EE405A
Perception

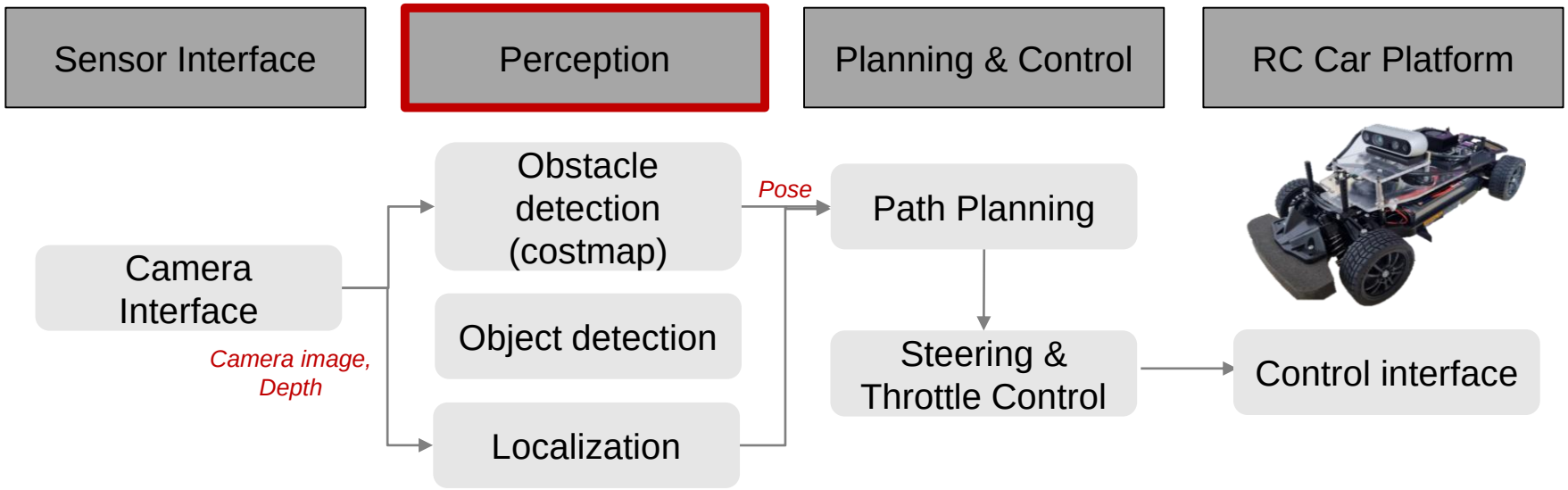
(TA) Gyuree Kang
School of Electrical Engineering
KAIST

November. 17th, 2023

Perception Module for Autonomous driving

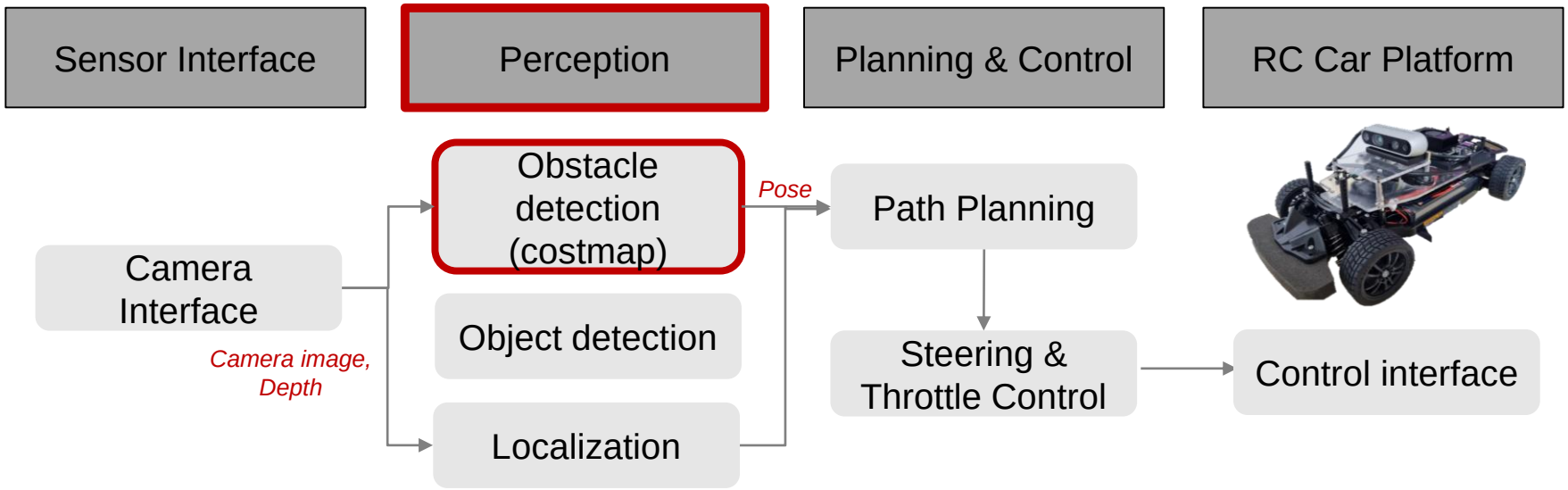
Perception Module

- Perception module
  - Perception module for autonomous driving
    - Obstacle detection
    - Drivable region detection
    - (Car) Lane detection
  - Perception module for task execution
    - Target object detection
    - Object pose estimation



