CENTRALE
NANTES

MSc Control and Robotics
Academic year 2023 - 2024

AUVE Lab 1

Raffaele Pumpo
Emanuele Buzzurro
Under the supervision of Charlotte Beaune

Contents

1 Introduction **2**

2 Single actor **2**

2.1 Boxes in the actor frame . . . 2

2.2 Extraction of the corner . . . 3

3 Multiple actors **4**

3.1 Multiple point clouds in ego frame . . . 4

3.2 Boxes in different frames . . . 4

4 Bird Eye View **6**

4.1 Filtering points . . . 7

4.2 3D points in 2D pixels . . . 8

4.3 From 3D in 2D Boxes . . . 8

5 Segmentation of the obstacles **9**

6 Conclusion **11**

1 Introduction

Autonomous vehicles represent a groundbreaking technological advancement in the field of transportation, promising safer and more efficient roadways. In the pursuit of creating intelligent, self-driving vehicles, the integration of cutting-edge technologies plays a pivotal role. The focus of Lab 1, titled "Collaborative Perception for Autonomous Vehicles", is about the developing and understanding how the Lidar datas are useful for this field to detect the obstacles, then the environment around the vehicle.
Specifically, the lab can be divided in 4 parts build incrementally to get first the 3D model of the environment, then the so-called BEV image (Bird Eye view) which is the based for the control of the vehicle.
In this laboratory, different vehicles and infrastructure (IRSU) have been considered with different LADAR devices attached to it.

2 Single actor

In the first part of the lab, only one vehicle has been considered with its own 32-channel LADAR.
The problems to manage in this part have been mainly two.
The first one, how to transform the boxes data received from the sensor, in such a way that they were referred in the frame of the vehicle (actor).
The second, having the boxes , composed by the position of the center, length, width,height and orientation, has been how to extract the corners of the different faces of it. The corners are essential for plotting the boxes in the reconstruction of the environment.

2.1 Boxes in the actor frame

The first challenge we faced, has been transforming the boxes data from the sensor frame, into the car one (actor frame).
For doing that an Homogeneous transformation matrix has been applied to the position of the center of each boxes detected by the sensor.
To reach this goal the Homogeneous transformation between sensor and actor frame has been obtained through $getsensorTactor()$ and applied to each center of the box through $applytf()$. Furthermore, for a correct visualization of the boxes, also its orientation should be changed in the correct frame transforming the rotation matrix in quaternion and then in yaw, sometimes if the matrix is not perfectly orthogonal we can't compute the quaternion but we need to compute directly the yaw angle between actor and ego .
Anyway, the yaw of the boxes in the actor frame is equal to that in the sensor frame minus the one between sensor and actor one.
In this case, the two reference frame have only translation along the z-axis, then the orientation of the boxes is the same in both reference frames, so this passage can be neglected.

2.2 Extraction of the corner

The second challenge was to calculate the coordinates of the eight corners of a 3D bounding box. This type of bounding box is commonly used to represent the spatial extent of objects in three-dimensional space.
To solve this problem has been implemented the function*boxtocorner*.
The function takes a single input parameter, *box*, which is expected to be a NumPy array with seven elements: the 3 coordinates of the center of the bounding box in 3D space,the 3 dimensions of the bounding box along the x (length), y (width), and z (height) axes and the rotation around the z axis.
The function starts calculating a rotation matrix, based on the rotation angle in input. This rotation matrix is crucial for transforming the dimensions of the bounding box into the correct orientation in 3D space.
Then, some offsets have been defined, which represent the offset of each corner from the center of the bounding box. These offsets are specified based on the dimensions of the bounding box and defined as the combination of positive and negative offsets along the x, y, and z axes.
The coordinates of each corner are calculated by applying the rotation matrix to the offset and then adding the center coordinates. This step ensures that the corners are correctly rotated and positioned in 3D space.
In summary, this function is a fundamental part of 3D object representation and manipulation. It takes the parameters of a bounding box, considers its orientation in space, and computes the coordinates of its corners.
Following, the result of the retrieved environment considering only one actor in the environment:

