show_img:
            img_show = np.array(np.transpose(image[0], (1,2,0)) * 255, dtyp
e=np.uint8)
            obj_detected_img = draw_bboxes(Image.fromarray(img_show), b
oxes, scores, classes, ALL_CATEGORIES)
            obj_detected_img_np = np.array(obj_detected_img)
            show_img = cv2.cvtColor(obj_detected_img_np, cv2.COLOR_RG
B2BGR)
            cv2.putText(show_img, "FPS:"+str(int(fps)), (10,50),cv2.FONT_H
ERSHEY_SIMPLEX, 1,(0,255,0),2,1)
            cv2.imshow("result",show_img)
            cv2.waitKey(1)
```

```
def _write_message(self, detection_results, boxes, scores, classes):
    """ populate output message with input header and bounding boxes i
nformation """
```

```
        if boxes is None:
            return None
        for box, score, category in zip(boxes, scores, classes):
            # Populate darknet message
            minx, miny, width, height = box
            detection_msg = BoundingBox()
            detection_msg.xmin = int(minx)
            detection_msg.xmax = int(minx + width)
            detection_msg.ymin = int(miny)
            detection_msg.ymax = int(miny + height)
            detection_msg.probability = score
            detection_msg.id = int(category)
            detection_results.bounding_boxes.append(detection_msg)
        return detection_results

    def publisher(self, boxes, confs, classes):
        """ Publishes to detector_msgs
        Parameters:
        boxes (List(List(int))) : Bounding boxes of all objects
        confs (List(double))  : Probability scores of all objects
        classes  (List(int))  : Class ID of all classes
        """
        detection_results = BoundingBoxes()
        self._write_message(detection_results, boxes, confs, classes)
        self.detection_pub.publish(detection_results)


#parse width, height, masks and anchors from cfg file
def parse_cfg_wh(cfg):
    masks = []
    with open(cfg, 'r') as f:
        lines = f.readlines()
        for line in lines:
            if 'width' in line:
                w = int(line[line.find('=')+1:].replace('\n',''))
            elif 'height' in line:
```

```
            h = int(line[line.find('=')+1:].replace('\n',''))
        elif 'anchors' in line:
            anchor = line.split('=')[1].replace('\n','')
            anc = [int(a) for a in anchor.split(',')]
            anchors = [(anc[i*2], anc[i*2+1]) for i in range(len(anc) // 2)]
        elif 'mask' in line:
            mask = line.split('=')[1].replace('\n','')
            m = tuple(int(a) for a in mask.split(','))
            masks.append(m)
    return w, h, masks, anchors

def img_callback(data):
    global xycar_image
    xycar_image = bridge.imgmsg_to_cv2(data, "bgr8")

def draw_bboxes(image_raw, bboxes, confidences, categories, all_catego
ries, bbox_color='blue'):
    """Draw the bounding boxes on the original input image and return it.

    Keyword arguments:
    image_raw -- a raw PIL Image
    bboxes -- NumPy array containing the bounding box coordinates of N o
bjects, with shape (N,4).
    categories -- NumPy array containing the corresponding category for e
ach object,
    with shape (N,)
    confidences -- NumPy array containing the corresponding confidence f
or each object,
    with shape (N,)
    all_categories -- a list of all categories in the correct ordered (required f
or looking up
    the category name)
    bbox_color -- an optional string specifying the color of the bounding bo
xes (default: 'blue')
    """
    all_categories.append('traffic_light')
```

```
    all_categories.append('traffic_light')

    draw = ImageDraw.Draw(image_raw)
    if bboxes is None and confidences is None and categories is None:
        return image_raw
    for box, score, category in zip(bboxes, confidences, categories):
        x_coord, y_coord, width, height = box
        left = max(0, np.floor(x_coord + 0.5).astype(int))
        top = max(0, np.floor(y_coord + 0.5).astype(int))
        right = min(image_raw.width, np.floor(x_coord + width + 0.5).astype(i
nt))
        bottom = min(image_raw.height, np.floor(y_coord + height + 0.5).asty
pe(int))

        draw.rectangle(((left, top), (right, bottom)), outline=bbox_color)
        draw.text((left, top - 12), '{0} {1:.2f}'.format(all_categories[category],
score), fill=bbox_color)