Perception Module

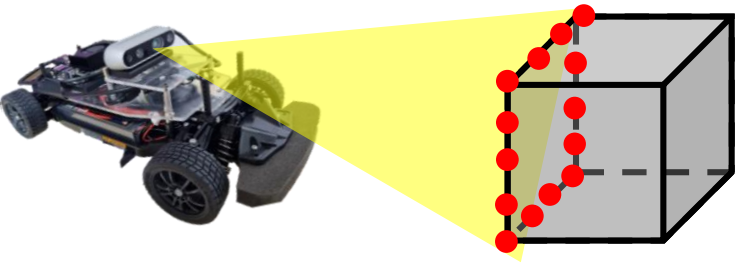
- Perception module
  - Perception module for autonomous driving
    - Obstacle detection
    - Drivable region detection
    - (Car) Lane detection
  - Perception module for task execution
    - Target object detection
    - Object pose estimation



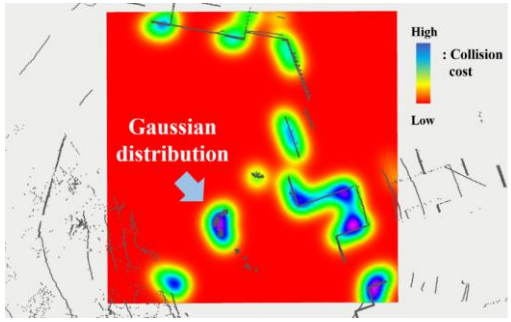
Obstacle detection

- Obstacle
  - **Static obstacle**: stationary and does not change its position over time relative to the observer
  - Dynamic obstacle: in motion, changing its position and velocity over time (e.g. human, cars, moving robots)
    - Need estimation of the future motion

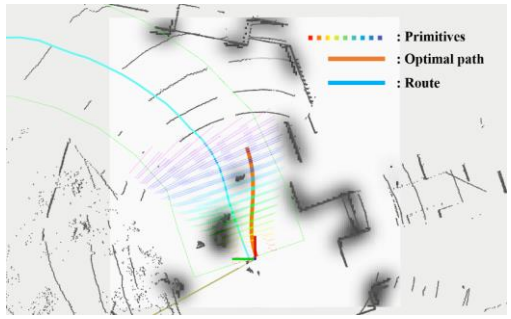
Camera depth image → Obstacle detection → Costmap generation → Path planning (local)



Static obstacle



Costmap

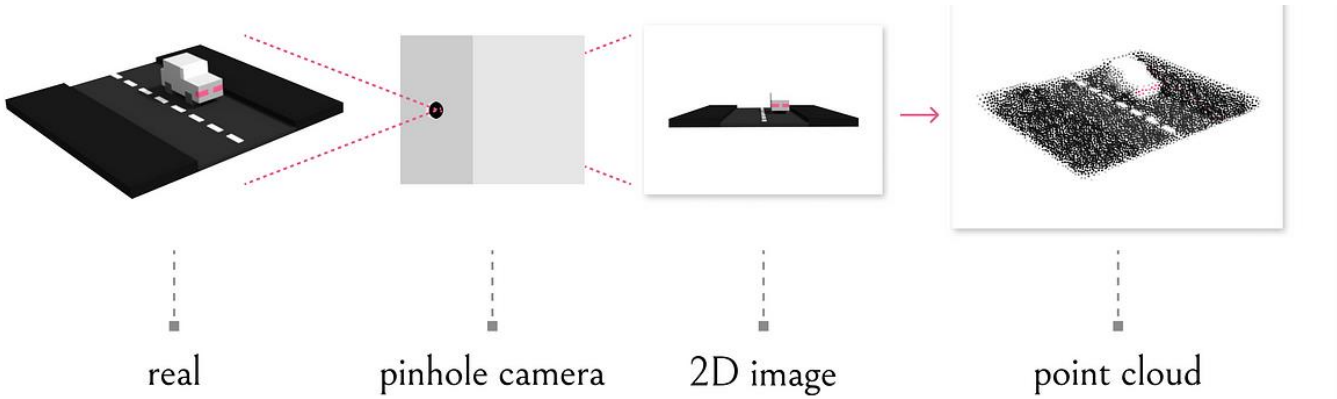


Local path

Depth Image to Point cloud

- Depth image to point cloud conversion
  - Image projection to 3D space
    - f_x, f_y: focal length
    - c_x, c_y: principal point
    - In depth camera_info
  - Advantages
    - Rich spatial information using (x, y, z) coordinates
    - Better integration with 3D algorithms
    - Flexibility in Data Processing (PCL library)