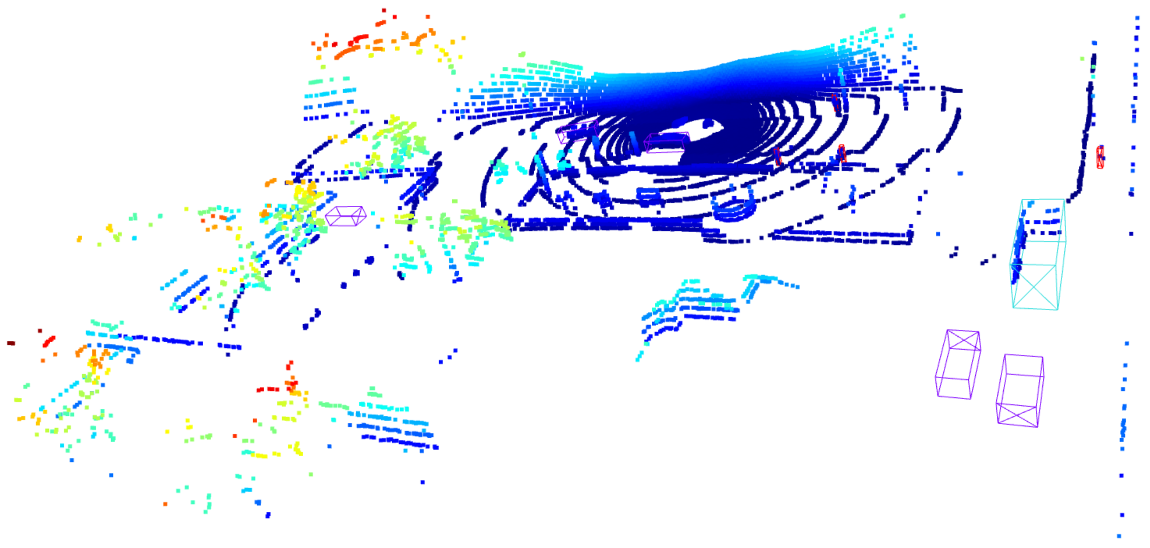


Figure 1: Environment with one actor's data

3 Multiple actors

In the ever-evolving landscape of autonomous vehicle technology, one of the key challenges lies in optimizing the vehicle's perception capabilities.
This part of the lab delves into the intricacies of leveraging Intelligent Road-Side Unit (IRSU) data to expand the vehicle's field of view and perception range and integrate all the different data coming from different vehicles within the environment.
The primary objective of this task is integrating data from diverse sources.
As previously said, this task is based on the previous one, in fact here we considered that all the point clouds and the boxes are already referred in each actor frame.
In this part of the lab we have faced two main difficulties, transforming the different **point clouds** and the **boxes** in the reference frame of the ego vehicle.

3.1 Multiple point clouds in ego frame

Having the different point clouds, each one expressed in the actor frame, is crucial to collect and merge for providing a consolidated and enhanced view of the environment in the ego frame.
The function *getavailablepointclouds* has been developed to satisfy this goal.
Firstly, It gets the essential Homogeneous transformations through *getactorTworld* between the ego vehicle (first vehicle in the list of the different frames) and a world reference frame, and between each other vehicle/infrastructures (actors) and the world frame.
In the second step, the function gets through *getpointcloud()* the different point clouds for each actors, and, applying the specific transformation , It gets the point cloud in the world frame.
Once the point clouds are expressed in the world frame, using the inverse of the transformation from the ego to the world frame, is possible get them in the ego frame, as desired.
Finally, the function does simply a merging of each point clouds, got from the others actors but expressed in the ego frame, with the point cloud got from the ego vehicle.

