Team4 Final Project
- Sensor fusion and visualization

조원: 김민욱, 목진우, 황혜진

멘토: 양은성

Team4: 기타등등

목차

table of contents

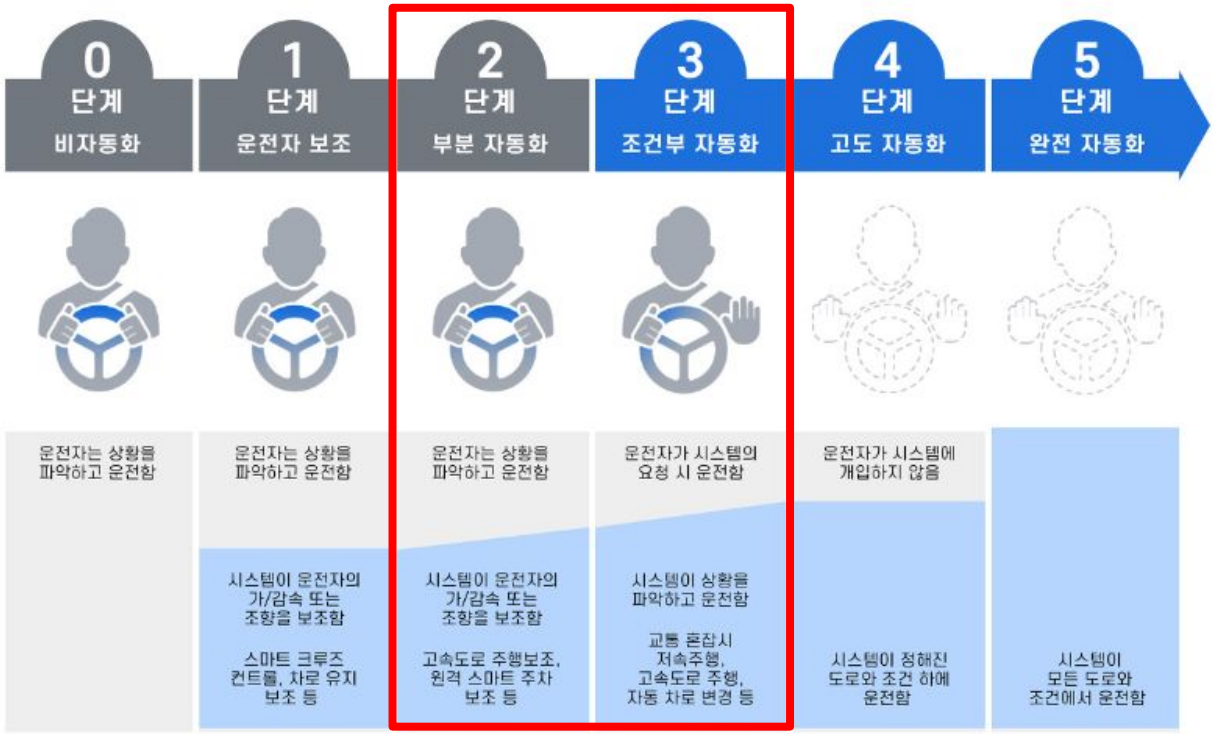
1

Project purpose

Project purpose

문제 정의

- 현재 자율주행 기술이 발달하고 있고, **고속도로**와 같은 환경에서 사용되는 **2 ~ 3** 단계의 기술들이 발전함.
- 실제 도심로의 경우 차량 및 다양한 동적 장애물들이 있어서 자율주행이 쉽지 않음.
- 동적 장애물들의 거리 및 속도를 측정하고 이에 맞게 차량을 제어할 필요가 있음.



Project purpose

프로젝트 목표

- 기하학적인 방식으로 Camera & Lidar의 Sensor Fusion을 수행하고, Vehicle Coordinate System으로 좌표를 변환하여 동적 객체 위치 인식, 속도 추정
- 구현한 ROS 프로젝트들을 통합하여 실행할 수 있는 GUI 환경 개발 (Xycar Ground Control System)
- github 브랜치 전략을 통해 팀원들 간 코드 형상관리 및 협업을 수행

프로젝트 내용	29	30	31	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Sensor Fusion						주말						설날							
- 주제 선정 및 계획 수립	■	■	■																
- 자료 조사 및 학습			■	■	■														
- Lidar Calibration								■	■	■	■								
- C++ YOLO_V3								■	■	■	■								
- Sensor fusion											■					■	■	■	■
- github 코드 형상 관리 및 협업											■					■	■	■	■
- C# 기반 UI 구성				■	■			■	■	■	■					■	■	■	■

Project purpose

팀원 역할 소개

김민욱	C#을 통한 Xycar data 시각화 및 동작 구현.
목진우	Project management, YOLO C++, sensor fusion.
황혜진	Sensor (lidar, camera, VCS) extrinsic calibration, sensor fusion, velocity estimation.