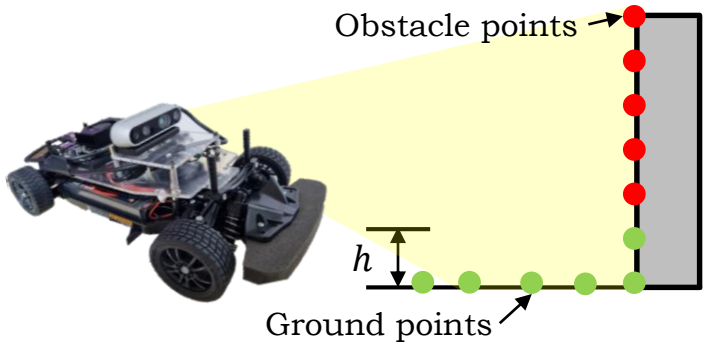
$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = z \begin{bmatrix} 1/f_x & 0 & 0 \\ 0 & 1/f_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u - c_x \\ v - c_y \\ 1 \end{bmatrix}$$



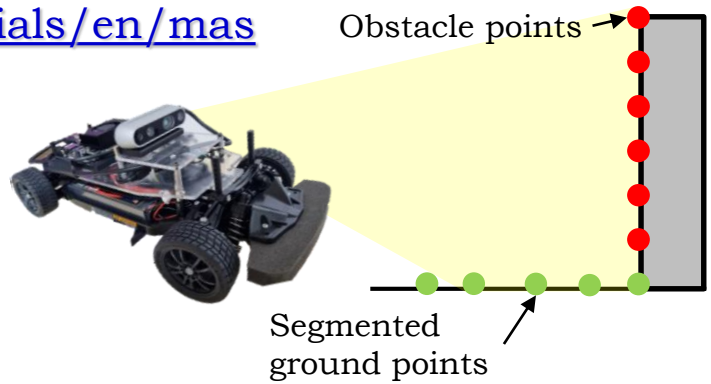
https://medium.com/yodayoda/from-depth-map-to-point-cloud-7473721d3f

Ground Filtering

- Height map
  - Distinguish ground points and object points using the height
    - Low computational cost, simple
    - Not effective in steep or irregular terrains

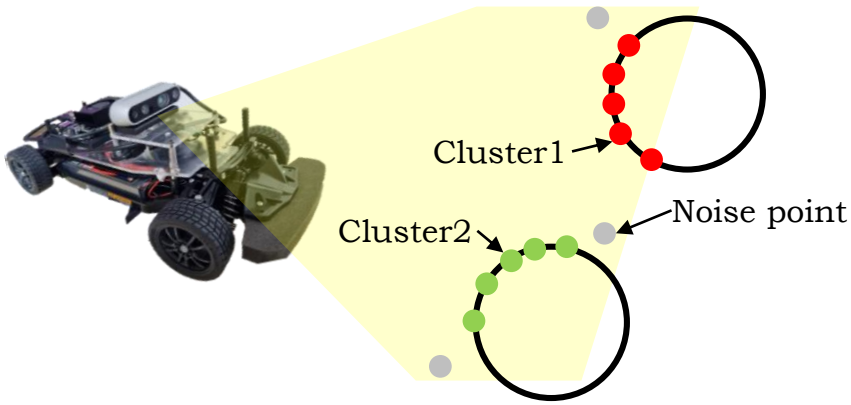


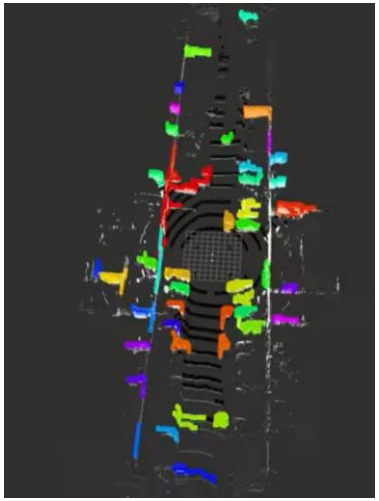
- Plane segmentation
  - RANSAC (RANdom SAmple consensus)
    - Applicable in diverse terrains
    - High computational cost, parameter tuning
    - Tutorial in C++ using PCL library https://pcl.readthedocs.io/projects/tutorials/en/master/planar_segmentation.html



Obstacle clustering

- Point cloud clustering
  - Improve robustness to sensor noise
  - Identify obstacles as group
    - Estimate position, shape, size of the object
    - Isolate distinct obstacles in the environment
  - Euclidean Cluster Extraction
    - Tutorial in C++ using PCL library https://pcl.readthedocs.io/projects/tutorials/en/master/cluster_extraction.html