3.2 Boxes in different frames

This part of the lab consists in transforming all the boxes retrieved from the different actors within the environment, in the frame of the ego vehicle and merging it with the boxes already retrieved in its own frame.
For this step we have used the same pipeline applied to transform the different point clouds.
In this case, instead of simply points, we have some boxes composed by the 3 coordinates of the center, length, width, orientation and a label describing the type of the box (car, truck ecc).
Therefore we have briefly followed this steps to transform the coordinates of the center of each box in the ego frame:

- Got the boxes in the ego vehicle through *getboxesinactorframe* knowing that the first actor is the ego vehicle
- Got the transformation between ego frame and world frame with *getactorTworld*,(actor 0 is the ego one)
- Got for each actor the boxes with *getboxesinactorframe*
- Transformed their center in the ego frame with the two transformation from actor to world and world to ego (inverse of ego to world transformation) got with *getactorTworld*
- Finally concatenated with boxes already retrieved from the sensor of the ego vehicle.

Applying this pipeline the center of the boxes is expressed in the ego frame, but there is still a problem to solve, the orientation angle (yaw) of the boxes is still expressed with respect to the actor frame.
The correct orientation in the ego frame is obtained using the total Transformation matrix between actor frame and ego one (multiplication of the actor-world and world-ego transformations).
The latter is transformed through *Quaternion()* in quaternion representation and then through *quaternionyaw()* in yaw angle.
Using this method we run into some difficulties because for some errors , the transformation of the actor *"othervehiclebehind"* was not orthogonal. We have passed this problem not convrting the matrix in quaternion but directly computing the yaw angle from the matrices. Then, having the offset angle between the actor frame and the ego one (rotation along z axis), the correct orientation of the boxes in the ego frame can be seen as the difference between its orientation with respect the actor frame and the angle just computed.
Following, the resulting environment enhanced by the concatenation of the data of multiple actors:

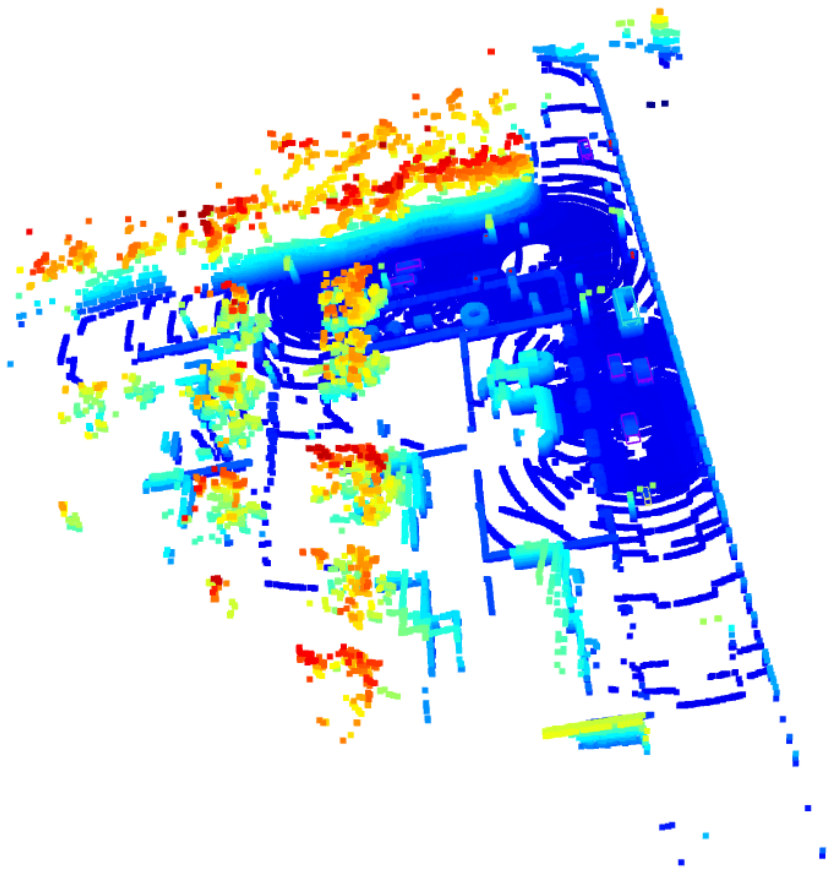


Figure 2: Environment with multiple actor's data

4 Bird Eye View

A Bird's Eye View (BEV) in autonomous vehicles refers to an overhead perspective of the vehicle and its surroundings. This perspective is usually generated by combining data from various sensors, such as cameras or lidar sensors, strategically placed around the vehicle.
The BEV provides a top-down representation of the environment, offering a comprehensive and panoramic view. This view is valuable for autonomous vehicles for several reasons such as Parking assistance, 360 degrees view of the environment, Mapping, localization and so on.
In summary, the Bird's Eye View is a crucial component of the sensor fusion process in autonomous vehicles, contributing to their ability to perceive and navigate their environment effectively.
Considering that all the previous task have been correctly implemented, in this part the aim is transforming all the 3D-data in the ego frame, in a 2D image as a top view of the environment.
The task can be divided in three main steps, a filtering of the point, transformation of the 3D points in 2D (pixels), the transformation of the 3D boxes in 2D ones.

4.1 Filtering points

For building a BEV image a filtering of the points can be useful for different reasons.

- Isolating and analyzing points within a specific region of interest, such as the area around the vehicle or within a defined region on the ground.
- Reducing noise in the data. This is particularly important in applications like object detection or localization, where accurate and relevant data points are essential for reliable algorithm performance.
- Operating on a reduced set of points within the specified range can lead to performance improvements, especially when dealing with large datasets. It allows algorithms to focus on the most relevant data, improving efficiency.

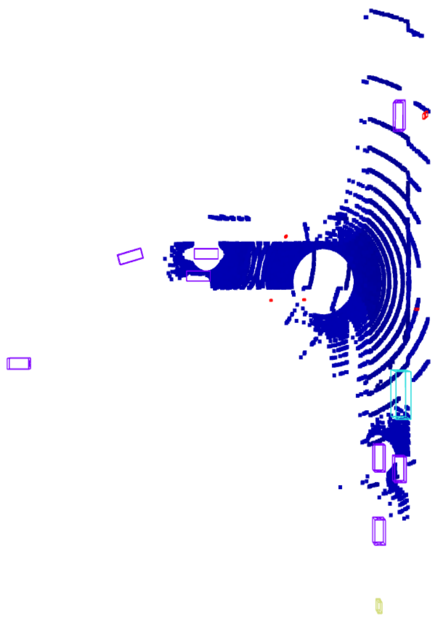
With the aim of filtering the points, the function *filterpoints* has been implemented. It simply takes the 3D points to filter, and the range desired in which the points must belong.
For each point given, the function checks if it belongs or not to the range, in positive case it add the point in a vector. In this way the output will be the vector containing only the points in the desired range.

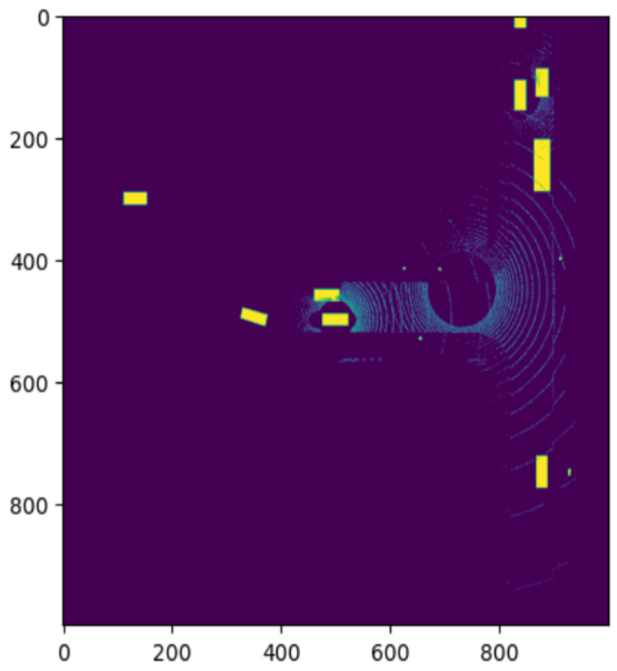
4.2 3D points in 2D pixels

The second step is converting the spatial coordinates of the filtered points into pixel coordinates within the Bird's Eye View image.
The function *pointstopixels*, takes a set of 3D points, the size of a Bird's Eye View image in pixels (height and width), and the resolution of the Bird's Eye View image in meters per pixel.
It calculates the pixel coordinates in the Bird's Eye View image for each point by taking the x and y 3D-coordinates and dividing them by the resolution. Then, it adds half of the height and half of the width of the Bird's Eye View image to the x and y coordinates, respectively. This adjustment is made to ensure that the coordinates are centered within the image.
The resulting array is transposed to have each row represent a pair of pixel coordinates (x, y).