개발 환경

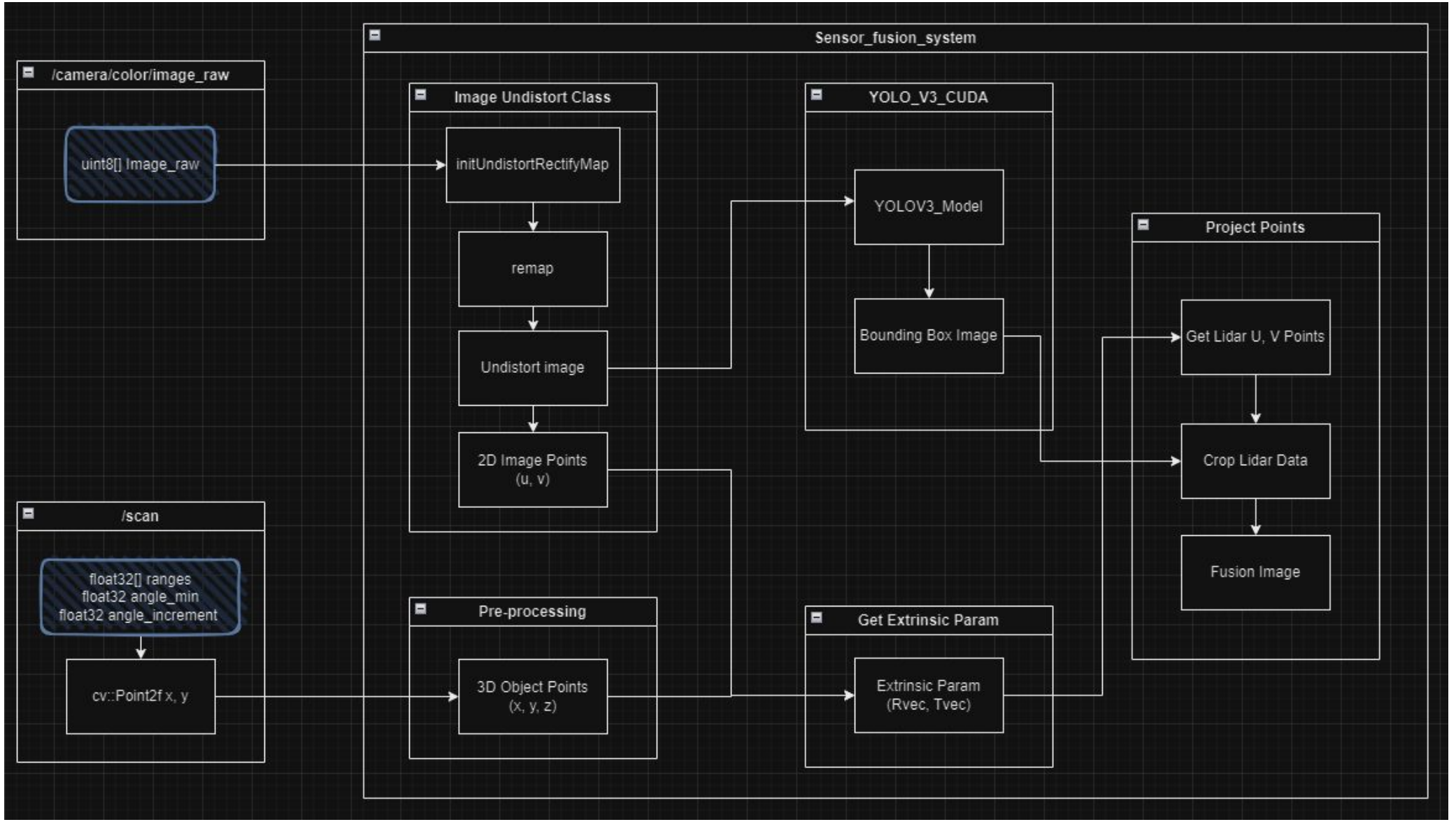
사용 기술 스택	용도
ROS	usb_cam & lidar scan message topic handling
C++	Sensor calibration & sensor fusion & YOLO object detection
Python & C#	Xycar Ground Control System
github	Sensor calibration & sensor fusion & YOLO object detection Version check

2

Pipeline

Part 2

Pipe line



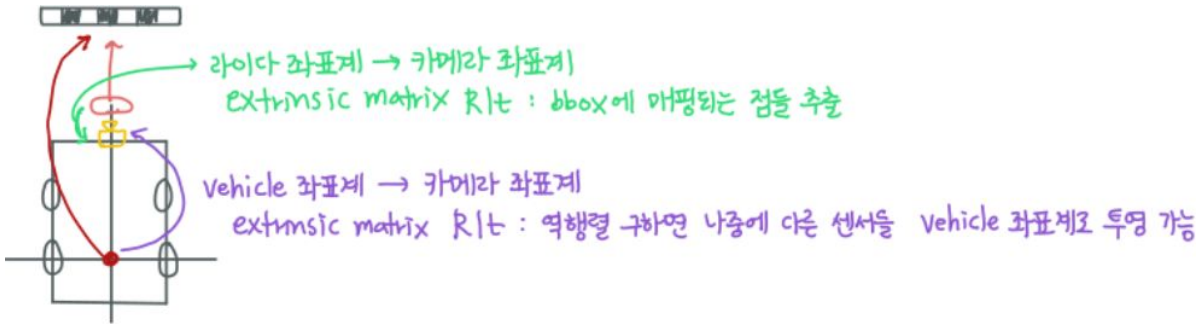
3

Sensor calibration

Idea: Get Vehicle coordinate location from lidar points

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

2D Image Coordinates | Intrinsic properties (Optical Centre, scaling) | Extrinsic properties (Camera Rotation and translation) | 3D World Coordinates



다음과 같은 방식으로 라이다 좌표계의 포인트로부터 Vehicle 좌표계의 포인트를 추출한다.