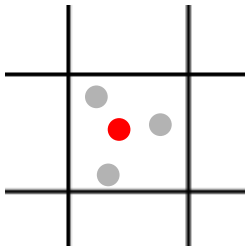




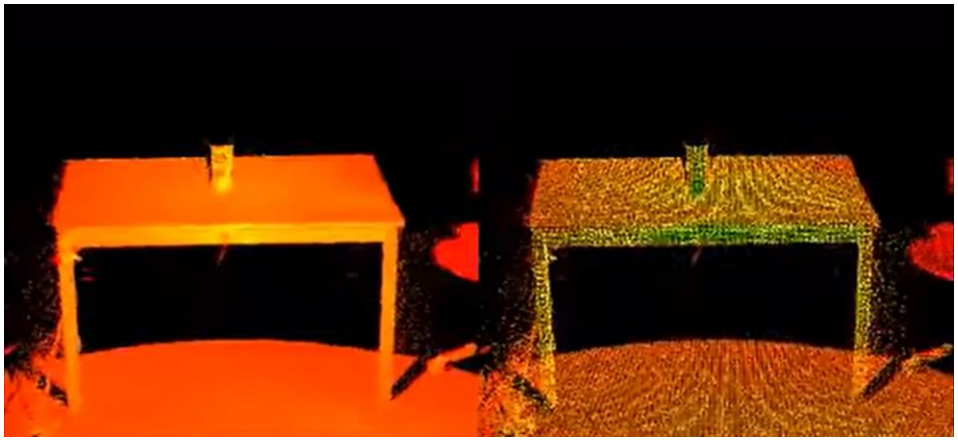
Result

Point Cloud Preprocessing

- Downsampling
  - Reducing the number of points in a point cloud dataset while retaining essential information
    - Computational efficiency
    - Data Storage
  - 3D VoxelGrid filter
    - Reduce the density of a point cloud by dividing it into cubic voxels and retaining only a single representative point within each voxel
    - Tutorial in C++ using PCL library https://pcl.readthedocs.io/projects/tutorials/en/master/voxel_grid.html



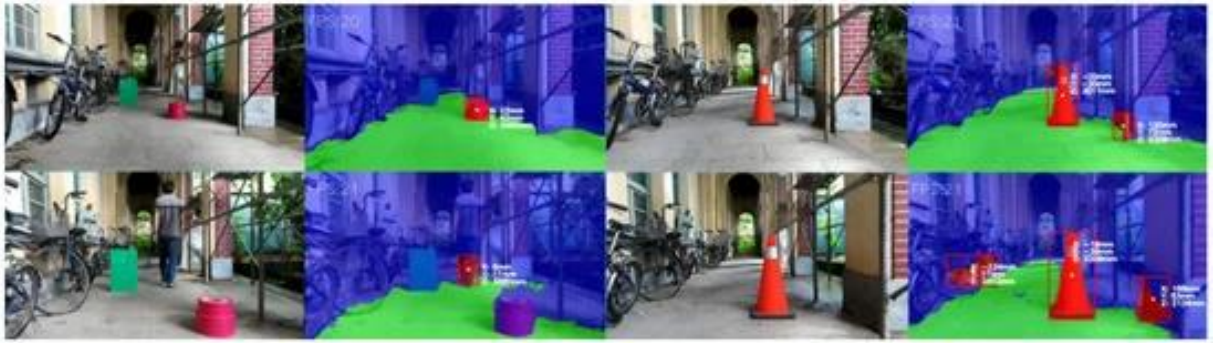
Voxelized grid



Result

Drivable region

- Drivable region detection
  - Drivable region: the area where a self-driving vehicle can safely operate.
  - Extracting drivable region instead of detecting obstacles.
  - Deep learning-based approaches



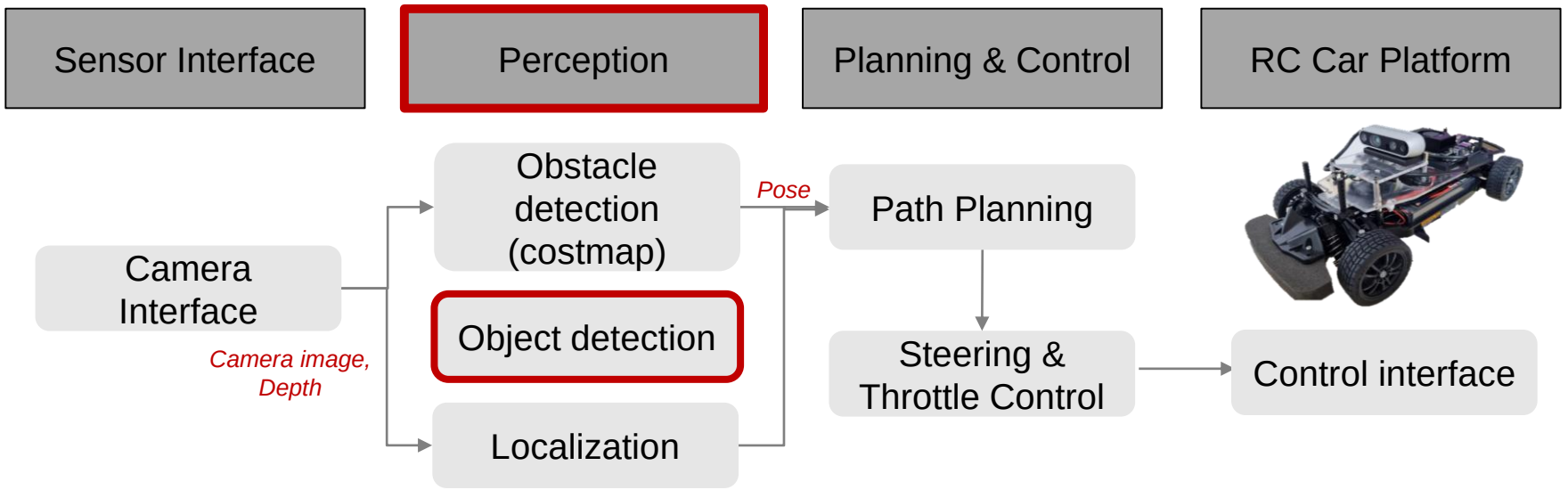
Nguyen, et al. "Effective free-driving region detection for mobile robots by uncertainty estimation using RGB-D data." 2022

