ROBOTICS CLUB:SDC

**MAKING A MAP USABLE BY SDC**

horizontal line

# 

**Data Acquisition:**

**LiDAR Sensor:**

− "Light Detection and ranging sensor" This sensor is used for distance measurement and depth measurement.

− It is also used to measure the elevation of things like buildings, trees and nowadays, we can also see that this technology is being rapidly adopted by the automobile industry.

− Overall, we can see that this technology is taking place of the ultrasonic sensors.

**How does a LiDAR sensor works?**

− The functionality of the Lidar sensor is similar to Radar and Sonar, but here LIDAR uses a light source for its measurements. It uses green or near infrared light because this light source reflects strongly off of vegetation.

− It continuously emits light energy, when any object cut that light, it reflects back to sensor's receiver then control unit will calculates the time taken by light to reflect back.

Distance=Travel time\*speed of light / 2

By doing this calculation we can calculate the distance between ground and LIDAR.

− The measured results will be OK if Lidar is mounted on ground level (in stable condition)

− Additionally for more accurate results, we will need the GPS module

**Gps**= for proper positioning.

**Distance - Altitude= Accurate Distance.**

In this way we will get accurate results.

**High Definition Camera:**

* While LiDAR excels at 3D, cameras provide valuable visual information for lane marking recognition and road surface condition assessment.
* Use a high-resolution camera with a wide field of view (FOV) capturing a significant portion of the road ahead, typically mounted behind the windshield or on the roof facing forward. A FOV of around 60° horizontally is recommended.
* Consider additional cameras facing sideways or slightly backwards to capture relevant information for specific use cases, like intersection detection or sign recognition.
* Most Importantly, Synchronization of LiDAR and HD Cameras plays an important role in Data acquisition which is critical for associating visual features with the 3d points in LiDAR point cloud data.
* In the modern sensors, there are built-in synchronization features, but if they don’t have it. We have to perform synchronization with trigger signals or time stamps in the post preprocessing phase of the project.

**Processing:**

**Removing the outliers and Noise of LiDAR data:**

1. **Statistical Outline Removal(SOR)**

* This is a common technique that identifies points deviating significantly from their neighbors.
* It calculates the distance between each point and its k-nearest neighbors (points closest in space).
* Points exceeding a user-defined threshold distance from their neighbors are considered outliers and removed.
* You can adjust the threshold based on the expected noise level and data density.

1. **Radial Outlier Removal(ROR)**

* This method focuses on the range (distance from the sensor) of the points.
* It analyzes the distribution of points radially outward from the sensor.
* Points falling outside a statistically defined range are flagged as outliers and removed.
* ROR is effective for removing isolated noise points or points caused by birds flying through the scan.

1. **Ground Filtering**

* A specific type of filtering aimed at removing points not representing the ground surface.
* Various algorithms exist, like the iterative closest point (ICP) or morphological filtering.
* These algorithms identify planar regions corresponding to the ground and remove points above a certain tolerance from the plane.
* Ground filtering is crucial for tasks like terrain modeling and extracting objects above the ground.

**Software Tools :**

* **CloudCompare** (<https://www.danielgm.net/cc/>): A free and open-source point cloud processing software with various filtering tools, including statistical outlier removal and ground filtering algorithms like progressive morphological filters (PMF).
* **PCL (Point Cloud Library)** (<https://pointclouds.org/>): An open-source library for point cloud processing that can be integrated into custom workflows. It provides modules for statistical outlier removal, radius outlier removal, and ground filtering using methods like random sample consensus (RANSAC).

**Segmentation:**

**Deep-Learning Based Segmentation**

This advanced approach utilizes deep neural networks trained on labeled LiDAR point cloud data.

The network learns to identify patterns and relationships between the spatial coordinates (x, y, z) and intensity values (reflectance) of points, classifying them into different categories.