1) 라이다 좌표계 ⇒ 카메라 좌표계 간 Extrinsic matrix, E_l
2) Vehicle 좌표계 ⇒ 카메라 좌표계 간 Extrinsic matrix, E_v
3) 임의로 들어온 라이다 포인트를 P_l이라 할 때, P_l을 Vehicle 좌표계 P_v로 변경하기 위해서는,

$$P_v = Inverse(E_v) * E_l * P_l$$

을 적용하면 된다.

Process of Sensor Calibration

Camera Intrinsic Calibration

▼

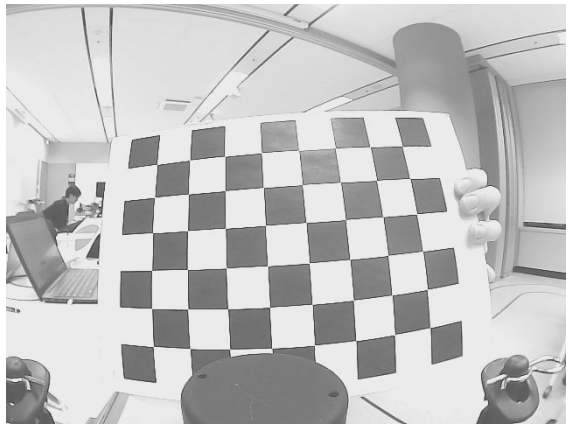
Camera - Lidar Extrinsic Calibration

▼

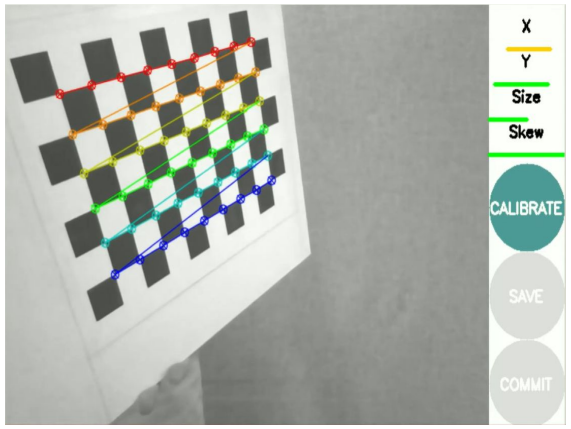
Camera - VCS Extrinsic Calibration

Part 3

Camera Intrinsic Calibration - ROS Camera Calibration



raw image



ROS Camera Calibration

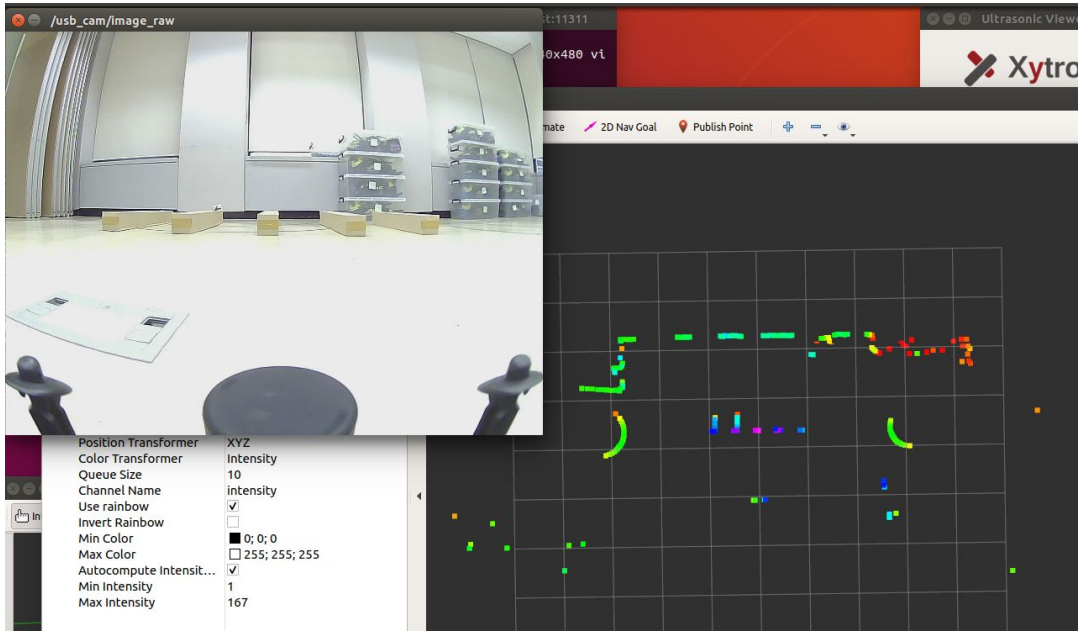
camera matrix
354.484625 0.000000 297.107003
0.000000 355.810675 238.133763
0.000000 0.000000 1.000000