Drivable region

Perception Module for Task Execution

Perception Module

- Perception module
  - Perception module for autonomous driving
    - Obstacle detection
    - Drivable region detection
    - (Car) Lane detection
  - Perception module for task execution
    - Target object detection
    - Object pose estimation



Target object detection

- Traditional computer vision approaches
  - Feature extraction techniques
    - Utilize methods like Histogram of Oriented Gradients (HOG) for detecting object features.
  - Cascade classifiers
    - Employ cascades of simple classifiers for efficient and rapid object detection.

- Deep learning approaches
  - Convolutional Neural Networks (CNNs), faster R-CNN
  - **YOLO** (You Only Look Once) - Processes the entire image in one forward pass, suitable for real-time applications.

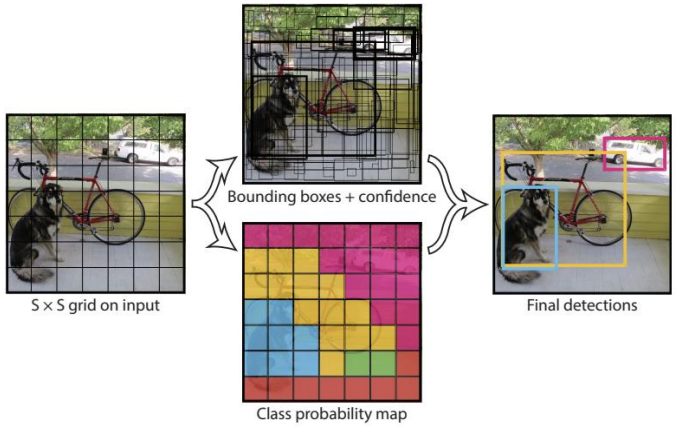
Deep Learning Approaches

- Object detection algorithm
  - Fine-tuned the model trained with a large dataset to detect and classify certain objects.
    - Bounding box
      - Lower computation, easier to label (gain data)
    - Semantic segmentation
      - Fine-grained information, higher compution
  - Few shot learnings
    - Model is trained to recognize and generalize from a very small amount of labeled examples (shots)
      - Li, Bowen, et al. "Airdet: Few-shot detection without fine-tuning for autonomous exploration." 2022.

YOLO v4 ROS Installation

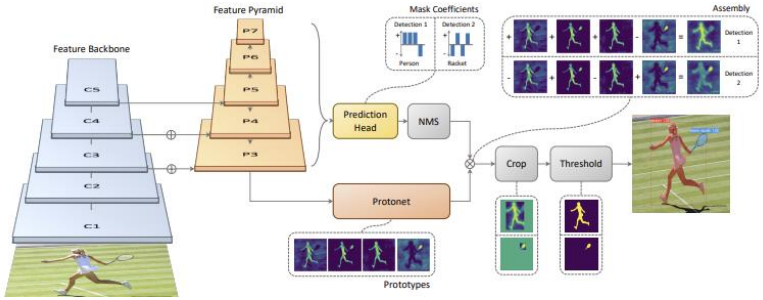
Detection algorithms

- Bounding box-based detection model
  - YOLO (You Only Look Once)
    Use anchor boxes for fast detection
    Fast detection speed
    Real-time object detection



YOLO

- Semantic segmentation model
  - YOLOACT (You Only Look At CoefficienTs) Fast Return mask and its class
    Fast detection speed
    Boundary of detection is more accurate than using bounding box-based model



YOLACT

YOLO V4

Darkent YOLO V4

- https://github.com/AlexeyAB/darknet
- Download and build source code (detailed instructions)

```
$ git clone https://github.com/AlexeyAB/darknet
$ cd darknet
$ mkdir build_release
$ cd build_release
$ cmake ..
$ cmake --build . --target install --parallel 8
```

If the model is trained by cpu instead of gpu, change the parameter in Makefile, then re-make (Recommend using GPU)

- Train custom dataset (detailed instructions ← **READ THIS**)
  1. Download pre-trained dataset
     - https://github.com/AlexeyAB/darknet#pre-trained-models
     - Recommend (yolov4.conv.137) with yolov4-custom.cfg
  2. Create .config file
     - Copy and past original .config file corresponding to pre-trained weight file
     - **!IMPORTANT!** Follow instructions to edit your .config file according to your custom dataset