Popular methods include:

**PointNet++:** This network architecture processes point clouds directly, treating each point as a unique feature vector. It can effectively learn complex relationships between points for segmentation.

**SqueezeSegV2:** This network leverages convolutional neural networks (CNNs) typically used for image data. It projects the LiDAR points onto a spherical image and then uses CNNs for segmentation, achieving high accuracy.

**Software Tools:**

Many software tools mentioned previously for pre-processing also offer functionalities for LiDAR point cloud segmentation using both traditional filtering and deep learning approaches. Examples include CloudCompare (open-source)

**Feature Extraction:**

**Machine Learning Techniques:** These methods leverage algorithms that learn feature representations from the data itself.

**Support Vector Machines (SVM):** Can be trained to classify points based on various features, aiding in segmentation and feature extraction.

**PointCNN (Deep Learning):** A deep learning architecture designed for point cloud data. It can automatically learn complex feature representations suitable for various tasks.

**Integration with Mapping Software:**

* The output of the segmentation process, typically a point cloud with each point labeled according to its class (ground, lane marking, etc.), can be used as input for further processing.
* Tools might be available within the mapping software itself or in conjunction with LiDAR processing software to convert the segmented point cloud data into the specific format required by openDRIVE or Lanelet2. This format often involves lane centerline data, lane width information, and potentially additional data like curvature.

**Preprocessing of Camera Data:**

**Cleaning of the Data and making it fit for Data Extraction.**

**1. Radiometric Calibration:**

* This corrects for variations in sensor response across the image. It ensures consistent pixel values represent actual light intensity.

**2. Color Correction:**

* Cameras might capture color with a bias. This step adjusts the image to a standard color space for consistent feature extraction.

**3. Noise Reduction:**

* Sensor noise can introduce artifacts. Techniques like filtering are applied to remove noise while preserving details.

**4. Deblurring (Optional):**

* Motion blur due to camera shake can happen. Deblurring algorithms can be used to sharpen the image, especially for lane markings.

**Data Extraction**

**Edge Detection:**

* This is a fundamental step. Algorithms like Canny edge detection identify sharp transitions in pixel intensity, which often correspond to lane markings.

**Image Segmentation and Clustering:**

* This groups pixels with similar characteristics. It can be used to differentiate lane markings from the road surface, background objects, and shadows. Techniques include:
  + Thresholding: Separates pixels based on intensity values.
  + Region Growing: Groups connected pixels with similar properties.
  + Clustering: Groups pixels based on a distance metric in a feature space (e.g., color).
* We can also apply Hough Transform in order to identify lines representing the center of each lane marking. The peak in the Hough are dominant line parameters.

**For Road Curvature:**

* Utilize techniques like curve fitting to model the centerline of the lane as a parametric curve (e.g., polynomial).
* The curvature of the road at any point can then be calculated by analyzing the second derivative of the fitted curve.

Open-Source Libraries:

* *OpenCV (Open Source Computer Vision Library):*<https://opencv.org/> This versatile library offers a wide range of image processing and computer vision functionalities, including thresholding, region growing, and various clustering algorithms. It's a popular choice for prototyping and experimentation due to its ease of use and extensive documentation.
* *Scikit-image***:**<https://scikit-image.org/> This Python library provides a collection of algorithms for image processing, computer vision, and related tasks. It offers implementations of various segmentation techniques like thresholding, region growing, and clustering algorithms.

**Spatial Correspondence:**

* This step is crucial as it is used to establish a relationship between points in the LiDAR point cloud and the corresponding pixels in the camera image.
* *Techniques and their description:*

**1. Camera-LiDAR Calibration:**

* This is the fundamental step for establishing a relationship between the coordinate systems of the LiDAR and camera sensors mounted on the vehicle.
* It involves capturing data of a specific calibration target (checkerboard pattern or specific calibration rig) with both sensors simultaneously.
* Mathematical models are then used to estimate the extrinsic parameters, which describe the relative transformation (rotation and translation) between the LiDAR and camera coordinate systems.
* Intrinsic parameters of each sensor (lens distortion for camera) might also be considered during calibration.

