DLIP : Final Project

- subject : Implementation of emergency braking system for autonomous vehicles using deep learning object detection and sensor calibration
- name : Min-Woong Han, Seung-eun Hwang
- Final project
- Spring semester, 2023
- Class : Deep Learning Image Processing
- Instructor : prof. Young-Keun Kim

DLIP : Final Project
- **1. Introduction**
- **2. Requirements**
  - 2.1. Hardware
  - 2.2. Software installation
  - 2.3. Software installation guide
    - 2.3.1. Nvidia Graphic Driver Installation
    - 2.3.2. Cuda 11.8 installation
    - 2.3.3. cuDNN 8.6 installation
    - 2.3.4. pytorch installation
    - 2.3.5. Opencv gpu 4.6.0
    - 2.3.6. Checking
- **3. Program development**
  - 3.1. Software architecture
  - 3.2. System flowchart
  - 3.3. Sensor calibration
  - 3.6. Calibration result
  - 3.7. Model training for object detection
    - 3.7.1 Custom dataset
    - 3.7.2 Why YOLO v8?
    - 3.7.3 Performance Comparison
    - 3.7.4. Performance evaluation
  - 3.8. Object detection, calculation of relative coordinate
  - 3.9. Emergency braking system implementation
- **4. Result**
- **5. Discussion and analysis**
- **6. Appendix**

1. Introduction

In autonomous driving at a stage where human judgment and intervention are completely eliminated, the vehicle operates solely based on real-time perception information from sensors. Therefore, accurate sensor data is crucial for safe autonomous driving, enabling efficient and rapid decision-making and response. As a background for this final project, a case from Uber's autonomous driving test in 2018, where a pedestrian crossing illegally was involved in a fatal accident with the test vehicle, has been selected. Dealing with jaywalking pedestrians requires precise perception, as well as calculating the distance and direction of the objects, followed by the vehicle's ability to make its own judgment and apply emergency braking if necessary. Recognizing the importance of testing the vehicle's capability in handling such scenarios, the experiment was conducted. Throughout whole process, camera sensors were used for object perception, and 2D

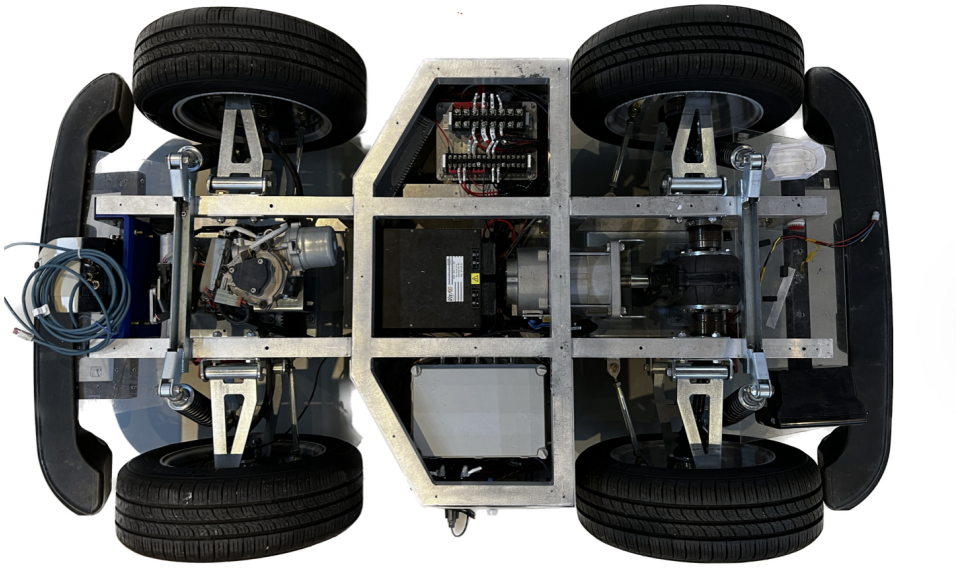
LiDAR sensors were utilized for distance measurement. Initially, YOLO V8 was employed for object detection due to its advantage of achieving high accuracy with a smaller dataset compared to YOLO V5, provided accurate labeling is performed. Subsequently, a sensor calibration issue arises when fusing the information from the 2D LiDAR and the camera, as the coordinate axes of each sensor need to be unified. To address sensor calibration, intrinsic matrices representing the internal parameters of the camera and extrinsic matrices compensating for the sensor's position and orientation differences need to be defined, and detailed processes for calculating them are provided. Once these processes are successfully executed, the world coordinates of the LiDAR are projected onto the camera's pixel coordinates. After testing and verification, the planned experiments can be conducted. The goals hoped to achieve through this project are as follows. First, to make the error of the distance extracted through calibration come within 1%, that is, to make the sensor calibration perfectly. And, to conduct a total of 10 experiments and to make all 10 times appropriately operate emergency brakes, that is, to build a 100% emergency response system. From the next part, text introduces the requirements, object recognition process, sensor calibration, and strategies for implementing an emergency braking system.

	# of test trials	Target value
Distance calculation	4	Error within 1.0[%]
Braking system test	10	100[%]

2. Requirements

2.1. Hardware

- Used platform



- Sensor Information

2D Lidar sensor	Camera sensor
	

  - 2D lidar
    - 0.25 horizontal resolution
    - 1 vertical channel (2D sensor)
    - Information available : azimuth, according distance
    - xyz coordinate (sensor centered)
    - 50 Hz sampling frequency

  - Camera (Logitech BRIO 4K)
    - Image 60Hz
    - Pixel coordinate
    - Setting : 640 x 480 image width, height
    - About 10ms ~ 15ms

  - Laptop spec
    - Windows 10 pro (64 bit)
    - Prosessor : 11th Gen Intel i7 - 11800H
    - NVIDIA GEFORCE RTX 3070

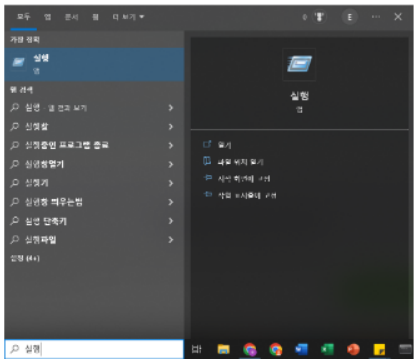
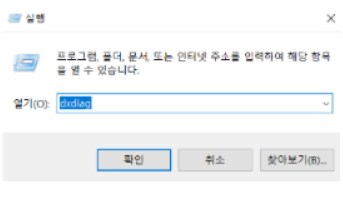
2.2. Software installation

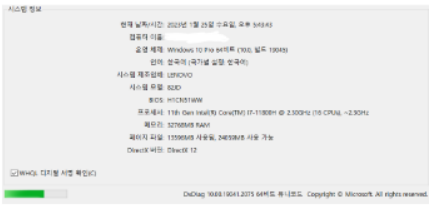
- Python : 3.10.12
- Cuda : 11.8
- CUDNN : 8.6.0 (for Cuda 11.x version)
- Pytorch : 2.0.0 + cu118
- Torchvision : 0.15.0
- Opencv gpu : 4.6.0 (build with CMAKE)
- YOLO V8

2.3. Software installation guide

2.3.1. Nvidia Graphic Driver Installation

1. Search for **Run** in the **Start** menu search bar
2. In the **Open** box, type "dxdiag".
3. CPU and windows specs

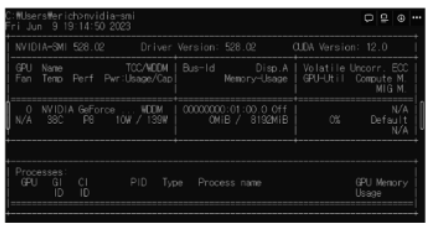





4. Check user's GPU specs
5. Go to the link below and proceed with the installation. Select the specifications of the computer checked above

- link : [NVIDIA graphic driver installation]

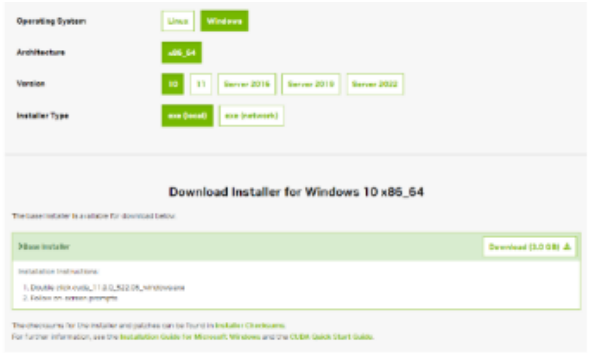
6. Check the installed graphics driver version. The cuda version on the right is the recommended cuda version, not installed version.
7. `nvidia -smi (in cmd terminal)`

2.3.2. Cuda 11.8 installation

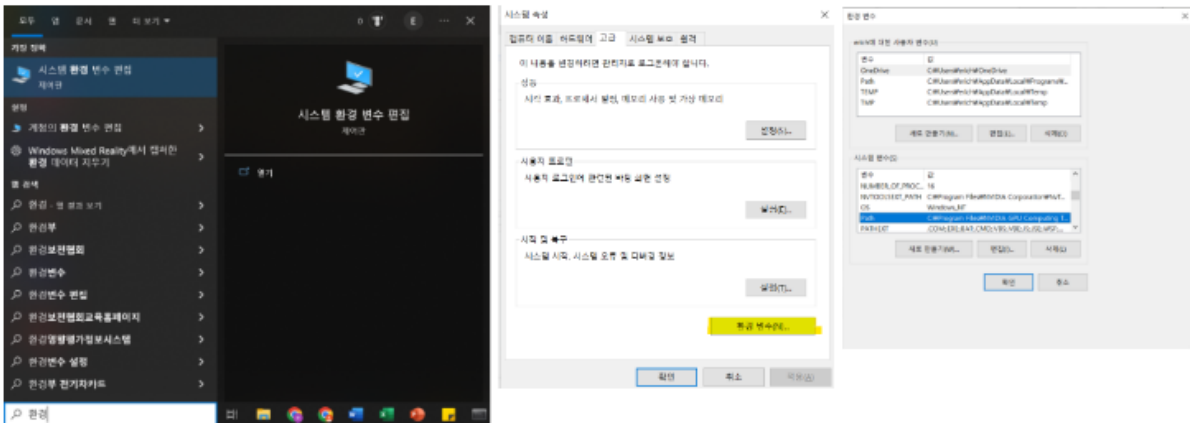
- download link : click here
- Download process and command if downloaded

```
nvcc -V   or   nvcc --version   (in cmd terminal)
```



2.3.3. cuDNN 8.6 installation

- download link : cuDNN installation
- After decompressing the zip file, copy the files inside the bin, include, and lib files, and paste them to the same file name in cuda that you have installed.
- After copying, you need to set system environment variables to recognize cuDNN.
- go into environment variables
- Select the bottom path in Environment Variables and enter Edit



- Click New and enter the following paths (if your cuda version is different, you can change it to suit your environment)

```
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.8\bin
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.8\include
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.8\lib
```

- After setting the environment variables, be sure to reboot.