distortion
-0.324182 0.092529 0.001340 0.000334 0.000000

Get Parameter

Part 3

Camera - Lidar Extrinsic Calibration



left: camera raw image / right: lidar range data

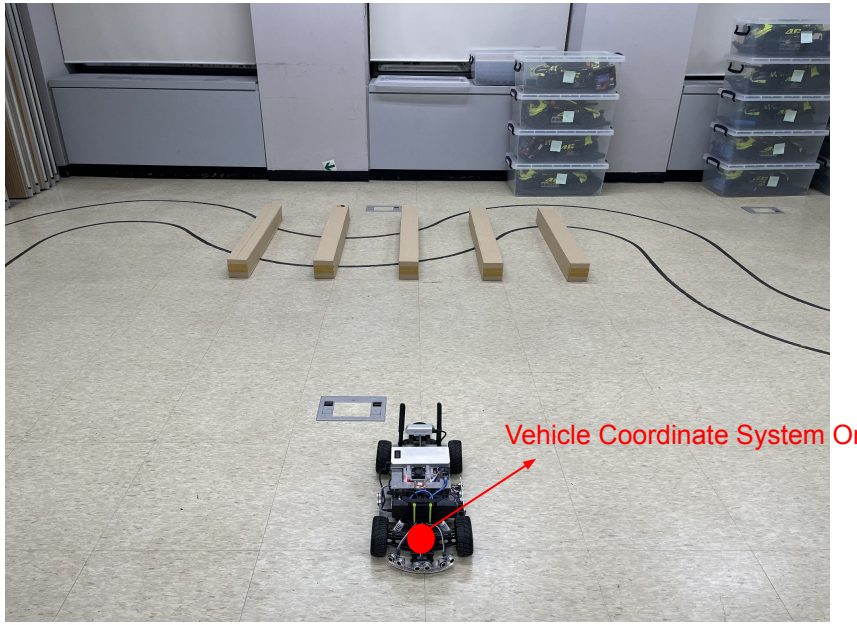
```
========================================
2d image point
========================================

(84.857, 216.255)
(108.035, 215.645)
(192.209, 216.864)
(217.828, 216.864)
(300.172, 218,694)
(324.57, 218.084)
(408.134, 218.084)
(431.923, 219.304)
(514.267, 219.304)
(536.836, 219.304)

============================

(84.857, 240.043)
(108.035, 240.653)
(192.209, 241.873)
(217.218, 242.483)
(299.562, 242.483)
(324.57, 242.483)
(408.744, 243.703)
(431.923, 244.313)
(513.047, 244.923)
(535.616, 243.093)
```

```
========================================
Lidar 3d coordinate
========================================

(-1.34259, -0.940092, 0.105)
(-1.34395, -0.840092, 0.105)
(-1.34139, -0.526456, 0.105)
(-1.34416, -0.426456, 0.105)
(-1.34638, -0.0840328, 0.105)
(-1.3479, 0.0159672, 0.105)
(-1.33974, 0.358982, 0.105)
(-1.34321, 0.451296, 0.105)
(-1.33782, 0.787527, 0.105)
(-1.33728, 0.887527, 0.105)

============================

(-1.34259, -0.940092, 0)
(-1.34395, -0.840092, 0)
(-1.34139, -0.526456, 0)
(-1.34416, -0.426456, 0)
(-1.34638, -0.0840328, 0)
(-1.3479, 0.0159672, 0)
(-1.33974, 0.358982, 0)
(-1.34321, 0.451296, 0)
(-1.33782, 0.787527, 0)
(-1.33728, 0.887527, 0)
```

get extrinsic matrix
from 2d image points & Lidar 3d object points

Part 3

Camera - VCS Extrinsic Calibration



Vehicle Coordinate System

```
========================================
2d image point
========================================

(84.857, 216.255)
(108.035, 215.645)
(192.209, 216.864)
(217.828, 216.864)
(300.172, 218,694)
(324.57, 218.084)
(408.134, 218.084)
(431.923, 219.304)
(514.267, 219.304)
(536.836, 219.304)

============================

(84.857, 240.043)
(108.035, 240.653)
(192.209, 241.873)
(217.218, 242.483)
(299.562, 242.483)
(324.57, 242.483)
(408.744, 243.703)
(431.923, 244.313)
(513.047, 244.923)
(535.616, 243.093)
```

```
========================================
Vehicle Coordinate System
========================================

(1.8, 1.0, 0.105)
(1.8, 0.9, 0.105)
(1.8, 0.55, 0.105)
(1.8, 0.45, 0.105)
(1.8, 0.1, 0.105)
(1.8, 0.0, 0.105)
(1.8, -0.35, 0.105)
(1.8, -0.45, 0.105)
(1.8, -0.8, 0.105)
(1.8, -0.9, 0.105)

============================

(1.8, 1.0, 0)
(1.8, 0.9, 0)
(1.8, 0.55, 0)
(1.8, 0.45, 0)
(1.8, 0.1, 0)
(1.8, 0.0, 0)
(1.8, -0.35, 0)
(1.8, -0.45, 0)
(1.8, -0.8, 0)
(1.8, -0.9, 0)
```