**2. Point Projection:**

* Once calibration is done, you can project 3D points from the LiDAR data onto the camera image plane.
* This is achieved by applying the estimated extrinsic parameters and performing back-projection calculations.
* By knowing the corresponding pixel location in the camera image for a specific 3D point in the LiDAR data, you establish the spatial link between the two data sources.

**3. Techniques for Point Projection:**

* There are different approaches for point projection, depending on the calibration method and sensor setup:
  + **Linear Camera Model:** This is a simplified approach assuming a perfect pinhole camera model. It might be sufficient for initial estimations but can have limitations for wide-field-of-view cameras.
  + **Non-Linear Camera Models:** More complex models consider lens distortion effects for more accurate projection, especially for wide-angle cameras.

**Joint Clustering:**

* Apply clustering algorithms like K-means or DBSCAN to group together data points (from both LiDAR and camera data) that share similar characteristics.
* For instance, points with similar heights in the LiDAR data (potentially lane markings) and corresponding pixels in the camera image with consistent lane marking color or texture would be clustered together.

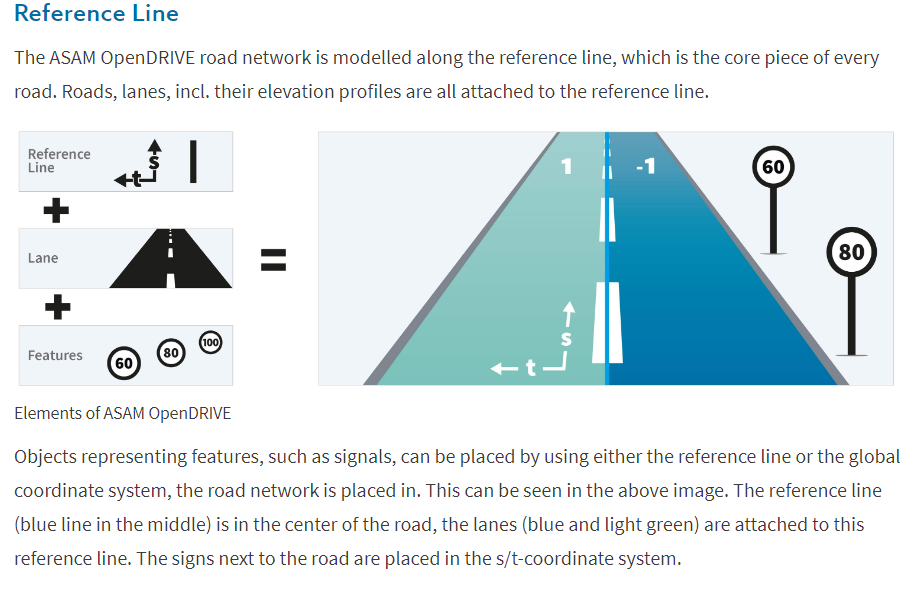
**Mapping Software:**

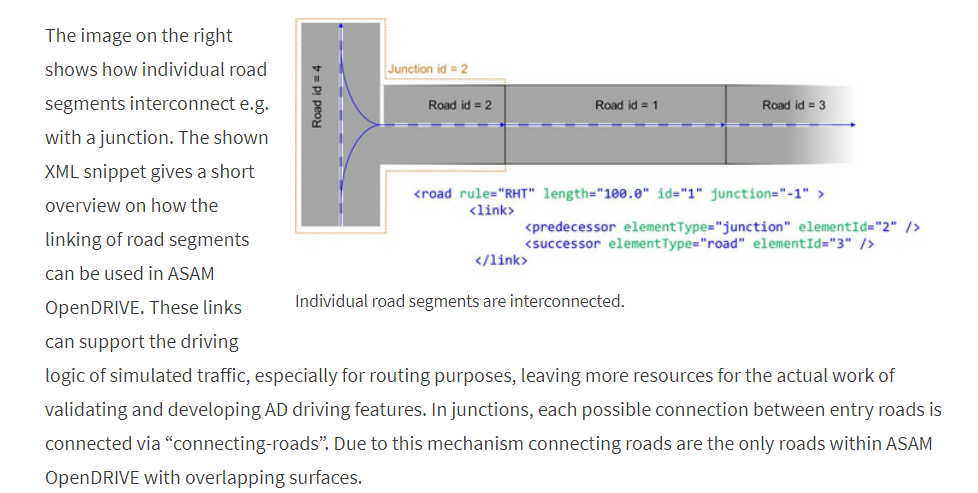
Tools like OpenDRIVE or Lanelet are specifically designed for creating high-definition (HD) maps for autonomous vehicles. These tools allow for:

* **Defining Road Network:** Specifying lanes, lane markings, speed limits, and traffic signs.
* **3D Environment Representation:** Integrating LiDAR data for precise road geometry and object placement.

**OpenDRIVE:**

The ASAM OpenDRIVE format provides a common base for describing road networks with extensible markup language (XML) syntax, using the file extension xodr. The data that is stored in an ASAM OpenDRIVE file describes the geometry of roads, lanes and objects, such as roadmarks on the road, as well as features along the roads, like signals. The road networks that are described in the ASAM OpenDRIVE file can either be synthetic or based on real data.





For Further Details:<https://www.asam.net/standards/detail/opendrive>