2.3.4. pytorch installation

- Failure to install the appropriate version of Pytorch that is compatible with CUDA can lead to compatibility issues, such as encountering Torch backend errors and other related errors, based on my experience. Therefore, it is crucial to install the suitable versions of Torch and torchvision to ensure proper functionality.
- reference link : click here
- Command

```
pip install torch==2.0.0+cu118 torchvision==0.15.1+cu118 torchaudio==2.0.1 --index-url https://download.pytorch.org/whl/cu118
```

2.3.5. Opencv gpu 4.6.0

- download link : click here

To enhance the speed of OpenCV operations, we built OpenCV for GPU usage by employing the CMake build system. This procedure requires a detailed configuration to match specific hardware requirements, which may differ from one system to another. Those seeking to expedite their calculations are advised to follow the guide provided in the linked resource.

- Please refer to this link to get the further process :
  - link1 for reference
  - link2 for reference
  - link3 for reference

2.3.6. Checking

- For proper compatibility, results below have to be shown.
- Command

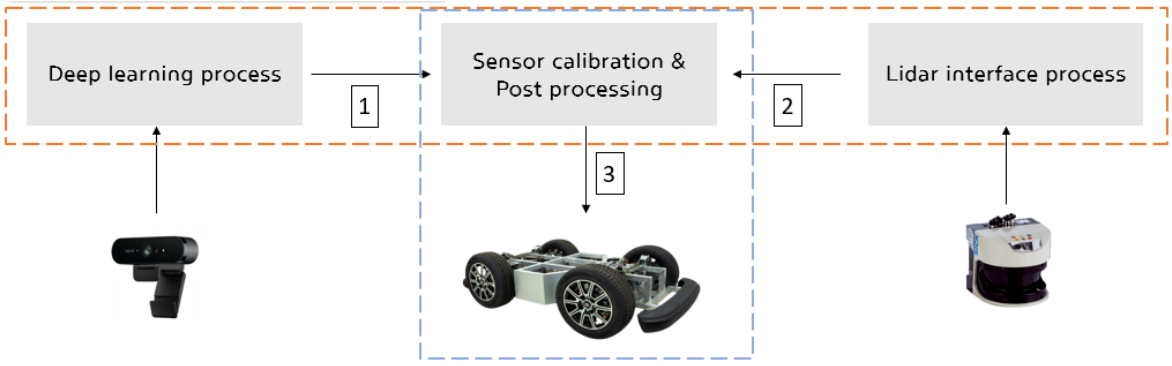
```
import torch
import torchvision
print(torch.cuda.is_available())
print(torch.torch.cuda.get_device_name())
print(torchvision.__version__)
```

```
(base) C:₩Users₩hanmu>conda activate py39

(py39) C:₩Users₩hanmu>python
Python 3.9.12 (main, Apr  4 2022, 05:22:27) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> import torchvision
>>> print(torch.cuda.is_available())
True
>>> print(torch.torch.cuda.get_device_name())
NVIDIA GeForce RTX 3070 Laptop GPU
>>> print(torchvision.__version__)
0.15.1+cu118
```

3. Program development

3.1. Software architecture

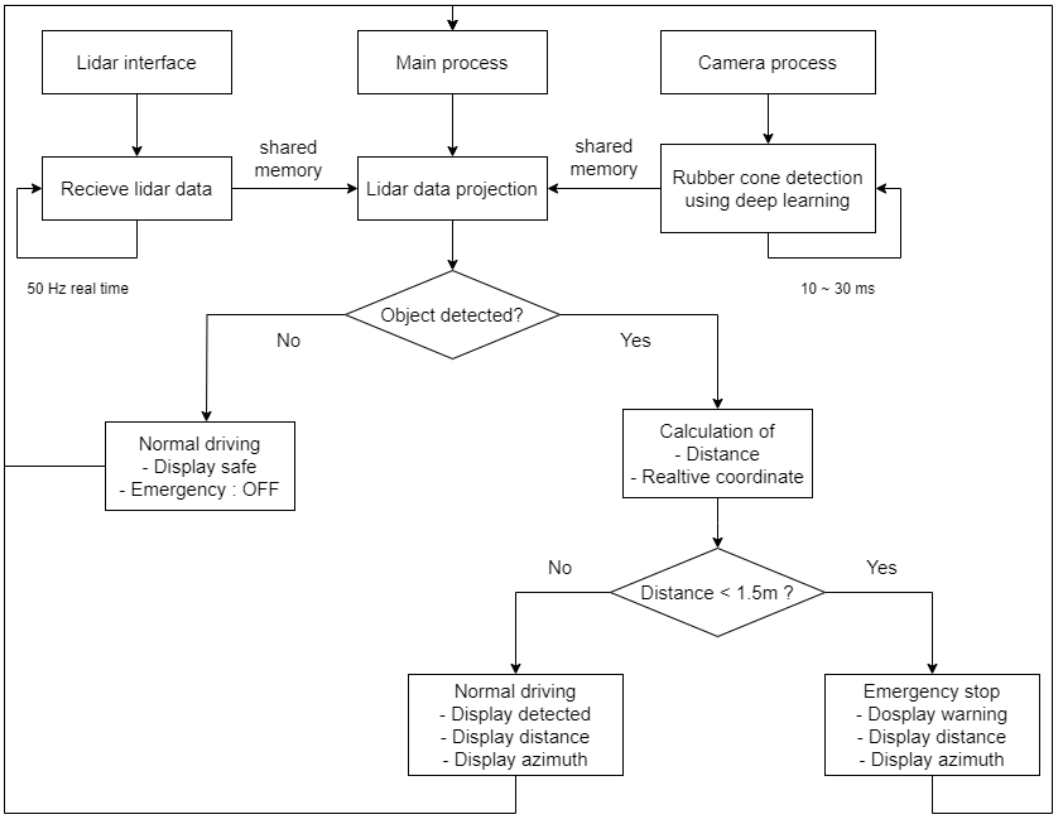
- Process 1 : Lidar interface code (C interface)
- Process 2 : YOLO object detection (Python interface)
- Process 3 : Lidar, Cam calibration & Further process (Python interface), Make commands to platform
- Communication technique for data sharing between all projects : shared memory (C to python, Python to Python)
- All process structure can be schematized as follows :



	Contents	Communication method
1	Bounding box center point of detected object	Shared memory
2	Distance and azimuth	
3	Velocity & Steering	RS232 serial communication

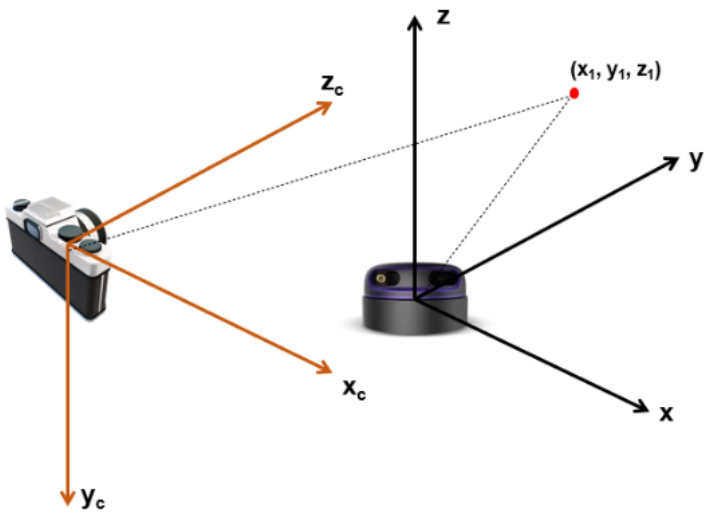
3.2. System flowchart

- Three processes run simultaneously. And each code communicates with other process with `shared memeory` method.
    1. Lidar interface process for data receiving : 50 Hz sampling frequency real time structure
    2. Main process : 50 Hz sampling frequency real time structure
    3. Camera process for deep learning : 40 ~ 70 FPS

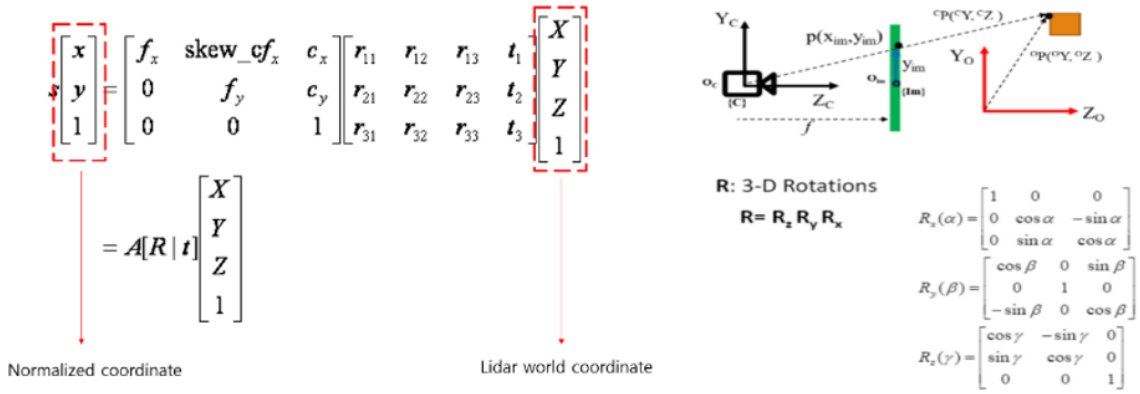


3.3. Sensor calibration

- Coordinate system of each sensors is as follows :



- The world coordinate systems of lidar and the camera are defined differently. For instance, as evident from the above Figure, the world coordinate system of the camera defines depth information along the z-axis, whereas lidar sensor defines depth information along the x-axis. Therefore, when defining the rotation matrix, it is necessary to first unify the coordinate axes that are defined differently.