    return image_raw

def get_engine(engine_file_path=""):
    """Attempts to load a serialized engine if available, otherwise builds a n
ew TensorRT engine and saves it."""
    if os.path.exists(engine_file_path):
        # If a serialized engine exists, use it instead of building an engine.
        print("Reading engine from file {}".format(engine_file_path))
        with open(engine_file_path, "rb") as f, trt.Runtime(TRT_LOGGER) as r
untime:
            return runtime.deserialize_cuda_engine(f.read())
    else:
        print("no trt model")
        sys.exit(1)

if __name__ == '__main__':
    yolo = yolov3_trt()
```

```
rospy.init_node('yolov3_trt_ros', anonymous=True)
yolo.detect()
```

- 주행 시스템에서 카메라를 통해 감지된 객체를 식별하고 주행 동작 수행.
- `def __init__(self)`
  - YOLOv3 Tiny 모델을 TensorRT로 변환하여 ROS 환경에서 사용할 수 있도록 함.
- `def detect(self)`
  - ROS 토픽에서 영상 수신.
  - 수신된 이미지에 대해 TensorRT를 사용하여 객체 감지 수행.
- `def _write_message(self, detection_results, boxes, scores, classes)`
  - 검출된 객체의 정보를 가지고 메시지를 작성하는 메서드.
  - 입력으로 받은 검출 결과를 이용하여 detection_results 메시지 작성, 반환.
- `def publisher(self, boxes, confs, classes)`
  - detection_pub 를 이용하여 검출 결과를 발행하는 메서드.
  - `_write_message`를 호출하여 검출 결과를 작성한 후, `detection_pub`를 통해 발행.
- `def parse_cfg_wh(cfg)`
  - YOLOv3 설정 파일에서 너비, 높이, 마스크, 앵커를 파싱.
- `def img_callback(data) (image_raw, bboxes, confidences, categories, all_categories, bbox_color='blue')`
  - ROS 이미지 메시지를 OpenCV 이미지로 변환.
- `def draw_bboxes`
  - 객체 감지 결과를 입력 이미지에 시각적으로 표시.
- `def get_engine(engine_file_path="")`
  - 저장된 TensorRT 엔진 로드.
  - 저장된 엔지 없으면 새로운 엔진 빌드.

▼ trt_drive.py

```
#!/usr/bin/env python2

import rospy, serial, time
from xycar_msgs.msg import xycar_motor
from Hough_liner import HoughLiner


# def stop(cls):
#     cls.controller.go(0)
#     pass


# def left(cls):
#     cls.controller.go(-50)
#     pass


# def right(cls):
#     cls.controller.go(50)


liner = HoughLiner("Hough_Liner")
liner.run()
```

- Hough 변환을 사용하여 차선을 탐지하고 제어하는 ROS 노드.
- HoughLiner 객체를 만들고 초기화 한 다음, run() 메서드를 호출하여 Hough 변환. 차선은 탐지하고 제어 시작.

*Hough변환이란? 기하학적 모양을 감지하는 데 사용되는 이미지 처리 기술. 이미지에 존재하는 모양을 매개변수로 변환하여 탐색할 수 있게 함. 노이즈에 강하고 끊어진 직선을 탐색할 수 있음. 회전, 이동된 직선을 감지할 수 있음.

▼ Hough_liner.py

```
from pickle import FALSE
import rospy, random, cv2, math
```

```
import numpy as np
from collections import deque
from yolov3_trt_ros.msg import BoundingBox, BoundingBoxes

from liner import Liner
```

```
class HoughLiner(Liner):

    fps = 30.0
    target_b = 128
    start = False

    prev_angles = deque([0])
    q_len = 15
    straight_thres = 10
    pos_differ_thres = 550

    turn_signal= None
    ready2turn = False
    force_turn_count = 0
    force_go_count = 0
    stop_count = 0
    ignore_count = 0

    font = cv2.FONT_HERSHEY_SIMPLEX

    def __init__(self, node_name):
        Liner.__init__(self, node_name)
        self.sub_itrpt = rospy.Subscriber('/yolov3_trt_ros/detections', Bou
ndingBoxes, self.callback_itrpt, queue_size=1)
```

- 자이카 카메라로부터 입력받은 영상을 처리하여 주행하는데 사용되는 ROS 노드 구현 클래스.
- 기본 설정

- `fps` - 초당 프레임 수.
- `target_b` - 목표 밝기 값.
- `start` - 주행 시작 플래그.
- `prev_angles` - 이전 각도를 저장하는 deque.
- `q_len` - deque의 최대 길이.
- `straight_thres` - 직진 상태를 판단하는 임계값.
- `pos_differ_thres` - 차선 감지 간격 임계값.
- `turn_signal` - 회전 신호.
- `ready2turn` - 회전할 준비 여부.
- `force_turn_count`, `force_go_count`, `stop_count`, `ignore_count` - 강제 회전, 강제 전진, 정지 및 무시 횟수.

- `def __init__ (self, node_name)`
  - /yolov3_trt_ros/detections 토픽에서 BoundingBoxes 메시지 구독. 객체 감지 정보를 제공하는 데 사용됨.