**Lanelet2:**

Lanelet2 is a C++ library for handling map data in the context of automated driving. It is designed to utilize high-definition map data in order to efficiently handle the challenges posed to a vehicle in complex traffic scenarios. Flexibility and extensibility are some of the core principles to handle the upcoming challenges of future maps.

For further details:

https://github.com/fzi-forschungszentrum-informatik/Lanelet2

**HOW AUTONOMOUS VEHICLES AVOID OBSTACLES IN THEIR PATH**

In autonomous vehicles, one of the most important features is the correct and accurate detection of obstacles as well as the track of the vehicle. The vehicle or car should be able to detect the presence of an obstacle precisely and well in time so that it can stop itself at a safe distance in order to avoid the collision.

There are various ways of implementing effective obstacle detection for vehicle safety systems.

1. **Sensor-based approach**
2. **Camera-based approach**
3. **Deep learning-based approach**
4. **Computer vision + PID Controller**
5. **Laser scanner approach**

Let’s discuss about each one of them

1. **Sensor-based approach**

The most common and widely used approach for both obstacle and track detection is the sensor-based approach. The commonly used sensors are Acoustic, Radar, Laser/LiDAR, Optical sensors, but most frequently used ones are Radar and LiDar sensors.

1. **LiDAR sensors:** LiDAR stands for Light imaging Detection and Ranging. It works by sending out laser light pulses and measures the time it takes the light to hit the object and bounce back to the sensor. The pulses are sent out in nanoseconds, creating a high speed way to scan areas and get thorough, highly accurate 3D images to create elevation maps, contours and more. Here is an example: <https://www.youtube.com/watch?v=IbAR8mawcLo>

**Advantages:**   
i) LiDAR technology is very accurate, consistent and can be used to detect small objects and create exact 3D models.

ii) Since Light travels at an enormous amount of speed, it is possible to scan huge areas in fairly short time.

iii) LiDAR is significantly less expensive than any other method of land surveying.

**Disadvantages:**

i) LiDAR technology is complex, it requires an experienced personnel to be used.

ii) Purchasing high-end LiDAR sensors can be costly.

**b) RADAR sensors**: RADAR stands for ‘**Radio Detection And Ranging’** and is an active transmission and reception method in the microwave GHz range. Radar sensors are used for contactless detection, tracking, and positioning of one or more objects by means of electromagnetic waves.