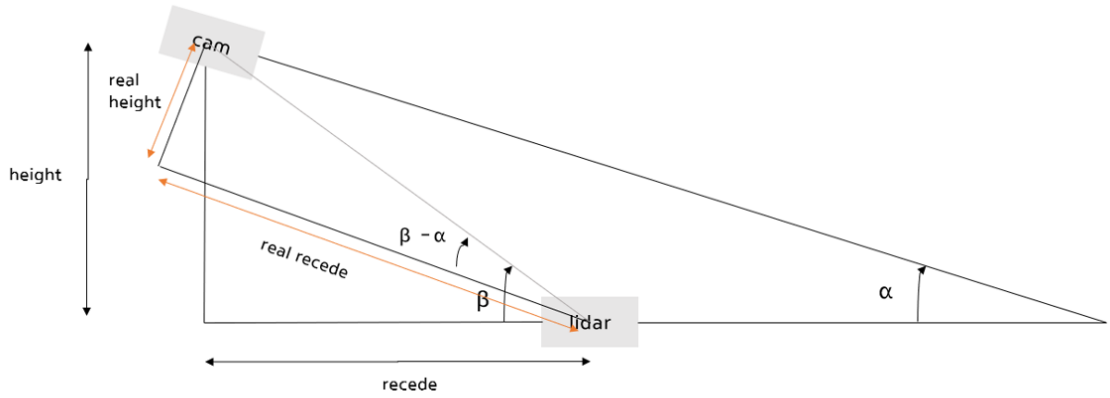


- The matrix R is a 3x3 matrix that accounts for the difference in orientation between the two sensors and provides the necessary calibration. The matrix T, on the other hand, is a 3x1 matrix that considers the physical displacement between the two sensors and provides the corresponding calibration. As for the intrinsic parameter matrix mentioned earlier, it represents the distortion correction matrix specific to the camera itself. Ultimately, the most crucial part of the calibration process is accurately determining the Extrinsic matrix (R|t), which takes into account the positional and rotational differences between the two sensors. In the case of the camera sensor, it is not facing directly forward but rather angled approximately 20 degrees below the front-facing position. Therefore, the alpha value, required to compute the rotation matrix, is set to the converted radian value of 110 degrees. As there is no deviation in the left-right orientation between the two sensors, the Beta and Gamma values can be set to 0 for the calibration process. Python code that calculates extrinsic matrix was made like below.

```
self.D2R            = pi/180
self.Alpha          = 110 * self.D2R
self.Beta           = 0 * self.D2R
self.Gamma          = 0 * self.D2R

self.rotX = np.array([[1 ,          0             ,              0        ],
                      [0 ,   np.cos(self.Alpha)   ,   -np.sin(self.Alpha) ],
                      [0 ,   np.sin(self.Alpha)   ,    np.cos(self.Alpha)
]])

self.rotY = np.array([[np.cos(self.Beta)  ,   0  ,    np.sin(self.Beta) ],
                      [    0              ,   1  ,        0             ],
                      [-np.sin(self.Beta) ,   0  ,    np.cos(self.Beta) ]])

self.rotZ = np.array([[np.cos(self.Gamma)    ,   -np.sin(self.Gamma) ,    0
],
                      [np.sin(self.Gamma)    ,   np.cos(self.Gamma)  ,    0
],
                      [    0                 ,        0              ,    1
]])

self.rotMat    = self.rotZ @ self.rotY @ self.rotX

self.transMat = np.array([[      0      ],
                          [self.realHeight],
                          [self.realRecede]])
```

```
self.M_ext = np.hstack((self.rotMat, self.transMat))

self.M_int = np.array([[self.focalLen/self.sx , 0                       ,
self.ox ],
                       [0                     , self.focalLen/self.sy   ,
self.oy ],
                       [0                     , 0                       , 1
      ]])
```

- Translation matrix that calibrates the physical locationcal difference of two sensors can be calculated as follows :

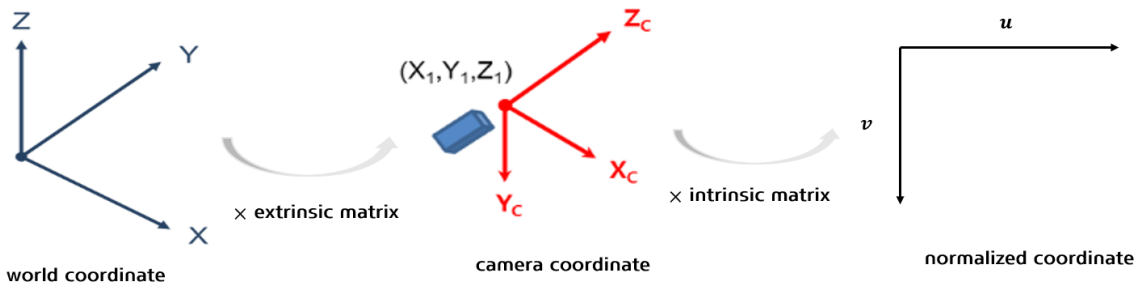


$$\alpha = 20\ [deg], \qquad \beta = \operatorname{atan}\left(\frac{height}{recede}\right), \qquad X = \sqrt{recede^2 + height^2}$$

$$real\ Height = X * \sin(\beta - \alpha)$$

$$real\ Recede = X * \cos(\beta - \alpha)$$

The calibration of the positional difference between the two sensors is achieved using a translation matrix. By applying a rotation matrix to the world coordinate system, the rotation transformation is applied. Then, the physical positional difference is corrected. In this case, both sensors are not facing directly forward. The camera sensor is angled approximately 20 degrees below the front-facing position. Therefore, instead of using the recede and height values mentioned in the diagram, the translational matrix should be defined using the real recede and real height values to complete the extrinsic matrix. All the mathematical processes are specified above.



3.6. Calibration result

As the intrinsic matrix and extrinsic matrix are multiplied with the world coordinate system, the world coordinate system is projected onto the camera's normalized coordinate system (in pixel units). According to the equation, the extrinsic matrix (Rlt) is first multiplied with the world coordinate system of the lidar. This means that a rotational transformation is applied to the lidar's world coordinate system, followed by the addition of the translational matrix to correct for the positional difference. As a result, the data defined in the lidar's coordinate system is mapped to the camera's world coordinate system by the extrinsic matrix. Finally, when the intrinsic matrix composed of the camera's internal parameters is multiplied, the data is projected onto the normalized coordinate system.

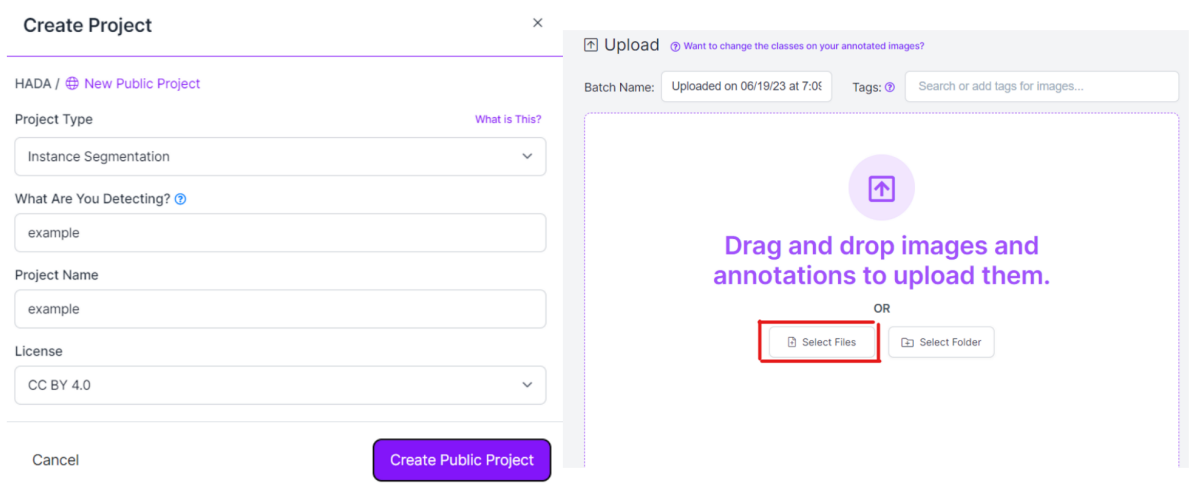
Through this entire process, the actual coordinates of the lidar are transformed into the image pixel coordinates of the camera. The figure below shows the results of projecting the 2D lidar's horizontal points onto the camera's normalized coordinate system (converted to pixel coordinates). The yellow dots represent the projected results of the lidar's actual normalized coordinate system.

3.7. Model training for object detection

3.7.1 Custom dataset

Train data was annotated by uploading the video taken directly to the roboflow.