get extrinsic matrix
from 2d image points & VCS 3d object points

4

YOLO C++ & Velocity estimation

Camera - YOLO_V3 Python to C++

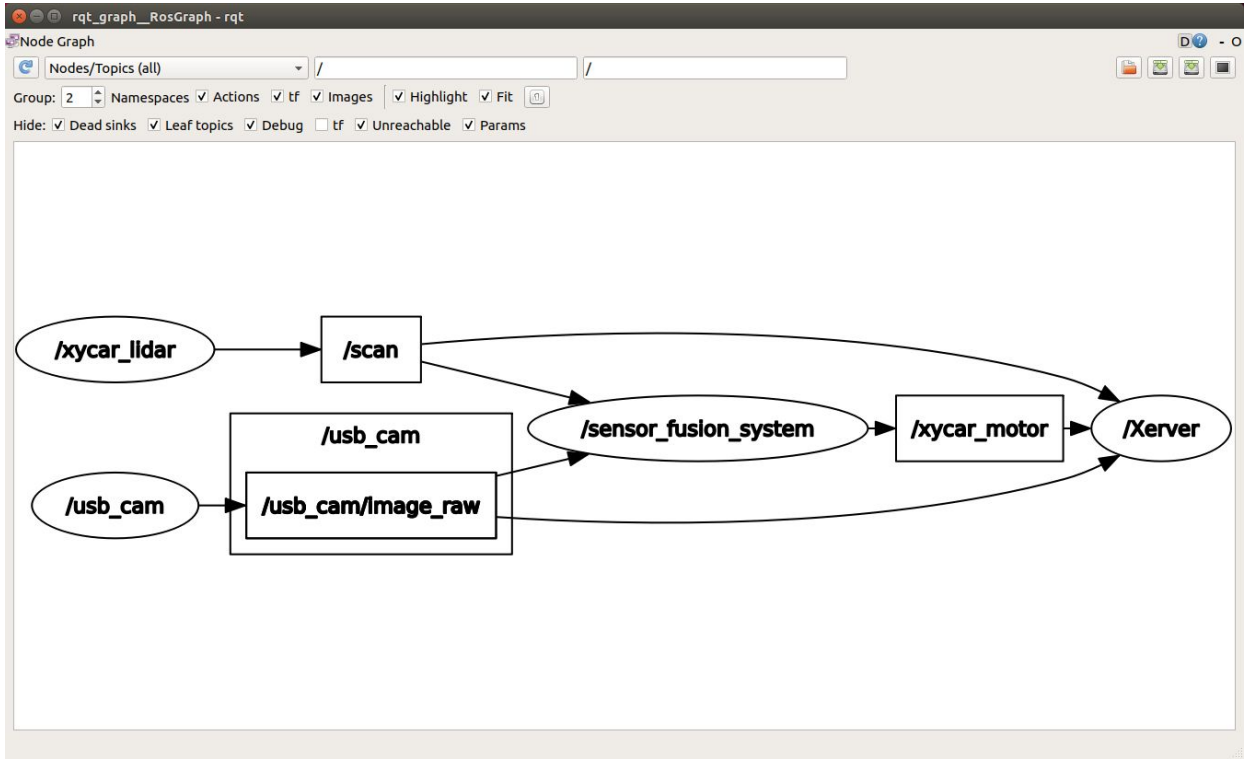
- 기존 yolov3_trt_ros의 경우 Python으로 작성이 되었으며, Bounding Box 정보를 C++ 노드로 전달.
- yolov3_trt_ros의 경우 실행시 12 ~ 16 FPS.
- C++로 yolov3를 구현하면 추가 노드 없이 FPS 향상을 기대.

Camera - YOLO_V3 Python to C++

```
#include "opencv/dnn.hpp"

int main(){
    mTemp = img.clone();
    mNeuralNet = cv::dnn::readNetFromDarknet(mYoloConfig, mYoloModel);

    // Neural Net setting
    if(mNeuralNet.empty()){
        std::cerr << "Network load failed!" << std::endl;
    }

    mNeuralNet.setPreferableTarget(cv::dnn::DNN_TARGET_CUDA);
    mNeuralNet.setPreferableBackend(cv::dnn::DNN_BACKEND_CUDA);

    // Convert Mat to batch of images
    cv::Mat blob = cv::dnn::blobFromImage(mTemp, 1 / 255.f, cv::Size(416, 416), cv::Scalar(), true);

    // Set the network input
    mNeuralNet.setInput(blob);
}
```

CPU가 적용된 YOLOV3

Camera - YOLO_V3 Python to C++



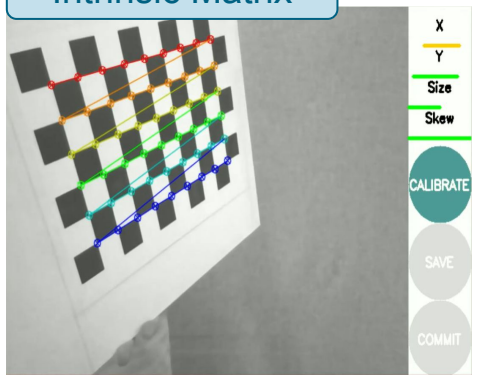
rqt_graph

CUDA가 적용된 YOLOV3

Part 4

Sensor fusion (lidar points ⇒ Image points)