4.3 From 3D in 2D Boxes

Finally, the function *boxtopixels*, takes in a set of 3D bounding boxes,the size of a Bird's Eye View image in pixels, and the resolution of the Bird's Eye View image. The function converts the 3D bounding boxes into pixel coordinates within the Bird's Eye View image and creates a binary mask indicating the pixels occupied by the boxes.
Once got the corners through *boxtocorner*, previously developed, the function selects the top face of the box, only useful for this type of image, using the convention chosen for the face. Finally, It transforms the 3D corners into 2D calling the function previously explained, and create a binary mask where the non zero values represents the pixels occupied by the bounding boxes and returns it.
This function is crucial for generating a binary occupancy map in the Bird's Eye View image. The resulting mask provides a clear representation of the areas occupied by the 3D bounding boxes in the top view.
The binary mask can be further used in conjunction with other perception and planning algorithms to make informed decisions about the vehicle's path, taking into account the presence of obstacles represented by the bounding boxes.
Here, the resulting Filtered environment and BEV image:


(a) Environment with filtered points



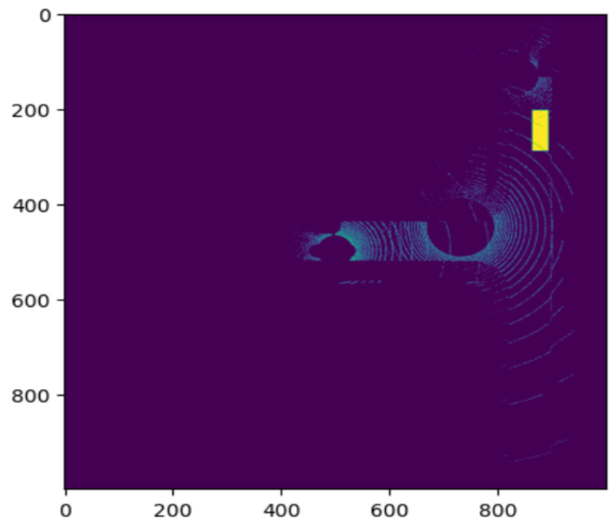
(b) BEV images

Figure 3: BEV

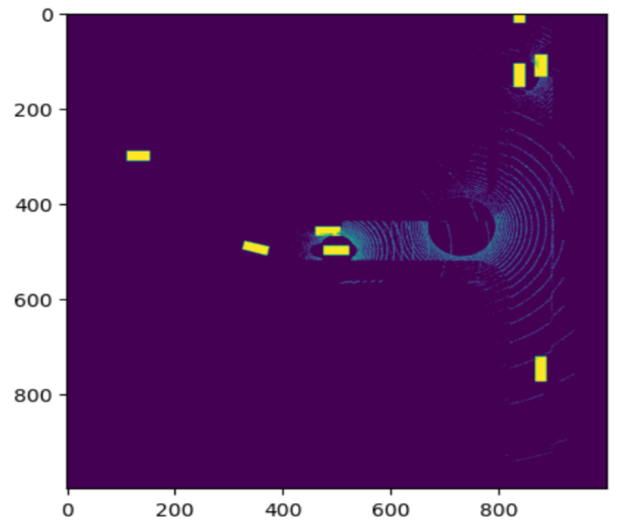
5 Segmentation of the obstacles

The last goal that we have reached during this lab, was to develop a method to segment only some objects in the environment according to their type.
In this part indeed, the object boxes contains also a label called *"class"* that describes the type of the object of the bounding box.
The *detectionobj()* takes in a frame number, a list of actors and a specified detection method. The function filters the objects based on the specified method and returns a NumPy array containing the bounding boxes of the detected objects. The function firstly gets all the boxes in ego frame through *getavailableboxesinegoframe*, previously implemented, then It defines numerical identifiers (0.0 for car, 1.0 for truck, 2.0 for motorcycle, and 3.0 for pedestrian), finally the bounding boxes are filtered based on the label of the box, using a series of conditional statements.
The filtered bounding boxes are stored in separate lists and the appropriate list is selected based on the specified *method*.
The value of method can be of different type based on the object that the user wants to segment out, for this reason we decided to put some buttons to allow the user to chose easier the type of the object. Then the method will be assigned.
The *detectionobj()* allows for the selective extraction of bounding boxes for specific types of objects (cars, trucks, motorcycles, pedestrians) or all objects in the scene based on the specified detection method.