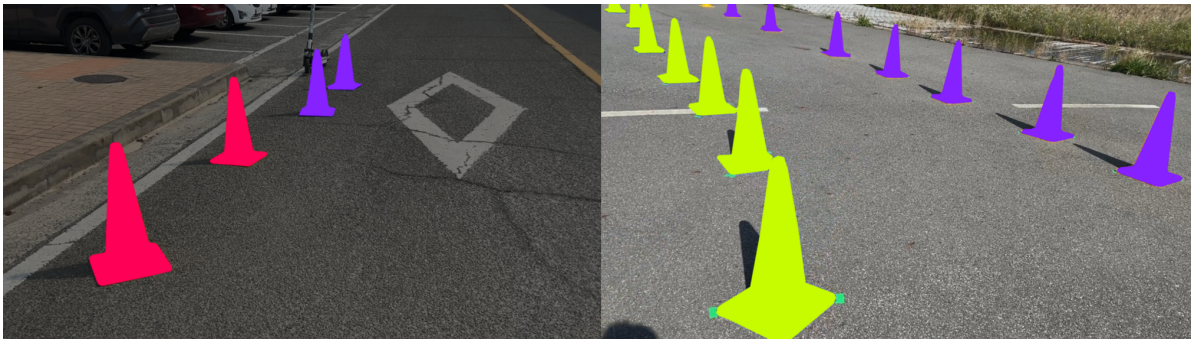
If you don't have a roboflow ID, go to this link to sign up and create a new project. Determine the type of object recognition in the project. If you select an instance segment, you can produce an annotation made of polyline. Simply write down which object it recognizes, name the project, and create a project.



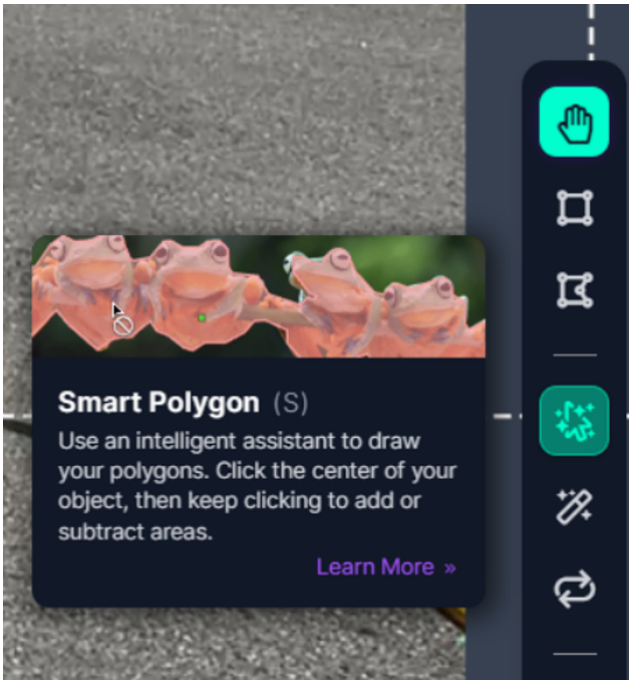
After selecting a file, select an image to be used for labeling work.

In the video to be used, it is recommended not to use images that are ambiguous to label because there are as many objects as possible or too many objects.

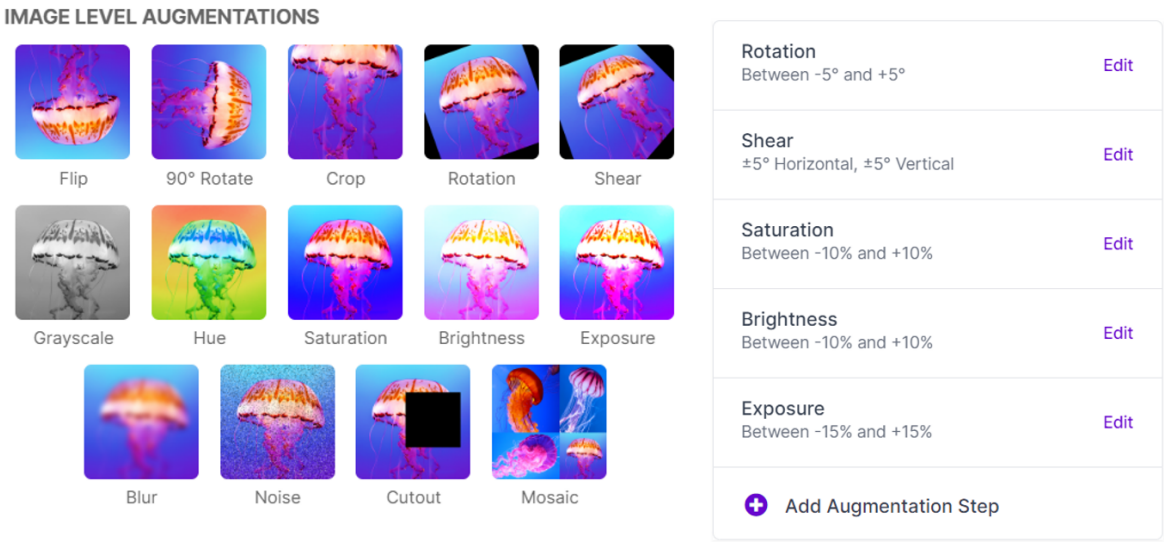
- a good source to use (left), a bad source to use (right)



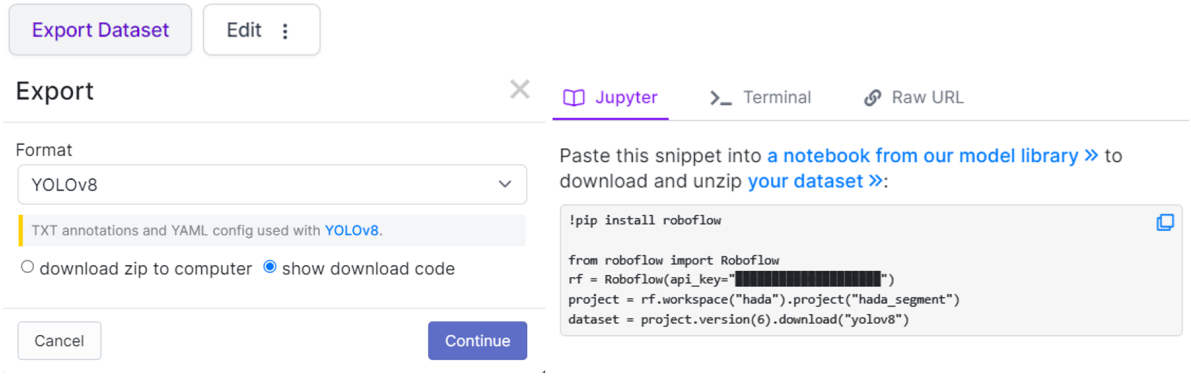
For labeling, segment annotation is performed using smart polygon



If the labeling work is completed as follows, proceed with data augmentation. By bending the same image, it looks like a picture of various environments, so you can create a dataset that becomes stronger against disturbance.



When data augmentation is completed, export data set is carried out. Dataset proceeds in the form of downloading from jupyter, so you can choose as follows. And you can paste the code shown below.



It is convenient to use a Jupiter laptop for learning. The code is placed in the source file.

When object recognition is performed, **yolov8_rubber.py** can be executed. I couldn't upload the lidar interface code, so I posted an example of a code to communicate with lidar. Running **sm_lidar.py** crosses the coordinates of the focus of object recognition through yolo.

3.7.2 Why YOLO v8?

- **YOLO v5 Methodology**

YOLO v5 utilizes 3,500 images for model training. Annotations are performed using bounding boxes which facilitate the labeling process. However, the model tends to learn the background along with the target object during deep learning. This causes a significant drop in detection accuracy if there is a difference between the training and operational environments. For rectangular objects such as signs and traffic lights, the recognition rate remains high due to their shape similarity with the bounding box. In contrast, objects like vehicles, pedestrians, and rubber

cones (as used in this experiment) are affected by the background occupying approximately two-thirds of the bounding box.

- **YOLO v5 Drawbacks**

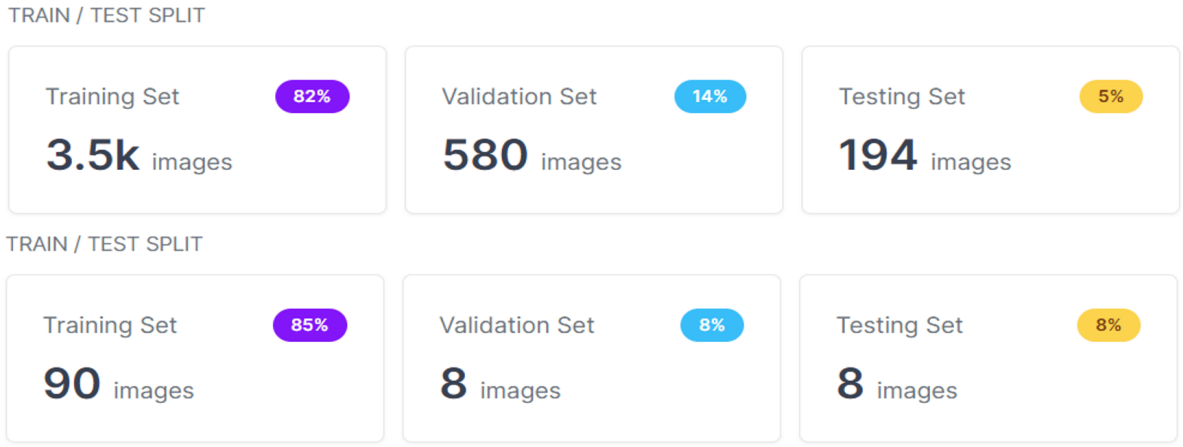
Training the model to recognize an object in diverse environments demands a large dataset. The time invested in labeling and learning increases considerably.

- **YOLO v8 Motivation for Transition**

There are three primary reasons for transitioning to YOLO v8. First, YOLO v8 exhibits commendable performance even with a smaller dataset. Second, it offers codes for converting to the ONNX structure, which is advantageous for transitioning to engine files. Third, it supports annotation types other than bounding boxes.

- **Methodology and Performance**

YOLO v8 demonstrates satisfactory performance even with smaller datasets. For instance, in training the model to recognize rubber cones, YOLO v5 required approximately 3,500 images (~1,700 images per object). YOLO v8, on the other hand, achieved an average accuracy of 0.85 to 0.9 with only about 90 images. For stabilized accuracy, the dataset was increased to approximately 400 images, resulting in an accuracy range of 0.94 to 0.96.