Example: <https://www.youtube.com/watch?v=ahhb5EjHleY>

**Working:** The radar antenna emits a signal in the form of radar waves, which move at the speed of light and are not perceivable by humans. When the waves hit objects, the signal changes and is reflected back to the sensor- similar to an echo. The signal arriving at the antenna contains information about the detected object.

**Advantages:**

i) RADAR is independent from weather conditions

ii) tolerates extreme heat or cold

iii) works in the dark

iv) is maintenance free.

**2. Camera-based approach**

The second most popular approach is the camera-based approach that is used for detecting the track and obstacles in autonomous vehicles. Some researchers consider it a sub-category of sensor-based approach

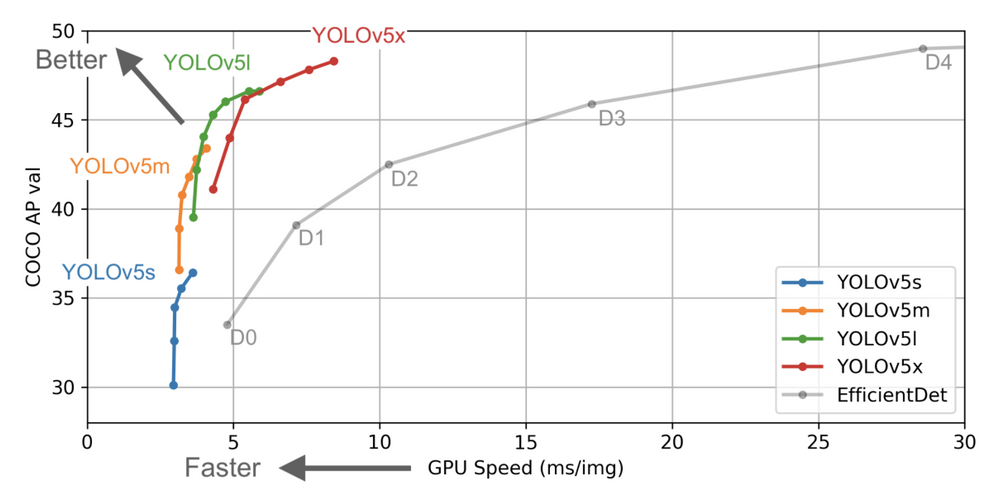
It is divided into 3 Categories

1. Knowledge based
2. Stereo vision based
3. Motion based

Detection of obstacles is done from a camera mounted on an autonomous UAV using two schemes, 1) Face detection and 2) Color Detection. a robust vehicle detection system is described that detects vehicles in the rear view of the host car. It records the motion parameters of the host vehicle to determine the driven path.

**3. Deep learning using YOLO v5:**

YOLOv5 is a model in the You Only Look Once (YOLO) family of computer vision models. YOLOv5 is commonly used for detecting objects. YOLOv5 comes in four main versions: small (s), medium (m), large (l), and extra large (x), each offering progressively higher accuracy rates. Each variant also takes a different amount of time to train.



In the chart above, you can see that all variants of YOLOv5 train faster than EfficientDet. The most accurate YOLOv5 model, YOLOv5x, can process images multiple times faster with a similar degree of accuracy than the EfficientDet D4 model. This data is discussed in more depth later in the post.

YOLOv5 derives most of its performance improvement from PyTorch training procedures, while the model architecture remains close to YOLO4

Object detection, a use case for which YOLOv5 is designed, involves creating features from input images. These features are then fed through a prediction system to draw boxes around objects and predict their classes.

The YOLO model was the first object detector to connect the procedure of predicting bounding boxes with class labels in an end to end differentiable network.

The YOLO network consists of three main pieces.

1. **Backbone**: A convolutional neural network that aggregates and forms image features at different granularities.
2. **Neck**:A series of layers to mix and combine image features to pass them forward to prediction.
3. **Head**:Consumes features from the neck and takes box and class prediction steps.