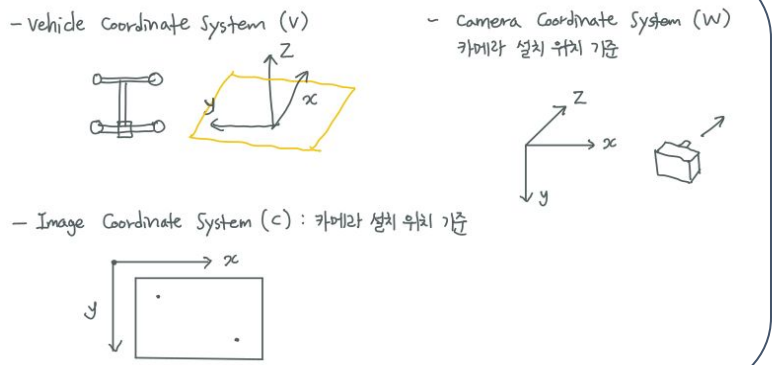
Intrinsic Matrix



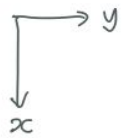
ROS Camera Calibration

Lidar New
3d Object Points

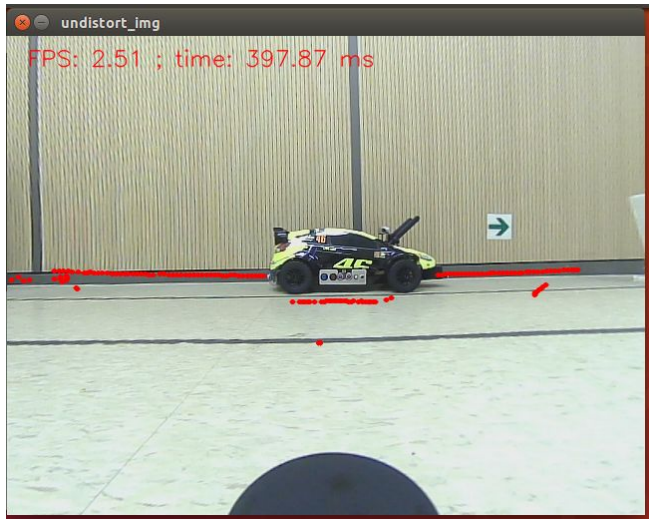
Extrinsic Matrix



VCS & Camera & Image
coordinate system



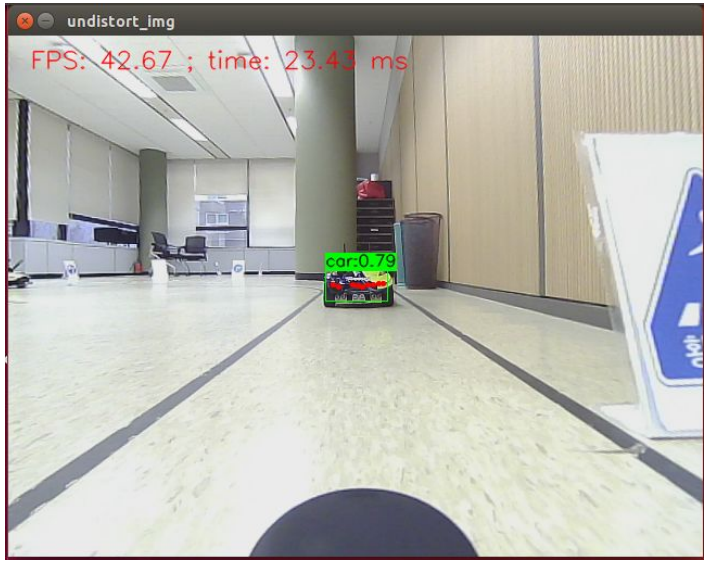
Lidar coordinate
system

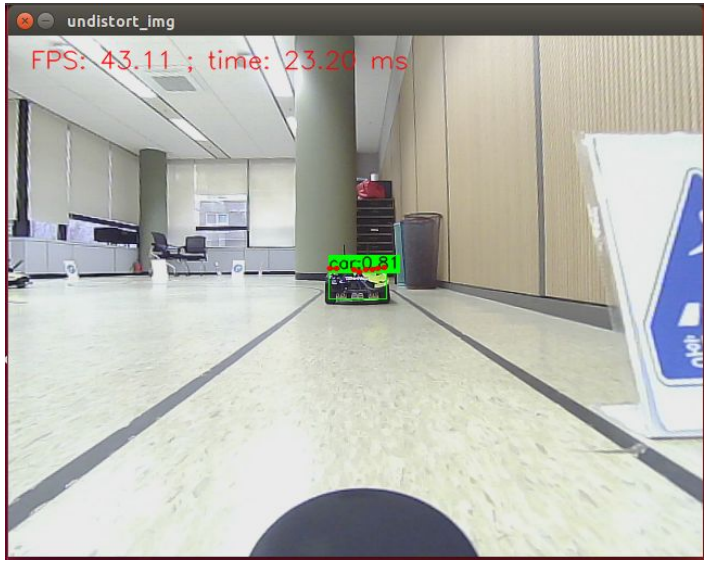


Projection to
2d image coordinate

Part 4

Velocity estimation





(x: 1.66033, y: -0.0707655, z: -0.0179314)