- **Computational Efficiency**

By utilizing engine weights that can be computed with TensorRT, YOLO v8 significantly reduces the image processing time. In autonomous vehicles, real-time object recognition using cameras is critical. While YOLO v5, using weights in the form of yolov5l.pt, shows a loop time of 30 to 50 ms, YOLO v8 achieves object recognition times as low as 10 to 20 ms (up to 80 fps) and 15 to 30 ms (at least 30 fps) when computed with yolov8l-seg.engine weight and TensorRT. This enhancement is crucial for safe vehicle operation as it minimizes the overall system time in real-time control.

- **Advanced Labeling**

Lastly, YOLO v8 facilitates learning through segmentation. Unlike bounding boxes, segmentation allows learning of the object exclusively, without the background. This contributes to increased accuracy and stable object recognition.

3.7.3 Performance Comparison

The performance metrics indicate that YOLO v5 exhibits an accuracy between 0.78 to 0.82 with a fluctuating size of the bounding box. In contrast, YOLO v8 consistently outputs a bounding box of constant size and achieves an accuracy ranging from 0.94 to 0.96.

video link : click here

	YOLO V5	YOLO V8
Training set	About 3,500 images	About 400 images
Average confidence	0.78 ~ 0.82	0.94~0.96

Transfer learning was utilized during the model training process. We used a pretrained model, yolov8l-seg.pt, and adapted it to our custom dataset. This technique allows for the leveraging of pre-existing neural networks, saving time and computational resources while also often improving the model's performance on the specific task.

3.7.4. Performance evaluation

All learning process has been completed through transfer learning, and since one of the most important elements of this project is accurate object detection, performance evaluation is essential. Using the learned model, the environment and the location of the rubber cone were different 100 times to evaluate the recognition rate of the rubber cone. The evaluation results for this are as follows. It has been confirmed that 98 out of 100 times of recognition is accurate, and 2 times of recognition is incorrect or disconnected from time to time, and that the problem occurs mainly in dark environments.

	Performance
Prediction	100
True prediction	98
False prediction or failure	2
Success rate	98 [%]

3.8. Object detection, calculation of relative coordinate

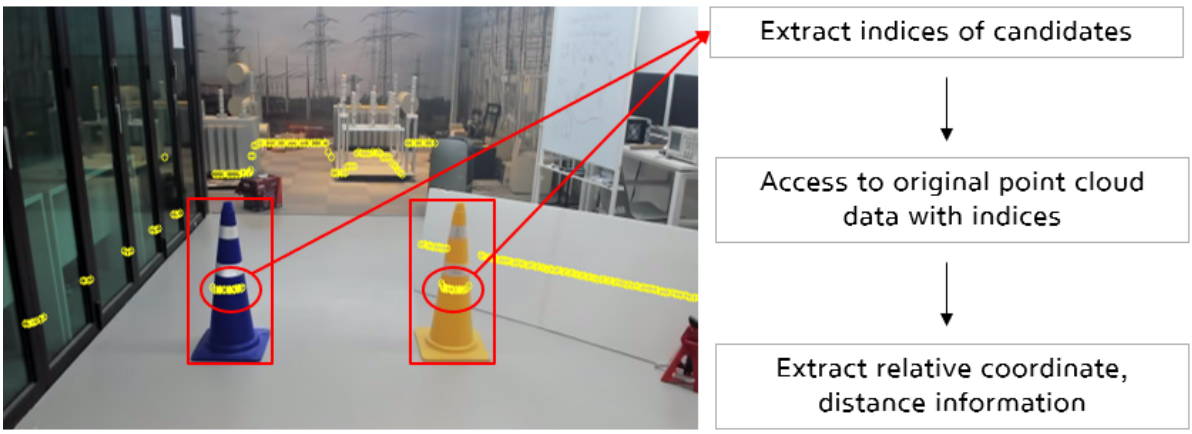
If all the previous steps have been successfully executed, the list of known information includes:

- The original world coordinates of the lidar data.
- Point cloud data projected onto the camera's normalized coordinate system.
- The centroid of the bounding box of objects detected by the camera.

The process for extracting the real relative coordinates of the detected objects using the above information is as follows:

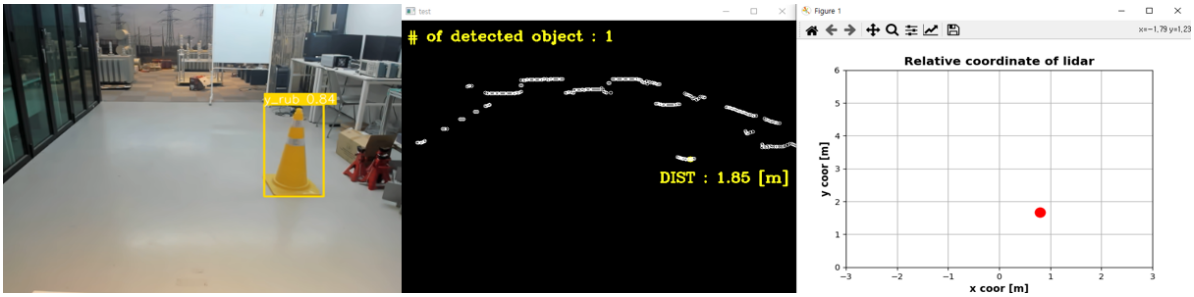
1. Define the projected point cloud data's x-coordinate as 'projection X' and the y-coordinate as 'projection Y'.
2. Extract the indices of 6 points that satisfy the following conditions and consider them as candidate points:
    - abs(projection X - bounding box centroid x-coordinate) < 15
    - (projection Y > bounding box centroid y-coordinate - 20)
3. By doing so, indices of the projection data that fall within the bounding box can be obtained. Accessing the same indices in the actual point cloud data allows user to calculate the relative coordinates and actual distance of detected object with deep learning algorithm.

All processes above enables the extraction of the relative coordinates and actual distance by matching the projection data with the point cloud data using the identified indices. The visualization of the aforementioned processes and the accuracy evaluation results for distance calculation are as follows:

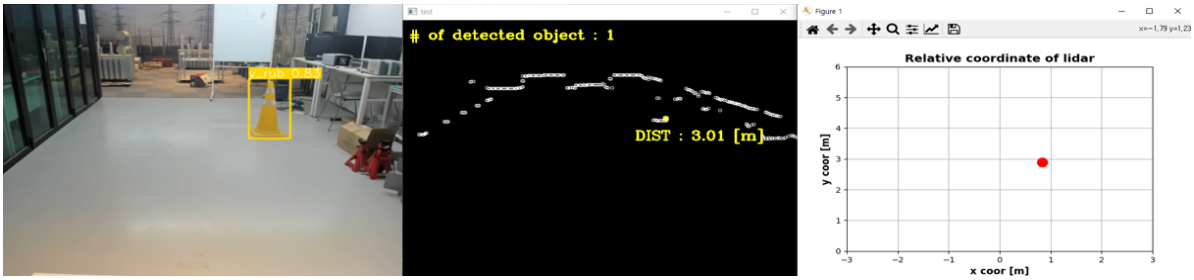


According to the described algorithm, it is possible to calculate accurate distance and relative coordinates for recognized objects. To ensure a reliable validation of the calibration, the positions of a single rubber cone were changed four times, and the distances were calculated and verified. Cases 5 and 6 demonstrate that even with multiple objects present, it is possible to calculate the distances and relative coordinates for all of them, showcasing the capability to handle multiple objects. This provides evidence for the validity of the calibration results for the two sensors.