```
def callback(self, msg):
        if self.start == False:
            return

        if self.force_go_count > 0:
            print("force go")
            self.force_go_count -= 1
            self.controller.go(0)
            return
        if self.stop_count > 0:
            print("stop_count:", self.stop_count)
            self.stop_count -= 1
            self.controller.stop()
            return
        elif self.ignore_count > 0:
            self.ignore_count -= 1
```

```
        if self.force_turn_count > 0:
            print("force_turn")
            self.force_turn_count -= 1
            if self.turn_signal == 0:
                self.controller.go(-35)
            else:
                self.controller.go(40)
            return

        frame = self.imgmsg2numpy(msg)
        self.width_offset = 0
        self.width = msg.width
        self.height = msg.height
        self.offset = 310
        self.gap = 60
        self.lpos = self.width_offset
        self.rpos = self.width - self.width_offset

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        gray = cv2.GaussianBlur(gray, (0, 0), 2.0)

        low_thres = 60
        high_thres = 70

        roi = gray[self.offset:self.offset + self.gap, 0 + self.width_offset:se
lf.width - self.width_offset]
        curr_b = np.mean(roi)

        gray = gray + np.uint8(self.target_b - curr_b)

        ret, gray = cv2.threshold(gray, 145, 255, cv2.THRESH_BINARY_IN
V)

        #cv2.imshow('gray', gray)
```

```
        edge = cv2.Canny(np.uint8(gray), low_thres, high_thres)
        roi = edge[self.offset:self.offset + self.gap, 0 + self.width_offset:s
elf.width - self.width_offset]

        lines = cv2.HoughLinesP(roi, 1, math.pi / 180, 30, 30, 10)

        if lines is not None:
            left_lines, right_lines, mid = self.divide_left_right(lines)
            frame, self.lpos = self.get_line_pos(frame, left_lines, left=True)
            frame, self.rpos = self.get_line_pos(frame, right_lines, right=Tru
e)



            frame = self.draw_lines(frame, left_lines)
            frame = self.draw_lines(frame, right_lines)
            frame = self.draw_rectangle(frame)

            # if lpos not detected
            if self.lpos == self.width_offset:
                if self.rpos > self.width * 0.6:
                    angle = 0
                else:
                    angle = -40
            # if rpos not detected
            elif self.rpos == self.width - self.width_offset:
                if self.lpos < self.width * 0.4:
                    angle = 0
                else:
                    angle = 40
            # if both pos not detected
            else:
                center = (self.lpos + self.rpos) / 2
                error = (center - self.width / 2)
                if abs(self.lpos - self.rpos) < 135 or self.rpos < self.lpos:
                    if self.prev_angles[-1] > 0:
                        angle = 40
```

```
                elif self.prev_angles[-1] < 0:
                    angle = -40
                else:
                    angle = 0

            else:
                angle = self.pid.pid_control(error) * 1

        # if noise occur while on the curve, keep direction (filltering)
        if (self.turn_signal == 0 and angle > self.straight_thres) or (self.t
urn_signal == 1 and angle < -self.straight_thres):
            angle = 0
            print("force_turn")
            self.force_turn()

        # pop oldest prev_angle
        if len(self.prev_angles) >= self.q_len:
            self.prev_angles.popleft()

        # push current angle
        self.prev_angles.append(angle)

        # If xycar plan to turn left or right soon
        if self.ready2turn:
            if self.lpos != self.width_offset and self.rpos != self.width - se
lf.width_offset and self.rpos - self.lpos > self.pos_differ_thres:
                print("force_turn")
                self.force_turn()
        else:
            # check straight
            avg_angle = abs(sum(self.prev_angles)/self.q_len)
            if avg_angle < self.straight_thres:
                #print("straight_____________________")
                self.turn_signal = None

        # print("lpos: {}, rpos: {}".format(self.lpos, self.rpos))
```

```
        else:
            if self.turn_signal == 0:
                angle = -50
            elif self.turn_signal == 1:
                angle = 50
            else:
                angle = 0