(x: 1.76814, y: 0.0323236, z: 0.0545476)

(x: 1.90645, y: 0.0327209, z: 0.111647)

distance (m)	0.14916
time (s)	0.33 (10 frame)
velocity (m/s)	**0.452016**

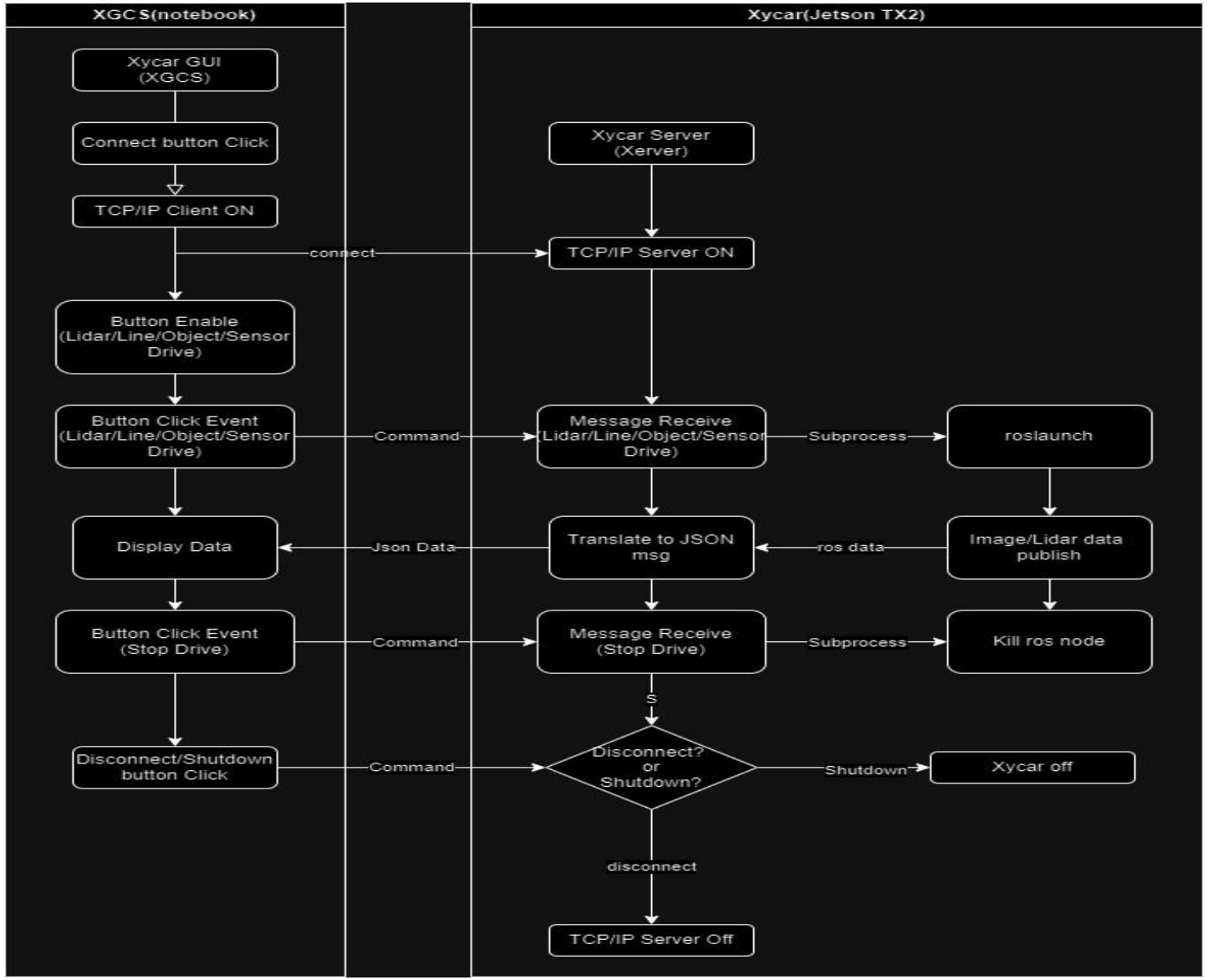
distance (m)	0.138311
time (s)	0.33 (10 frame)
velocity (m/s)	**0.419123**

5

Xycar Ground Control System

Part 5

Xycar Ground Control System(Block Diagram)



Part 5

Xycar Ground Control System(GUI 구성도)