YOLO V4

- Train custom dataset (detailed instructions) - continued
  2. Create .config file
     - **!IMPORTANT!** Follow instructions to edit your .config file according to your custom dataset

- change line `batch` to `batch=64`
- change line `subdivisions` to `subdivisions=16`
- change line `max_batches` to (`classes*2000` , but not less than number of training images and not less than `6000`), f.e. `max_batches=6000` if you train for 3 classes
- change line `steps` to 80% and 90% of max_batches, f.e. `steps=4800,5400`
- set network size `width=416` `height=416` or any value multiple of 32:

darknet/cfg/yolov3.cfg
Lines 8 to 9 in 0039fd2

```
8   width=416
9   height=416
```

- change line `classes=80` to your number of objects in each of 3 `[yolo]` -layers:
  - darknet/cfg/yolov3.cfg
    Line 610 in 0039fd2
    `610  classes=80`
  - darknet/cfg/yolov3.cfg
    Line 696 in 0039fd2
    `696  classes=80`
  - darknet/cfg/yolov3.cfg
    Line 783 in 0039fd2
    `783  classes=80`

- change [`filters=255`] to filters=(classes + 5)x3 in the 3 `[convolutional]` before each `[yolo]` layer, keep in mind that it only has to be the last `[convolutional]` before each of the `[yolo]` layers.
  - darknet/cfg/yolov3.cfg
    Line 603 in 0039fd2
    `603  filters=255`
  - darknet/cfg/yolov3.cfg
    Line 689 in 0039fd2
    `689  filters=255`
  - darknet/cfg/yolov3.cfg
    Line 776 in 0039fd2
    `776  filters=255`
- when using `[Gaussian_yolo]` layers, change [`filters=57`] filters=(classes + 9)x3 in the 3 `[convolutional]` before each `[Gaussian_yolo]` layer
  - darknet/cfg/Gaussian_yolov3_BDD.cfg
    Line 604 in 6e5bdf1
    `604  filters=57`
  - darknet/cfg/Gaussian_yolov3_BDD.cfg
    Line 696 in 6e5bdf1
    `696  filters=57`
  - darknet/cfg/Gaussian_yolov3_BDD.cfg
    Line 789 in 6e5bdf1
    `789  filters=57`

So if `classes=1` then should be `filters=18` . If `classes=2` then write `filters=21` . **(Do not write in the cfg-file: filters=(classes + 5)x3)**

YOLO V4

- Train custom dataset (detailed instructions) - continued
  3. Write obj.names file in directory build\darknet\x64\data\
  4. Put image-files (.jpg) of your objects in the directory build\darknet\x64\data\obj\
  5. Create train.txt and test.txt file in build\darknet\x64\data\
     train.txt and test.txt contain list of location of labeled data
     the recommended ratios of training data and evaluation data is 8:2
  6. Put pre-trained weight file in build\darknet\x64
  7. Write obj.data file in build\darknet\x64\data\
     (backup folder where the trained weights are saved)

```
classes = 2
train  = data/train.txt
valid  = data/test.txt
names = data/obj.names
backup = backup/
```

  8. Start training
     $ cd darknet
     $ ./darknet detector train [path to obj.data] [path to .cfg] [path to pre-trained weight file]
     (EX. $ ./darknet detector train data/obj.data yolo-obj.cfg yolov4.conv.137)

YOLO V4

- Label custom dataset (more the better)
    1. labelImg (video tutorial)
    2. Yolo_mark (video tutorial)

- Tip: Pseudo-labelling: auto labeling using trained weight
  - If you trained sufficient amount of custom weight, you can use it to label new images

    $ ./darknet detector test [path to obj.data] [path to .cfg] [path to trained weight file] -thresh 0.25 -dont_show -save_labels < [path to where you want to save]
  - It will generate label file, check if it is OK and edit it and add newly labeled data to the training/evaluation dataset

YOLO for ROS

YOLO ROS

- V4: https://github.com/Tossy0423/yolov4-for-darknet_ros.git

- Download source code

  $ cd src
  $ git clone --recursive https://github.com/Tossy0423/yolov4-for-darknet_ros.git

- Place your custom weight file to darknet_ros/darknet_ros/yolo_network_config/weights/

- Edit files

  1. Copy and past .cfg file you used while training custom dataset in darknet_ros/darknet_ros/yolo_network_config/cfg/
  2. Edit .yaml file in darknet_ros/darknet_ros/config/
  3. Edit launch file in darknet_ros/darknet_ros/launch/

- Launch

  $ roslaunch darknet_ros [your launch file]

YOLACT ROS

YOLACT ROS

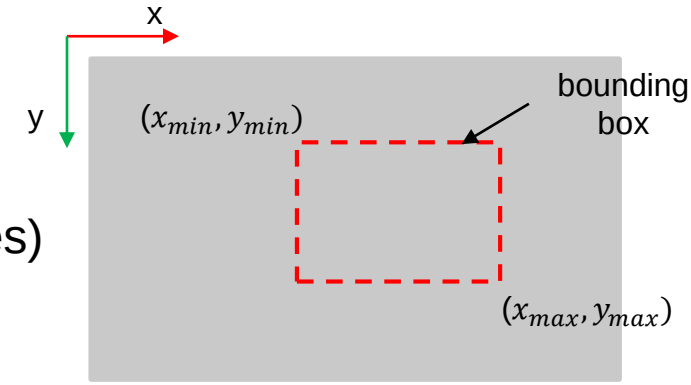
- https://github.com/Eruvae/yolact_ros.git
- Recommended labeling tool
  labelme (video tutorial)

```
$ pip3 install labelme
$ labelme
```