        # steering
        self.controller.go(angle)



        # for Debug

        # cv2.putText(frame, "angle " + str(angle), (50, 100), font, 1, (255,
0, 0), 2)
        # cv2.putText(frame, str(self.lpos) + ", " + str(self.rpos), (50, 44
0), font, 1, (255, 0, 0), 2)
        cv2.putText(frame, "ready to turn" + str(self.ready2turn), (440, 5
0), self.font, 1, (255, 0, 0), 2)
        cv2.imshow('frame', frame)
```

- 자이카로부터 입력된 영상을 처리하고 주행 제어를 수행하는 콜백 함수.
- start 플래그가 False면 함수 종료.
- `force_go_count`, `stop_count`, `ignore_count`, `force_turn_count` 등의 변수를 확인하여 주행 동작을 수행.
  - `force_go_count`가 0보다 크면, 주행을 강제로 진행.
  - `stop_count`가 0보다 크면, 주행을 중지.
  - `ignore_count`가 0보다 크면, 메시지를 무시.
  - `force_turn_count`가 0보다 크면, 주행 방향을 강제로 변경.
- 입력된 영상을 처리하여 차선 감지. 차선에 따라 주행 각도 결정.
- `ready2turn` 플래그를 확인하여 주변 상황에 따라 주행 동작 변경.

- 결정된 주행 각도를 사용하여 주행 제어.
- 디버그 정보를 영상에 표시.

```
def callback_itrpt(self, msg):
        if self.start == False:
            self.start = True
        if self.ignore_count > 0 or len(msg.bounding_boxes) == 0:
            return

        max_size = 0
        max_class = -1

        for bounding_box in msg.bounding_boxes:
            class_id = bounding_box.id
            xmin, ymin, xmax, ymax = bounding_box.xmin, bounding_box.y
min, bounding_box.xmax, bounding_box.ymax
            size = self.get_size(xmin, ymin, xmax, ymax)
            if size > max_size:
                max_size = size
                max_class = class_id

        if (max_class == 0 or max_class == 1) and size < 3200:
            return
        elif max_class in [5, 6, 7] and size < 7000:
            return
        elif max_class in [2, 3, 4] and size < 4000:
            return

        print("class_id", max_class)

        if max_class == 0:
            if self.turn_signal is not None:
                return
            self.turn_signal = 0
            self.ready2turn = True
```

```
        elif max_class == 1:
            if self.turn_signal is not None:
                return
            self.turn_signal = 1
            self.ready2turn = True
        # stop immediately 5sec when stop, crosswalk, uturn sign detect
ed
        elif max_class == 2 or max_class == 3:
            self.stop_5sec()
        elif max_class >= 5:
            #self.force_turn_count = 0
            if max_class == 5:
                self.stop()
                self.turn_signal = None
                print("red")
            elif max_class == 6:
                print("yello")
            elif max_class == 7:
                if self.turn_signal == None:
                    self.go_now()
                    self.force_go()
                print("green")


    def draw_lines(self, img, lines):
        for line in lines:
            x1, y1, x2, y2 = line
            color = (random.randint(0, 255), random.randint(0, 255), rando
m.randint(0, 255))
            img = cv2.line(img, (x1 + self.width_offset, y1 + self.offset), (x2
+ self.width_offset, y2 + self.offset),
                           color, 2)
        return img

    def draw_rectangle(self, img):
        ccen = (self.lpos + self.rpos) / 2
```

```
        ocen = self.width / 2

        cv2.rectangle(img, (self.lpos - 5, 15 + self.offset), (self.lpos + 5, 2
5 + self.offset), (0, 255, 0), 2)
        cv2.rectangle(img, (self.rpos - 5, 15 + self.offset), (self.rpos + 5, 2
5 + self.offset), (0, 255, 0), 2)
        cv2.rectangle(img, (ccen - 5, 15 + self.offset), (ccen + 5, 25 + sel
f.offset), (0, 255, 0), 2)
        cv2.rectangle(img, (ocen - 5, 15 + self.offset), (ocen + 5, 25 + sel
f.offset), (0, 0, 255), 2)

        return img

    def divide_left_right(self, lines):
        low_grad_thres = 0
        high_grad_thres = 5

        filtered_lines = []
        left_lines = []
        right_lines = []

        max_grad = -20
        min_grad = 20
        max_x = 0
        min_x = self.width

        for line in lines:
            x1, y1, x2, y2 = line[0]
            if y2 - y1 == 0:
                grad = 0
            else:
                grad = float(x2 - x1) / float(y2 - y1)

            if (abs(grad) > low_grad_thres) and (abs(grad) < high_grad_thre
s):
                if max_grad < grad:
```