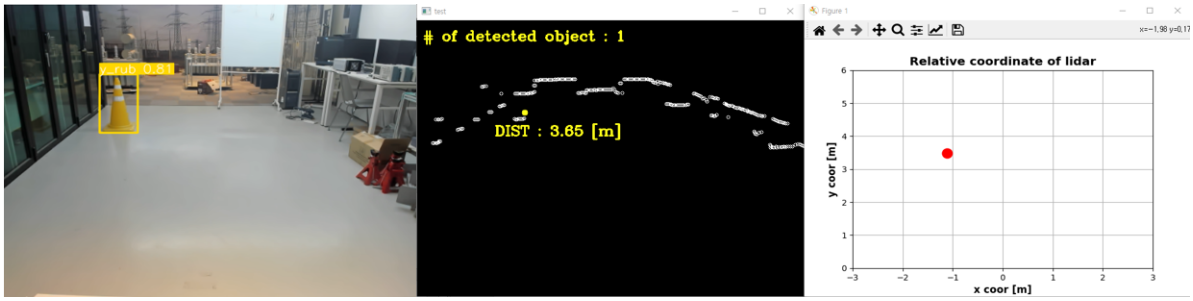
case 1



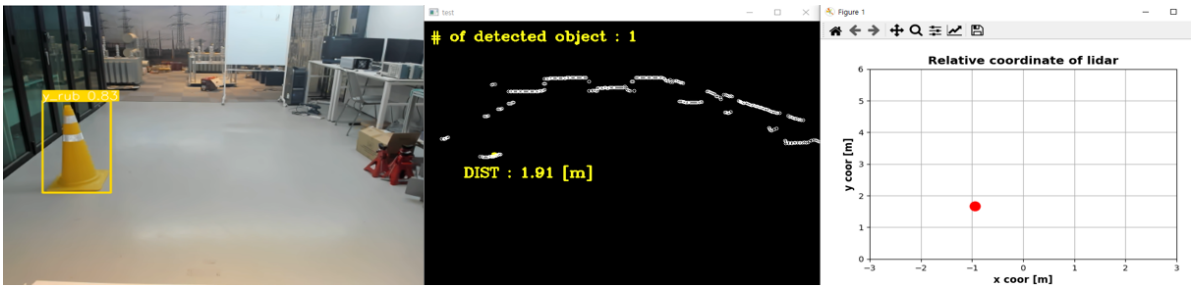
case 2



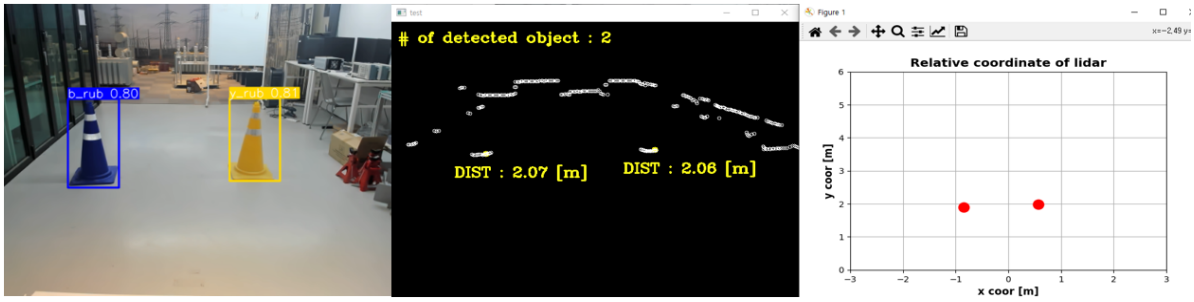
case 3



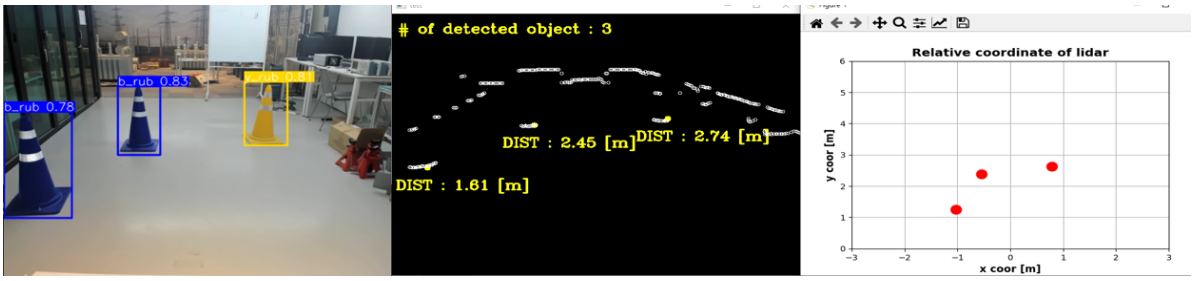
case 4



case 5



case 6



The measured distances and the distances of the actual recognized objects were quantified and summarized for cases 1 to 4. The percentile relative errors for all cases were below 1%. While calculating the relative error against the pre-calibrated values for the original point cloud data may not be highly meaningful, the fact that all distances for each case were properly calculated indicates the validity of the calibration process.

	calculated distance	real distance	Percentile relative error
Case 1	1.85 $[m]$	1.84 $[m]$	0.5 [%]
Case 2	3.01 $[m]$	3.07 $[m]$	0.2 [%]
Case 3	3.65 $[m]$	3.63 $[m]$	0.5 [%]
Case 4	1.91 $[m]$	1.90 $[m]$	0.5 [%]

3.9. Emergency braking system implementation

The purpose of this experiment was to implement a system that measures the distance and azimuth angle of sudden obstacles and responds appropriately. The vehicle operated in auto mode, relying solely on the implemented algorithm. To ensure safety during the experiment, a human was replaced with a rubber cone. The reason for training the deep learning model with rubber cone data is also for this purpose. The experiment consisted of a total of 10 trials, where the rubber cone would suddenly appear, and the system would respond accordingly. The following criteria were established for the experiment:

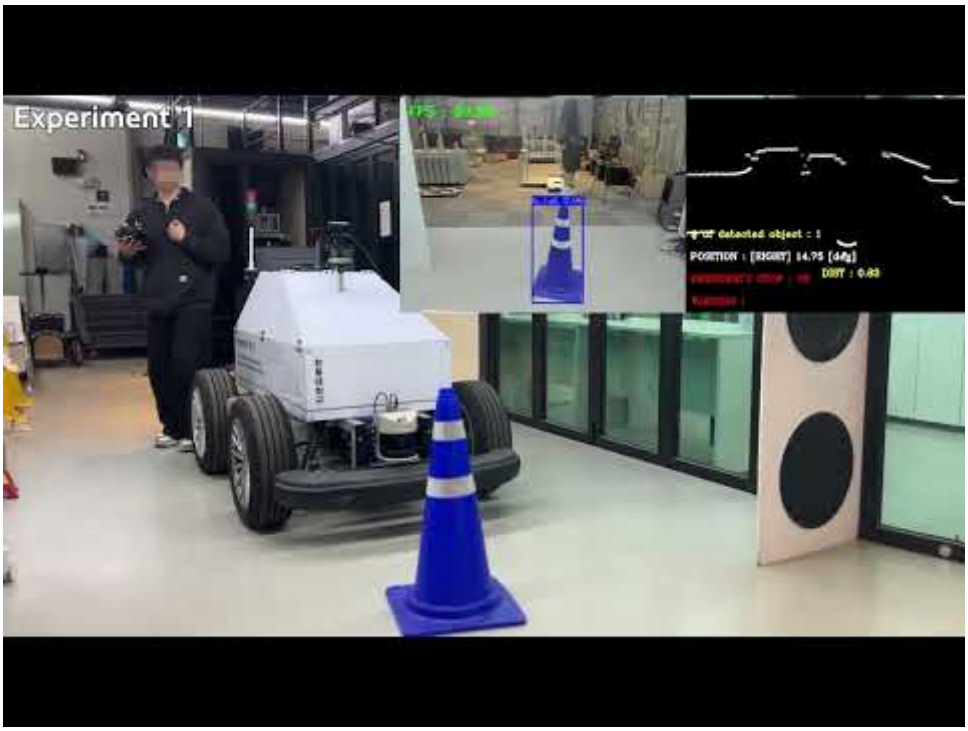
- If the rubber cone is not detected, the vehicle maintains its current speed and continues normal driving.
- If the rubber cone is detected, the system shows the number of cones detected and calculates the distance and azimuth angle for each cone.
- If the rubber cone comes within a range of 1.5m from the vehicle, the system detects it, initiates an emergency stop, and displays a warning flag to indicate the emergency braking.

The results of the experiment are presented in Section 4: Results. When the experiment was conducted for a total of 10 trials, the emergency braking system functioned appropriately in all 10 instances. Each time, the system accurately calculated the distance and azimuth angle for the obstacles.

# of experiment trials	10
Braking success	10
Braking failure	0
Success rate	100 [%]

4. Result

Demo video link : click here



5. Discussion and analysis

It's necessary to proceed with the evaluation and analysis of the implementation performance and experiments. The calibration of the two sensors was conducted to complement each other's shortcomings. While the camera sensor is capable of object recognition, it cannot calculate the distance to the recognized object. On the other hand, while the 2D LiDAR sensor is specialized in distance measurement, it doesn't have a way to identify what the object is. Therefore, the fusion of the two sensors was carried out, and calibration was an essential process for this. Through calibration, the coordinate axes were matched, and the distance to the object was calculated using the available information from both sensors, all of which converged within the targeted 1% error range. Accordingly, the verification for calibration was perfectly conducted, and we moved onto the implementation stage of the emergency braking system. We first established the criteria necessary for system construction, proceeded with the construction, and conducted a total of 10 experiments. We achieved our target of 10 out of 10 successes. In other words, it's possible to evaluate the successful construction of a stable emergency braking system using sensor calibration and deep learning, which was the ultimate goal of the project.

Since a 2D LiDAR was used, there could be a challenge when dealing with actual cyclists or pedestrian instead of rubber cones, as the points projected onto the object might or might not actually be on the person. However, this can be addressed through appropriate post-processing when 2D lidar is used. If the post-processing is unstable, it may be possible to use a 3D LiDAR to perform sensor calibration using the same process and process the data based on accurate point cloud data. The choice between these methods requires careful consideration. Using a 3D LiDAR can provide the advantage of accurate post-processing based on 3D point clouds, but it also brings the challenge of handling a large amount of point cloud data, which may slow down the sampling frequency of the main system that can cause delay of transferred commands to the platform. However, considering the trade-offs and advantages, choosing the appropriate sensor based on individual strengths and weaknesses can lead to a more accurate system implementation.