Detailed YOLO discussion: <https://blog.roboflow.com/yolov5-improvements-and-evaluation/>

**4. Computer Vision + PID Controller:**

### Computer Vision in SDCs

**Computer Vision** involves enabling machines to interpret and make decisions based on visual data from the environment. For SDCs, computer vision is crucial for understanding the surroundings, detecting obstacles, recognizing traffic signs, lane markings, and other vehicles, and making driving decisions.

Key components of computer vision for SDCs include:

1. **Cameras**: High-resolution cameras are mounted on the vehicle to capture continuous video feeds of the environment.
2. **Image Processing**: Algorithms process these images to detect and classify objects such as pedestrians, other vehicles, traffic signs, and obstacles.
3. **Deep Learning Models**: Convolutional Neural Networks (CNNs) and other deep learning models are trained to recognize and categorize objects accurately.
4. **Stereo Vision**: Using two cameras to obtain depth information, helping to gauge the distance to objects.
5. **Sensor Fusion**: Combining data from cameras with other sensors (like LiDAR and radar) for a more accurate understanding of the environment.

### Obstacle Detection and Avoidance

The process involves:

1. **Detection**: Identifying obstacles using image processing and deep learning algorithms.
2. **Localization**: Determining the position of obstacles relative to the vehicle using depth information from stereo vision or other sensors.
3. **Path Planning**: Calculating a safe path around obstacles using algorithms like Rapidly-exploring Random Tree (RRT), A\*, or Dijkstra’s algorithm.
4. **Control**: Adjusting the vehicle's trajectory in real-time to avoid obstacles while maintaining the desired route.

### PID Controller

A **PID Controller** is a control loop feedback mechanism widely used in industrial control systems. It continuously calculates an error value as the difference between a desired setpoint and a measured process variable and applies a correction based on proportional, integral, and derivative terms.

* **Proportional (P)**: The correction is proportional to the current error.
* **Integral (I)**: The correction is based on the accumulation of past errors.
* **Derivative (D)**: The correction is based on the prediction of future errors.

### Integrating Computer Vision and PID Controller

1. **Input from Computer Vision**: The computer vision system detects obstacles and provides real-time data on their position and movement.
2. **Error Calculation**: The desired path (setpoint) and the current trajectory (process variable) are compared to calculate the error.
3. **PID Correction**:
   * **Proportional Term**: Adjusts the steering angle in proportion to the current deviation from the desired path.
   * **Integral Term**: Adjusts based on the accumulated error over time to eliminate steady-state errors.
   * **Derivative Term**: Predicts future errors and adjusts to prevent overshooting or oscillations.
4. **Actuator Control**: The PID controller sends commands to the vehicle’s steering and throttle actuators to correct the trajectory.

### Practical Implementation Steps

1. **Set Up Cameras**: Install and calibrate cameras on the vehicle.
2. **Develop Vision Algorithms**: Use frameworks like OpenCV, TensorFlow, or PyTorch to develop and train object detection and recognition models.
3. **Integrate Sensors**: Combine data from cameras, LiDAR, radar, and other sensors.
4. **Path Planning Algorithms**: Implement path planning algorithms to generate safe paths around detected obstacles.
5. **PID Tuning**: Tune the PID controller parameters (Kp, Ki, Kd) to ensure responsive and smooth control.
6. **Testing and Validation**: Test the system in simulation environments and real-world scenarios, iteratively improving the algorithms and controller settings.

### Challenges and Considerations

* **Real-time Processing**: Ensuring that the vision and control systems can process data and respond in real-time.
* **Robustness**: Handling varying lighting conditions, weather, and unexpected obstacles.
* **Sensor Fusion**: Integrating data from multiple sensors for reliable obstacle detection and localization.
* **Safety**: Ensuring fail-safe mechanisms are in place for unexpected failures or extreme conditions.