YOLO ROS Result

YOLO ROS

- Detection result (darknet_ros_msgs/BoundingBoxes)



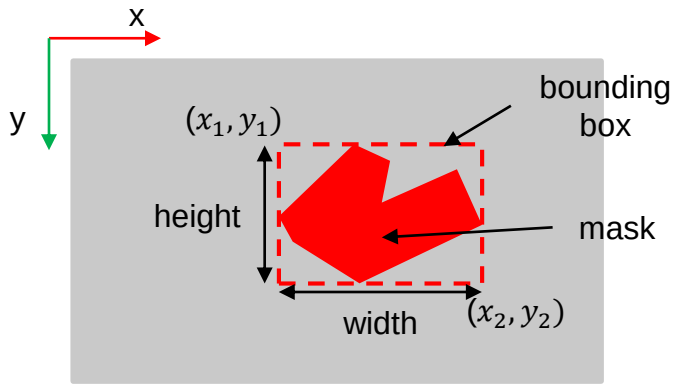

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
std_msgs/Header image_header
  uint32 seq
  time stamp
  string frame_id
darknet_ros_msgs/BoundingBox[] bounding_boxes
  float64 probability
  int64 xmin
  int64 ymin
  int64 xmax
  int64 ymax
  int16 id
  string Class
```

darknet_ros_msgs/BoundingBox
probability: detection reliability score
id: class id in integer
Class: class of the detected object

YOLACT ROS Result

YOLACT ROS

- Detection result (yolact_ros_msgs/Detections)

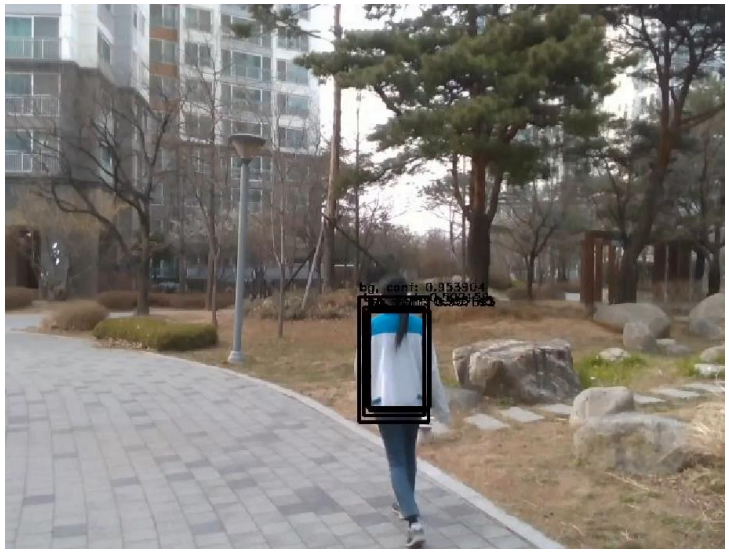