6. Appendix

- Main code is updated here & Please contact us if you need other codes for proper running !
- Main code (sensor calibration & platform control)
- Lidar code

```
# ------------------------------------------------------------------------------
---------------------
#
#   @   DLIP final project code  :   Implementation of emergency braking system
for automous vehicles
#   @   Update                   :   2023.6.11
#   @   Purpose                  :   Lidar Cam calibration
#
# ------------------------------------------------------------------------------
---------------------

from module import *
from serial_node import *
```

```
global lidarDist, azim
lidarDist  = []
azim = np.linspace(-5,185,761)


class READ_DATA(ct.Structure):

    _fields_ = [("xCoordinate",ct.c_double*761),("yCoordinate",ct.c_double*761),
("dist",ct.c_double*761),("angle",ct.c_double*761)]


class Lidar_SharedMem :

    def __init__(self):

        self.is_lidarSM = False

    def Lidar_SMopen(self) :

        self.FILE_MAP_ALL_ACCESS  = 0x000F001F
        self.FILE_MAP_READ        = 0x0004
        self.INVALID_HANDLE_VALUE = -1
        self.SHMEMSIZE            = 0x100
        self.PAGE_READWRITE       = 0x04
        self.TRUE  = 1
        self.FALSE = 0

        self.kernel32_dll               = ct.windll.kernel32
        self.msvcrt_dll                 = ct.cdll.msvcrt  # To be avoided

        self.CreateFileMapping          = self.kernel32_dll.CreateFileMappingW
        self.CreateFileMapping.argtypes = (wt.HANDLE, wt.LPVOID, wt.DWORD,
wt.DWORD, wt.DWORD, wt.LPCWSTR)
        self.CreateFileMapping.restype  = wt.HANDLE

        self.OpenFileMapping            = self.kernel32_dll.OpenFileMappingW
        self.OpenFileMapping.argtypes   = (wt.DWORD, wt.BOOL, wt.LPCWSTR)
        self.OpenFileMapping.restype    = wt.HANDLE

        self.MapViewOfFile              = self.kernel32_dll.MapViewOfFile
        self.MapViewOfFile.argtypes     = (wt.HANDLE, wt.DWORD, wt.DWORD,
wt.DWORD, ct.c_ulonglong)
        self.MapViewOfFile.restype      = wt.LPVOID

        self.memcpy                     = self.msvcrt_dll.memcpy
        self.memcpy.argtypes            = (ct.c_void_p, ct.c_void_p,
ct.c_size_t)
        self.memcpy.restype             = wt.LPVOID

        self.UnmapViewOfFile            = self.kernel32_dll.UnmapViewOfFile
        self.UnmapViewOfFile.argtypes   = (wt.LPCVOID,)
        self.UnmapViewOfFile.restype    = wt.BOOL

        self.CloseHandle                = self.kernel32_dll.CloseHandle
        self.CloseHandle.argtypes       = (wt.HANDLE,)
        self.CloseHandle.restype        = wt.BOOL
```

```
        self.GetLastError               = self.kernel32_dll.GetLastError

        self.rfile_mapping_name_ptr = ct.c_wchar_p("Lidar_smdat_ReadData")

        self.rbyte_len = ct.sizeof(READ_DATA)

        self.rmapping_handle = self.OpenFileMapping(self.FILE_MAP_ALL_ACCESS,
False, self.rfile_mapping_name_ptr)
        if not self.rmapping_handle:
            print("Could not open file mapping object:
{:d}".format(self.GetLastError()))
            raise ct.WinError()

        self.rmapped_view_ptr = self.MapViewOfFile(self.rmapping_handle,
self.FILE_MAP_ALL_ACCESS, 0, 0, self.rbyte_len)
        if not self.rmapped_view_ptr:
            print("Could not map view of file:
{:d}".format(self.GetLastError()))
            self.CloseHandle(self.rmapping_handle)
            raise ct.WinError()

        self.is_lidarSM = True

        print("Shared memory with lidar Interface program opened ...!")

    def Yolo_SMopen(self) :

        self.is_yoloSM  = True
        print("Shared memory with YOLO program opened ...!")

    def receiveDist(self):

        global lidarDist

        if self.is_lidarSM == True:

            read_smdat = READ_DATA()
            rmsg_ptr   = ct.pointer(read_smdat)
            self.memcpy(rmsg_ptr,self.rmapped_view_ptr,self.rbyte_len)
            lidarDist   = read_smdat.dist

    def sharedmemory_close(self):
        self.UnmapViewOfFile(self.wmapped_view_ptr)
        self.CloseHandle(self.wmapping_handle)
        self.UnmapViewOfFile(self.rmapped_view_ptr)
        self.CloseHandle(self.rmapping_handle)



class PROJECTION :

    def __init__(self) :

        self.colorYellow    = (25, 255, 255)
        self.colorWhite     = (255, 255, 255)
        self.colorRed       = (0, 0, 255)
        self.colorBlue      = (255, 0, 0)
```

```
        self.colorGreen     = (0, 255, 0)

        self.D2R            = pi/180
        self.R2D            = 180/pi

        self.Alphadeg       = 109.5
        self.Alpha          = self.Alphadeg * self.D2R
        self.Beta           = 0 * self.D2R
        self.Gamma          = 0 * self.D2R

        self.camHeight      = 0.93
        self.camRecede      = 0.77
        self.focalLen       = 0.00367
        self.imgWidth       = 640
        self.imgHeight      = 480
        self.fovX           = 60.92 * pi/180
        self.fovY           = 53.1432  * pi/180
        self.ox             = self.imgWidth/2

        self.oy             = self.imgHeight/2
        self.sx             = self.focalLen * math.tan(0.5 * self.fovX)/(0.5 *
self.imgWidth);
        self.sy             = self.focalLen * math.tan(0.5 * self.fovY)/(0.5 *
self.imgHeight);

        self.realHeight     = sqrt(self.camHeight**2 + self.camRecede**2) *
sin(atan(self.camHeight/self.camRecede) - (self.Alphadeg - 90) * self.D2R)
        self.realRecede     = sqrt(self.camHeight**2 + self.camRecede**2) *
cos(atan(self.camHeight/self.camRecede) - (self.Alphadeg - 90) * self.D2R)

        self.projectionX    = []
        self.projectionY    = []
        self.lidarX         = []
        self.lidarY         = []

        self.lidarxList     = []
        self.lidaryList     = []

        self.rotX = np.array([[1 ,          0            ,             0
],
                              [0 ,   np.cos(self.Alpha)   ,   -
np.sin(self.Alpha) ],
                              [0 ,   np.sin(self.Alpha)   ,
 np.cos(self.Alpha) ]])

        self.rotY = np.array([[np.cos(self.Beta)  ,   0  ,    np.sin(self.Beta)
],
                              [    0              ,   1  ,        0
],
                              [-np.sin(self.Beta) ,   0  ,    np.cos(self.Beta)
]])

        self.rotZ = np.array([[np.cos(self.Gamma)    ,   -np.sin(self.Gamma) ,
0 ],
                              [np.sin(self.Gamma)    ,   np.cos(self.Gamma)  ,
0 ],
                              [    0                 ,        0             ,
1 ]])
```

```
        self.rotMat   = self.rotZ @ self.rotY @ self.rotX


        self.transMat = np.array([[      0       ],
                                  [self.realHeight],
                                  [self.realRecede]])

        self.M_ext = np.hstack((self.rotMat, self.transMat))


        self.M_int = np.array([[self.focalLen/self.sx , 0
, self.ox ],
                               [0                     , self.focalLen/self.sy
, self.oy ],
                               [0                     , 0
, 1       ]])

        self.projectionMat = self.M_int @ self.M_ext

        self.candidates     = []
        self.rubberDist     = 0
        self.distance       = 0
        self.yoloBuffer     = []
        self.zeroCnt        = 0
        self.distVal        = 0
        self.maxDiff        = 25
        self.isyoloReady    = False
        self.steerCmd       = 0

        self.nObject        = 0
        self.azimuth        = 0
        self.objectDist     = 0
        self.isWarn         = False

        self.markerSize     = 12
        self.userFont       = cv.FONT_HERSHEY_COMPLEX

        self.Ux             = 0
        self.Uy             = 0
        self.prevsteerCmd   = 0
        self.startFlag      = 0

        self.portNum        = 'COM4'
        self.velocity       = 0     # [km/h]


    def polar2xy(self, dist, azi) :

        n = len(azi)
        x = np.zeros(n)
        y = np.zeros(n)

        for i in range(n) :

            x[i] = dist[i] * cos(azi[i] * self.D2R)
            y[i] = dist[i] * sin(azi[i] * self.D2R)
```

```
        return x, y



    def recieveYolo(self, frame) :

        shm = shared_memory.SharedMemory(name = "HADA3_CAM")
        self.yoloBuffer = np.ndarray((12,), dtype='int', buffer = shm.buf)

        self.nObject = 0

        for i in range(0, len(self.yoloBuffer), 2) :

            if(self.yoloBuffer[i] < 50) : self.yoloBuffer[i] =
self.yoloBuffer[i] - 25

            if(self.yoloBuffer[i] > 590) : self.yoloBuffer[i] =
self.yoloBuffer[i] + 25

            cv.circle(frame, (self.yoloBuffer[i], self.yoloBuffer[i+1]), 5,
self.colorYellow, -1)

        """
            - YOLO 객체 인식이 먼저 켜지면 오류가 생기는데, 방지하기 위해 startFlag를 사
용
        """

        if self.yoloBuffer[0] and self.yoloBuffer[1] == 0 :

            cv.putText(frame, "NO OBJECT DETECTED", (10,310),
cv.FONT_HERSHEY_COMPLEX, 0.7, self.colorGreen, 2)
            cv.putText(frame, "EMERGENCY STOP : OFF", (10, 410),
cv.FONT_HERSHEY_COMPLEX, 0.7, self.colorWhite, 2)
            cv.putText(frame, "SAFE !", (10, 460), cv.FONT_HERSHEY_COMPLEX, 0.7,
self.colorGreen, 2)

        else :

            self.isyoloReady = True

            if self.startFlag > 0 :

                self.lidarxList, self.lidaryList = [], []

                for Idx in range(0, len(self.yoloBuffer), 2) :

                    objectcenX, objectcenY = self.yoloBuffer[Idx],
self.yoloBuffer[Idx+1]

                    self.candidates = []

                    if objectcenX == 0 or objectcenY == 0 :
                        continue

                    for i in range(761):
```

```
                        if abs(self.projectionX[i] - objectcenX) < self.maxDiff
and self.projectionY[i] > objectcenY - 30:

                            self.candidates.append(i)

                            if len(self.candidates) == 4   :  break


                    if self.lidarY[i] > 0 :

                        self.nObject += 1

                        self.lidarxList.append(self.lidarX[i])
                        self.lidaryList.append(self.lidarY[i])

                        self.objectDist = round(sqrt(self.lidarX[i]**2 +
self.lidarY[i]**2), 2)
                        self.azimuth    = round(90 - atan2(self.lidarY[i],
self.lidarX[i]) * self.R2D, 2)

                        cv.putText(frame, f"DIST : {self.objectDist}",
(objectcenX - 50, objectcenY + 50), cv.FONT_HERSHEY_COMPLEX, 0.7,
self.colorYellow, 2)
                        # Warning flag
                        self.isWarn = True if self.objectDist < 2.0 else False


            if (self.isWarn) :

                cv.putText(frame, "EMERGENCY STOP : ON", (10, 410),
cv.FONT_HERSHEY_COMPLEX, 0.7, self.colorRed, 2)
                cv.putText(frame, "WARNING !", (10, 460),
cv.FONT_HERSHEY_COMPLEX, 0.7, self.colorRed, 2)


            else :

                cv.putText(frame, "EMERGENCY STOP : OFF", (10, 410),
cv.FONT_HERSHEY_COMPLEX, 0.7, self.colorWhite, 2)
                cv.putText(frame, "SAFE !", (10, 460), cv.FONT_HERSHEY_COMPLEX,
0.7, self.colorGreen, 2)


            if self.azimuth < 0 :
                cv.putText(frame, f"POSITION : [LEFT] {abs(self.azimuth)}
[deg]", (10, 360), cv.FONT_HERSHEY_COMPLEX, 0.7, self.colorWhite, 2)


            elif self.azimuth > 0 :
                cv.putText(frame, f"POSITION : [RIGHT] {abs(self.azimuth)}
[deg]", (10, 360), cv.FONT_HERSHEY_COMPLEX, 0.7, self.colorWhite, 2)


            cv.putText(frame, f"# of detected object : {self.nObject}",(10,310),
cv.FONT_HERSHEY_COMPLEX, 0.7, self.colorYellow, 2)
            drawnow.drawnow(self.plotLidar)

            self.startFlag += 1
```

```
    def platformControl(self) :

        # Velocity & Steer cmd
        sendCommand = erpSerial(self.portNum)
        sendCommand.send_ctrl_cmd(self.velocity, int(self.steerCmd))

        if self.isWarn == False :

            self.velocity = 6

        elif self.isWarn == True :

            self.velocity = 0


    def plotLidar(self) :

        plt.plot(self.lidarxList, self.lidaryList, 'ro', markersize =
self.markerSize)
        plt.xlabel('x coor [m]', fontsize='large', fontweight='bold')
        plt.ylabel('y coor [m]', fontsize='large', fontweight='bold')
        plt.title('Relative coordinate of lidar', fontsize='x-large',
fontweight='bold')
        plt.xlim([-3, 3])
        plt.ylim([0, 6])
        plt.grid(True)



    def lidarCamProjection(self, frame) :

        self.projectionX = []
        self.projectionY = []

        for i, dis in enumerate(lidarDist):

            lidarDist[i] = lidarDist[i] / 500

        self.lidarX , self.lidarY = self.polar2xy(lidarDist,azim)


        for i in range(len(azim)):

            pixelXY  = 0
            pointcloudXY = 0

            lx = self.lidarX[i]
            ly = self.lidarY[i]
            lz = 0
            Cz = ly + self.realRecede

            pointcloudXY = np.array([[lx],[ly],[lz],[1]])
            pixelXY = 1/Cz * self.projectionMat @ pointcloudXY

            pixelX = int(pixelXY[0])
```

```
            pixelY = int(pixelXY[1])

            self.projectionX.append(pixelX)
            self.projectionY.append(pixelY)

            cv.circle(frame, (round(pixelX) ,round(pixelY)), 3, self.colorWhite)

        # cv.imshow("test",frame)



if __name__ == "__main__" :

    fourcc   = cv.VideoWriter_fourcc(*'XVID')
    outVideo = cv.VideoWriter('./output/outputVideo2.avi', fourcc, 20.0, (640,
480))

    sim     = Lidar_SharedMem()
    project = PROJECTION()

    sim.Lidar_SMopen()
    sim.Yolo_SMopen()

    time_start = time.time()

    blackorgImg = np.zeros((480, 640, 3), dtype = np.uint8)

    while (time_stime < time_final):

        blackImg = np.copy(blackorgImg)

        # Recieve data
        sim.receiveDist()

        project.recieveYolo(blackImg)

        project.lidarCamProjection(blackImg)

        project.platformControl()

        outVideo.write(blackImg)

        cv.imshow("test",blackImg)

        if cv.waitKey(27) & 0xFF == ord('q'):

            outVideo.release()
            break

        while(1):

            time_curr = time.time()
            time_del = time_curr - time_start - time_stime

            if (time_del > time_ts) :

                time_cnt += 1
```

```
                time_stime = time_cnt*time_ts

                break


    sim.sharedmemory_close()
```