```
                max_grad = grad
            if min_grad > grad:
                min_grad = grad
            if x1 > max_x:
                max_x = x1
            if x1 < min_x:
                min_x = x1

            filtered_lines.append((line, grad))

        if max_x - min_x > 400:
            mid = (max_grad + min_grad) / 2
        else:
            mid = 0

        for line, grad in filtered_lines:
            x1, y1, x2, y2 = line[0]
            if grad < mid:
                left_lines.append(line[0].tolist())
            else:
                right_lines.append(line[0].tolist())

        return left_lines, right_lines, mid

    def get_line_params(self, lines):
        x_sum = 0.0
        y_sum = 0.0
        m_sum = 0.0

        size = len(lines)
        if not size:
            return 0, 0

        for line in lines:
            x1, y1, x2, y2 = line
```

```
            m = float(y2 - y1) / float(x2 - x1)

            x_sum += x1 + x2
            y_sum += y1 + y2
            m_sum += m

        x_mean = x_sum / (size * 2)
        y_mean = y_sum / (size * 2)
        m = m_sum / size
        b = y_mean - m * x_mean

        return m, b

    def get_line_pos(self, img, lines, left=False, right=False):
        m, b = self.get_line_params(lines)

        if m == 0 and b == 0:
            if left:
                pos = 0
            if right:
                pos = self.width
        else:
            y = self.gap / 2
            pos = int((y - b) / m)

            b += self.offset
            x1 = (self.height - b) / float(m)
            x2 = ((self.height / 2) - b) / float(m)

            cv2.line(img, (int(x1) + self.width_offset, self.height), (int(x2) +
self.width_offset, self.height / 2),
                     (255, 0, 0), 3)

        return img, pos + self.width_offset

    def force_turn(self):
```

```
            self.force_turn_count = 60
            self.ready2turn = False

            if self.turn_signal == 0:
                pass
            elif self.turn_signal == 1:
                pass
            else:
                raise Exception("self.turn_signal cannot be None")

        def force_go(self):
            self.force_go_count = 40

        def stop_5sec(self):
            self.stop_count = 7*self.fps
            # ignore msg from detector
            self.ignore_count = 7*self.fps

        def stop(self):
            self.stop_count = 7*self.fps

        def go_now(self):
            self.stop_count = 0
            #self.ignore_count = 0

        def get_size(self, xmin, ymin, xmax, ymax):
            return (xmax-xmin)*(ymax-ymin)
```

- 자이카가 표지판을 감지하고 표지판을 분류하는 콜백 함수.
- `def callback_itrpt(self, msg)`
  - `start` 플래그를 확인하여 주행 시작 혹은 정지.
  - `ignore_count` 가 0보다 크거나 메시지에 `bounding box` 가 없으면 함수 종료.
  - `bounding box` 의 크기와 클래스에 따라 주행을 계속하거나 정지, 신호에 따라 정지선에서 멈추거나 주행 진행.

- `def draw_lines(self, img, lines)`
  - 주어진 이미지에 선을 그리는 기능 수행.
  - 차선 가이드.
- `def draw_rectangle(self, img)`
  - 주어진 이미지에 사각형을 그리는 기능 수행.
  - 표지판 표시.
- `def divide_left_right(self, lines)`
  - 인식된 선들을 왼쪽과 오른쪽으로 구분하여 반환.
- `def get_line_params(self, lines)`
  - 선의 기울기와 Y절편을 계산하여 반환.
  - 선의 방향과 위치를 추정하는 데 사용.
- `def get_line_pos(self, img, lines, left=False, right=False)`
  - 주어진 이미지와 선의 좌표를 이용하여 선의 위치를 계산하고 반환.
  - 주행 경로 계산에 사용.
- `def force_go` , `stop_5sec` , `stop` , `go_now`
  - 주행 동작 구현.
- `def get_size(self, xmin, ymin, xmax, ymax)`
  - bounding box의 크기를 계산.

리뷰 후 느낀점

- PID 제어에 대한 공부를 하면 좋을 것 같음.
- ROS의 프로세스를 반드시 알아야 할 듯.