View Tab :
USB Cam 영상 /
Lidar Cloud point 영상
전환 탭

View Box :
USB Cam 영상 /
Lidar Cloud point 영상
출력 화면

Xycar Status Box :
현재 Xycar의 Speed,
Angle, Mode 출력

Xycar TCP/IP Box :
Server(aka. Xerver) IP
입력 및 연결, 연결 해제
버튼 / 연결 상태 출력

Xycar ROS launch Btn :
Xycar 관련 ROS 명령어
실행 버튼 (Project 1~4
명령어)

Xycar Active kill btn:
ROS 명령어 종료 버튼

Xycar kill btn:
Xycar(Jetson TX2) 종료
버튼

Xycar Ground Control System(Xerver Code)

```
    #print(type(jsonString))   # class str


if (USE_ROS):
    def launch_roslaunch(self, package, launch_file):
        command = ["roslaunch", package, launch_file]
        process = subprocess.Popen(command, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
        self.start_subscriber()
        return process

    def execute_command(self,command):
        try:
            subprocess.run(command, shell=True, check=True, stderr=subprocess.PIPE)
        except subprocess.CalledProcessError as e:
            print(f"Error executing command: {e}")

    def start_subscriber(self):
        # Create a new subscriber instance
        self.image_subscriber = rospy.Subscriber("/usb_cam/image_raw", Image, self.image_callback, queue_size=1)
        self.lidar_subscriber = rospy.Subscriber("/scan", Laserscan, self.lidar_callback, queue_size=1)
        self.motor_subscriber = rospy.Subscriber("xycar_motor", xycar_motor, self.motor_callback, queue_size=1)
        self.stop_subscriber_flag = False

    def stop_subscriber(self):
        if self.image_subscriber is not None:
            # Unregister the existing subscriber
            self.image_subscriber.unregister()
            self.image_subscriber = None

        if self.lidar_subscriber is not None:
            # Unregister the existing subscriber
            self.lidar_subscriber.unregister()
```

Part 5

Xycar Ground Control System(시연 영상)

Part 5

Xycar Ground Control System(추후 개선 사항)

GUI(현재)

- ROS Launch 명령어가 추가 될 때마다 버튼 추가
- Image 를 picturebox에 출력 시 bitmap으로 변환하여 사용 → 딜레이 및 프레임 끊김 발생
- Xycar Speed, Angle Data 수집 미구현
- Lidar Data 활용 Cloud point View 미구현

GUI(개선)

- ROS Launch 명령어 텍스트 방식 입력으로 간소화
- RTSP 방식으로 변경하여 ffmpeg 그대로 출력 → 딜레이 및 프레임 끊김 완화
- Xycar Speed, Angle Data 수집 후 Display 구현
- Lidar Data 활용 Cloud point View 구현

Xerver(현재)

- ROS bag을 이용한 시뮬레이션에서만 Image Data 전송 성공 → 실제 환경에선 실패
- Jetson TX2 가 Server , GUI(PC)가 Client인 방식

Xerver(개선)

- 실제 환경에서도 Image 데이터를 ROS topic으로 받아 TCP/IP로 전송 → 이후 RTSP 방식으로 변경
- GUI(PC)가 Server, Jetson TX2 가 Client인 방식으로 변경 -> 1개의 PC에 여러 Jetson TX2(Client)를 연결하여 다중 제어 가능하도록 변경 → 다중 SLAM까지 계획 가능

6

Conclusion & Future work

Conclusion & Future work

Conclusion

- Lidar, Camera Sensor Fusion으로 특정 물체의 거리 데이터를 확보.
- Sensor Calibration시 다양한 거리의 점 데이터를 확보하면 Calibration을 진행하면 오류가 줄어들 것으로 기대.
- Python의 YOLOV3보다 C++에서 실행한 YOLOV3의 성능 향상.
- UI를 사용하여 Window에서 편하게 roslaunch 실행가능.

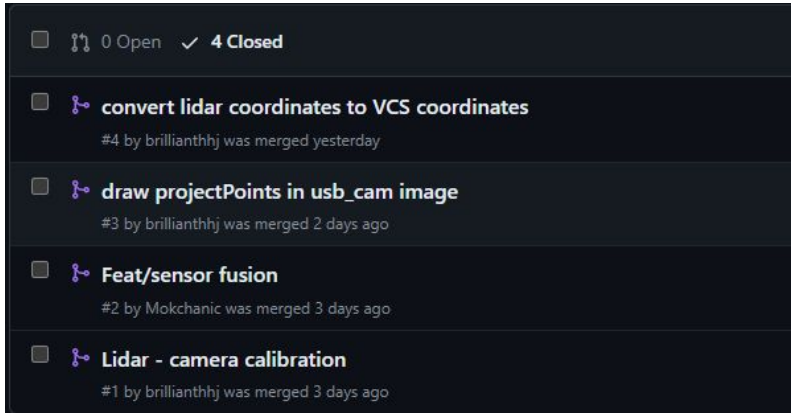
Future work

- yolo_trt_model을 적용하여 성능향상.
- Lidar sensor의 실측 범위를 늘려서 측정 및 Calibration을 진행.
- 물체에서 얻은 데이터를 바탕으로 동적 장애물의 속도 추정 자동화 진행.
- Xycar ground control system의 문제점 개선.

Part 6

Github collaboration

Pull requests



Github pull requests

Branch

Github branch

감사합니다.