- **Train**

```
from ultralytics import YOLO
model = YOLO("yolov8l-seg.pt")
model.train(task="segment", data="Path of train data folder",
                        batch=-1,
                        imgsz=640,
                        device=0,
                        plots=True,
                        save=True,
                        epochs = 120)
```

- **Detect**

```
# ------------------------------------------------------------------------------
---------------------
#
#   @   DLIP final project code  :   Implementation of emergency braking system
for automous vehicles
#   @   Update                   :   2023.6.11
#   @   Purpose                  :   Cmaera code using yolov8
#
# ------------------------------------------------------------------------------
---------------------

import cv2
import torch
from ultralytics import YOLO
import numpy as np
import time
import tensorrt
from multiprocessing import shared_memory
from   datetime    import datetime

weight_path = 'runs\segment\\demo.pt'
model = YOLO(weight_path)

my_cam_index = 0
cap = cv2.VideoCapture(my_cam_index, cv2.CAP_DSHOW)
# print('cap: ', str(cap))

cap.set(cv2.CAP_PROP_FPS, 60.0)
cap.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter.fourcc('M','J','P','G'))
cap.set(cv2.CAP_PROP_AUTO_EXPOSURE, 0.75)
# cap.set(cv2.CAP_PROP_EXPOSURE, -11.0)
print(cap.get(cv2.CAP_PROP_EXPOSURE))
```

```
# ------------------------------------------------------------------ #

width   = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height  = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps     = int(cap.get(cv2.CAP_PROP_FPS))

print('get cam fps: ', fps)
user_font    = {0   : cv2.FONT_HERSHEY_COMPLEX,
                1   : cv2.FONT_ITALIC,
                2   : cv2.FONT_HERSHEY_DUPLEX,
                }

sm_name = "HADA3_CAM"
print("start...")
shared_memory_name = "HADA3_CAM"
shm_size = 6 * 2 * 4  # (int int) * 6개
shm = shared_memory.SharedMemory(name=shared_memory_name, create=True,
size=shm_size)
print("sm memory is open...")

# 영상 저장 initialize
fourcc = cv2.VideoWriter_fourcc(*'XVID')
output_filename   : str =
datetime.today().strftime(".\output\output_%Y_%m_%d_%H%M.mp4")  # save results
to project/name
frame_size = (width, height)
out = cv2.VideoWriter(output_filename, fourcc, fps, frame_size)

with open('counting_result.txt', 'w') as f:
    f.write('')

def key_command(_key):
            # 정지

        if _key == ord('s') :   cv2.waitKey()

        # 카메라 노출 수동 조절
        elif _key == ord('i'):
        # Get current exposure.
            exposure = cap.get(cv2.CAP_PROP_EXPOSURE)
            # Increase exposure by 1.
            cap.set(cv2.CAP_PROP_EXPOSURE, exposure + 1)

        # Decrease exposure on 'd' key press.
        elif _key == ord('d'):
            # Get current exposure.
            exposure = cap.get(cv2.CAP_PROP_EXPOSURE)
            # Decrease exposure by 1.
            cap.set(cv2.CAP_PROP_EXPOSURE, exposure - 1)

# SM memory
def send_data(shm_name, points):
    shm = shared_memory.SharedMemory(name=shm_name)
    shared_data = np.ndarray((6,2), dtype='int', buffer=shm.buf)

    # Check if the middle_points list is empty and return if it is
    if not points:
```

```
            shared_data.fill(0)
            return
        else:

            # Clear the shared_data array with zeros
            n = len(points)

            # parsing 작업
            for i in range(n):
                min_idx = i
                for j in range(i+1, n):
                    if points[j][1] < points[min_idx][1]:
                        min_idx = j
                points[i], points[min_idx] = points[min_idx], points[i]

            # Keep only the first 6 sorted middle_points
            points = points[:6]

        shared_data.fill(0)

        for i in range(len(points)):
            shared_data[i] = points[i]

colors = {0: (255, 0, 0),  # Blue for class 0
          1: (40, 188, 249),  # white for class 1
          }

cls_name = {0: "b_rub",
            1: "y_rub", }

line_width = 2
font_thick = 1

# Loop through the video frames
while cap.isOpened():

    start_time = time.time()
    prev_time = start_time

    # Read a frame from the video
    ret, frame      = cap.read()

    if ret == True: # Run YOLOv8 inference on the frame

        results = model(frame, imgsz=640)
        result = results[0]
        len_result = len(result)

        if len_result != 0: # 객체가 인식이 된다면

            middle_points = []  # sm data list 초기화

            for idx in range(len_result):

                detection = result[idx]

                box = detection.boxes.cpu().numpy()[0]
                cls = int(box.cls[0])
```

```
                xywh    = box.xywh[0].astype(int)
                centerX = xywh[0]
                centerY = xywh[1]
                area    = xywh[2] * xywh[3]

                xyxy    = box.xyxy[0].astype(int)
                x1      = xyxy[0]
                y1      = xyxy[1]
                conf    = box.conf[0]

                if area > 3000 and conf > 0.8:

                    color = colors[cls]
                    conf = box.conf[0]
                    r = box.xyxy[0].astype(int) # box

                    cv2.line(frame, (centerX, centerY), (centerX, centerY), (0,
0, 255), 4)

                    cv2.rectangle(frame, r[:2], r[2:], color,
thickness=line_width, lineType=cv2.LINE_AA)
                    label = str(cls_name[cls]) + ' ' +str(f'{ conf:.2f}')
                    text_w, text_h = cv2.getTextSize(label, 0,
fontScale=line_width/3,thickness=font_thick)[0]
                    outside = r[:2][1] - text_h >= 3
                    text_p1 = r[:2]
                    text_p2 = r[:2][0] + text_w, r[:2][1] - text_h - 3 if
outside else  r[:2][1] + text_h + 3
                    cv2.rectangle(frame, text_p1, text_p2, color, -1,
cv2.LINE_AA)  # filled

                    cv2.putText(frame, label ,
                                (text_p1[0], text_p1[1] - 2 if outside else
text_p1[1] + text_h + 2),
                                0,
                                line_width/3,
                                (255,255,255),
                                thickness=font_thick,
                                lineType=cv2.LINE_AA)

                    # shared memory에 append를 해서 넘기는 부분
                    middle_points.append((centerX, centerY))

            send_data(sm_name, middle_points)
        else:
            middle_points = []  # 0으로
            send_data(sm_name, middle_points)

        diff_time = time.time() - prev_time

        if diff_time > 0:
            fps = 1 / diff_time

        cv2.putText(frame, f'FPS : {fps:.2f}', (20, 40), user_font[2], 1, (0,
255, 0), 2)

        out.write(frame)
```

```
        cv2.imshow("mask", frame)

        key = cv2.waitKey(1) & 0xFF
        if key == ord('q')   :   break
        key_command(key)

    else:
        print("Camera is Disconnected ...!")
        break

# Release the video capture object and close the display window
shm.close()
print("sm memory close...")

cap.release()
cv2.destroyAllWindows()
```