```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
yolact_ros_msgs/Detection[] detections
  string class_name
  float64 score
  yolact_ros_msgs/Box box
    int32 x1
    int32 y1
    int32 x2
    int32 y2
  yolact_ros_msgs/Mask mask
    int32 width
    int32 height
    uint8[] mask
```

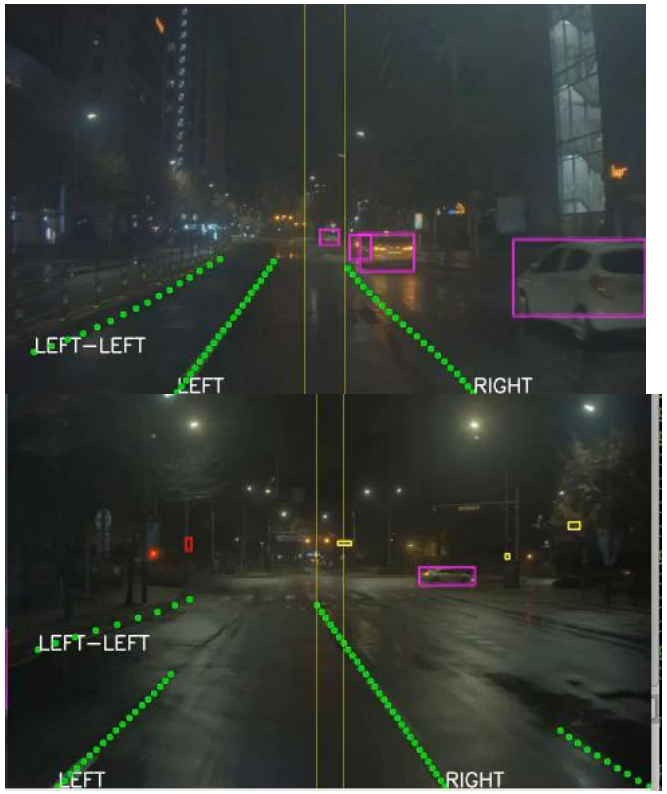
yolact_ros_msgs/Detection
class name: class of the detected object
score: detection reliability score
box: bounding box location in x, y

yolact_ros_msgs/Mask
width: width of the bounding box
height: height of the bounding box
Mask: segmented result of the image

Object Detection Result Visualization



YOLO object detection



Bounding box-based object detection
with lane detection

Segmentation Result Visualization

YOLACT ROS (video)

Target object (cup) pose estimation
& masked point cloud
(RVIZ)

Real-time image segmentation
(rqt_image_view)

Programming Assignment

- Perception for autonomous driving **Due: 11.29**
  1. Point cloud processing for obstacle point cloud extraction **(50%)**
     - Utilize at least one processing algorithm (such as ground filtering, downsampling, clustering, etc.) to detect obstacles in the point cloud.
     - Visualize the results in RVIZ using the sensor_msgs/PointCloud2 message along with the original point cloud input.
     - Submit a YouTube link to the video. (20%)
     - Submit a captured image of the code (only the processing part) (20%)
     - Write a pros and cons analysis of the chosen approach, considering the specific application's environment and robot configuration. (10%)
     - You cannot share it with your teammates
  2. Object detection **(50%)**
     - Train an object detection model (YOLOv4 or other) for the given target
       - Target object will be given to you on November 21st in room N5 2354.
     - Submit a YouTube link featuring a video demonstrating the detection of the target object using the custom-trained weights. The input for the detection should be RealSense camera images. (50%)
     - You can share the result with your teammates
- Submit a PDF/Word file named EE405A_perception_[Student ID]_[Full name].

Programming Assignment Tips

- You can modify the code from depthImage2pointCloud
  link: https://github.com/Guri-cccc/depthImage2pointCloud.git
  - Use the projected point cloud from depth image as original point cloud input

- Point cloud types:
  - Point cloud ROS message: sensor_msgs/PointCloud2
  - PCL point cloud: pcl::PointCloud<pcl::PointXYZ>
  - Conversion:
    - ROS message → PCL point cloud
      - pcl::fromROSMsg(message, pcl_pointcloud);
    - PCL point cloud → ROS message
      - Pcl::toROSMsg(pcl_pointcloud, message);

Programming Assignment Tips

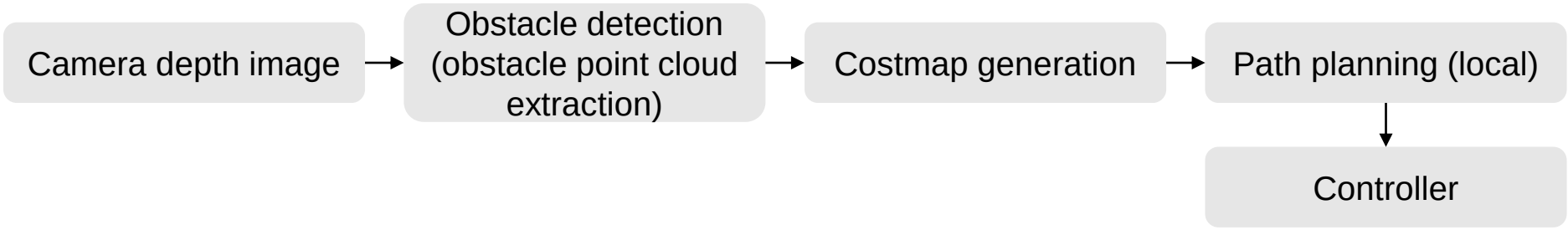
original point cloud input

Add your code here

```
void PointCloudProjection::DepthImageCallback(const sensor_msgs::ImageConstPtr& msg)
{
    cv::Mat depth_pic;
    cv_bridge::CvImagePtr depth_ptr;

    pcl::PointCloud<pcl::PointXYZ>::Ptr pointcloud (new pcl::PointCloud<pcl::PointXYZ>);

    try
    {
        depth_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::TYPE_32FC1);
    }
    catch(cv_bridge::Exception& e)
    {
        ROS_ERROR("Could not convert from '%s' to 'mono16'.", msg->encoding.c_str());
    }
    depth_pic = depth_ptr->image;
    *pointcloud = GetPointCloud(depth_pic);

    // viz
    sensor_msgs::PointCloud2 detected_pointcloud_m;
    pcl::toROSMsg(*pointcloud, detected_pointcloud_m);
    detected_pointcloud_m.header.frame_id = msg->header.frame_id;
    pub_pointcloud.publish(detected_pointcloud_m);

    pointcloud = nullptr;
}
```

Weekly Progress Report

- Perception
  - Make your own perception module for obstacle point cloud extraction. (using ground filtering, downsampling, clustering, etc.)
  - Combine your perception module with planning and control module and run your car in the real-world



Q & A

fingb20@kaist.ac.kr