In the context of Bird's Eye View (BEV) images in autonomous vehicles, this function can be useful for isolating and analyzing specific types of objects. For example, it enables the

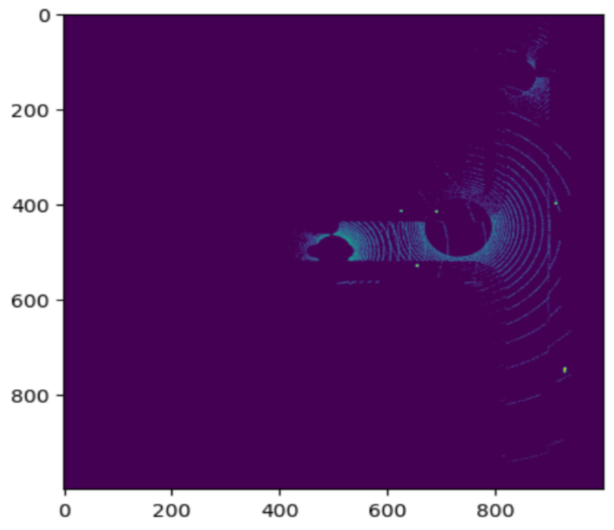
extraction of only the bounding boxes corresponding to cars or trucks, which can be crucial for tasks such as object tracking, collision avoidance, and overall scene understanding.
The ability to filter objects based on their type allows for more targeted processing and analysis, improving the efficiency of subsequent algorithms and decision-making processes in autonomous systems.
Here the resulting Filtered environment and BEV image with a filtering on the boxes:



(a) Truck



(b) Cars



(c) Pedestrians

Figure 4: Segmentation of the objects

6 Conclusion

The lab has provided valuable insights into the challenges and solutions associated with processing and integrating data from multiple actors in the context of autonomous vehicle perception.
One of the primary difficulties encountered was orienting the bounding boxes correctly, especially when dealing with multiple actors with varying orientations. Transforming the boxes into the frame of the ego vehicle required careful consideration of both translation and orientation, involving Homogeneous transformation matrices and quaternion representations.
Throughout the lab, we learned the importance of addressing challenges in transforming data to a common reference frame, as this is crucial for collaborative perception in autonomous systems. The ability to accurately represent the environment in a unified frame enables better decision-making for the vehicle.
Additionally, the lab emphasized the significance of sensor fusion and collaboration among intelligent road-side units (IRSUs) and vehicles.
Integrating point clouds and bounding box data from diverse sources presented challenges in terms of coordinate transformations, but overcoming these challenges was essential for a comprehensive understanding of the environment.
The implementation of a BEV was a key highlight, providing a top-down perspective of the environment. This representation is valuable for various applications, including mapping, localization, and obstacle detection.
Furthermore, the lab demonstrated the capability to segment specific objects based on their types, showcasing the flexibility in processing and analyzing data selectively. This functionality is crucial for tasks such as object tracking and collision avoidance.

In summary, this lab has equipped us with essential skills in handling complex data integration challenges in the realm of autonomous vehicle perception. The learned techniques and methodologies contribute to the broader goal of enhancing the safety and efficiency of autonomous systems on the road.