Politecnico di Torino

Master course in Computer Engineering

Master Degree Thesis

Exploiting infrastructure sensors for advanced vehicle manoeuvring assistance

Supervisor
Prof. Enrico Magli

Cosupervisors
Ing. Daniele Brevi
Ing. Edoardo Bonetto

Candidate
Federico Princiotto
s276173

Accademic Year 2021–2022

Abstract

In the last decade, there has been a huge increase in studies regarding autonomous vehicles and their use in real scenarios. Autonomous vehicles are now becoming safe and reliable also in everyday traffic, thanks also to a great number of sensors used to perceive the world around them. However, there are situations in which those sensors may not be so effective, mainly if something is blocking their view.

Smart infrastructure can help connected vehicles, providing information about the road status even before the vehicle is able to sense it.

In this thesis, it has been developed an entire software pipeline that extracts information from roadside sensors data and it uses this information to suggest to connected vehicles the manoeuvres to perform at an intersection. The roadside unit (RSU) is equipped with a camera and a LiDAR; data from these sensors are processed to identify all road actors, estimate their positions and predict their future trajectories.

The software architecture is composed of a set of modules that communicate with each other through a broker; this architectural choice provides high flexibility to the system, allowing to update or add modules easily. The main modules of the system are the ones that identify road actors from the video, merge this information with the data from the LiDAR to compute the actors' position and volume, and use this position to compute their future trajectories.

To ensure the consistency of the data extracted from the camera and the LiDAR when they are merged, this thesis also deals with the calibration and synchronization of those sensors.

The computation will not be performed on the roadside unit itself, but instead in a server located at the edge of the mobile network (Edge Server), that will receive sensors data flows from one or more RSUs.

Acronyms

5GAA 5G Automotive Association.

AABB Axis-Aligned Bounding Box.

ADAS Advanced Driver-Assistance Systems.

AMQP Advanced Message Queuing Protocol.

C-ITS Cooperative ITS.

CAM Cooperative Awareness Messages.

CPS Collective Perception Service.

CTRA Constant Turn Rate and Acceleration.

GPU Graphics Processing Unit.

HMI Human Machine Interface.

HV Host Vehicle.

IMA Intersection Movement Assistance.

ITS Intelligent Transportation Systems.

JSON JavaScript Object Notation.

KPI Key Performance Indicator.

LiDAR Laser Imaging, Detection, And Ranging.

LSTM Long Short-Term Memory.

MCS Maneuver Coordination Message.

MEC Multi-access Edge Computing.

NTP Network Time Protocol.

OBB Oriented Bounding Box.

OpenVINO OPEN Visual Inference and Neural network Optimization.

OSM OpenStreetMap.

RNN Recurrent Neural Network.

RSU Roadside Unit.

RTCP Real Time Control Protocol.

RTP Real time Transport Protocol.

RTSP Real Time Streaming Protocol.

RV Remote Vehicle.

SAE Society of Automotive Engineers.

SDK Software Development Kit.

SUMO Simulation of Urban MObility.

TraCI Traffic Control Interface.

V2I Vehicle to Infrastructure.

V2P Vehicle to Pedestrian.

V2V Vehicle to Vehicle.

V2X Vehicle to Everything.

VRU Vulnerable Road User.

YOLO You Only Look Once.

Contents

Chapter 1

Introduction

In the last years, a lot of work was done to increase the safety of the roads. Vehicles are getting cleaner, safer and smarter, and autonomous vehicles are becoming a tangible reality. Even though fully autonomous vehicles are still very rare and often cannot travel with other vehicles, it is not uncommon to have some degree of Advanced Driver-Assistance Systems (ADAS) in the everyday cars (for example, cruise control).
The Society of Automotive Engineers (SAE) classifies autonomous vehicles in a scale from 0 (not automated) to 5 (fully automated) [33]:

- **LEVEL 0**: the system can issue warnings to the driver;
- **LEVEL 1**: the system shares control with the driver in a few well defined scenarios (for example lane keeping assistance or automatic emergency brake);
- **LEVEL 2**: the system will drive itself, but the driver is required to always be careful and correct the driving if the system fails;
- **LEVEL 3**: the system will drive itself, without the need from the driver to monitor it; it is still possible for the system to explicitly ask for the driver's assistance for some manoeuvres;
- **LEVEL 4**: in some specific circumstances, the vehicle is about to drive itself without never asking assistance to the driver;
- **LEVEL 5**: no driver assistance required, in any circumstance.

Any smart vehicle is equipped with a variety of sensors (for example camera, LiDAR or RADAR) that can obtain data from the surroundings and use it to provide any level of assistance to the driver. However those **sensors can only sense in the near surroundings of the vehicle**, and this might lead to a **lack of safety in a not common scenario**, for example, when the scene is characterized by a high level of occlusion.
One solution to this problem can be the Vehicle to Vehicle (V2V) communication: in a scenario characterized by an high number of smart vehicles moving on the road, vehicles could connect to each other and send both what they sense and their intention, in order to coordinate each other. However, this is not easily feasible **in the scenario of the everyday traffic**, in which the smart vehicles are a small niche compared to the quantity of traditional ones. In this transitional scenario **the vehicle's sensors may not provide enough information** to the single vehicle to understand correctly what to do.
For this reasons, to help the integration of autonomous vehicle into the everyday traffic, it is also needed the assistance of the infrastructure.

The objective of this work is to develop a system that can use information extracted from the infrastructure at an intersection to help smart vehicles to perform manoeuvres safely. In Section 1.1 there is an high level explanation of how the infrastructure can assist smart vehicles; in Section 1.2 it is described more in detail the Intersection Movement Assistance (IMA) task; Section 1.3 reports the structure of the thesis.

1.1 V2I and V2X

The Vehicle to Infrastructure (V2I) communication allows smart vehicles to **obtain information about the road status from the infrastructure itself**; this means that any vehicle can receive an high level view of anything in that section of the road, without the requirement of the visibility from the vehicle's sensors itself.
Furthermore, **also a vehicle without smart sensors could, potentially, connect to the infrastructure and act as a smart one** in some specific sections of the road.
The Vehicle to Everything (V2X) communication is a set containing any type of

communication available to the vehicles (i.e V2I, V2V and V2P).
Thanks to the V2X, vehicles had transformed from Autonomous systems into Cooperative systems, commonly referred to as Cooperative ITS (C-ITS). The C-ITS allows any road actor (vehicle or infrastructure) to exchange information with all the others, thus increasing the safety of the whole road environment.
The 5G Automotive Association (5GAA) identified **7 groups of use cases** for the V2X technology [13]:

- **Safety**: this group includes any use case that improves the safety of the vehicle and the driver, such as emergency braking and collision warning;
- **Vehicle Operations Management**: this group includes the feature that can allow the vehicle manufacturer to monitor and improve the performance of some portion of the vehicle, such as remote status monitoring or software updates;
- **Convenience**: this group includes any feature that can provide some non mandatory value to the driver, such as cooperative navigation or autonomous parking;
- **Autonomous driving**: this group includes any feature that could help an autonomous vehicle of level 4 or 5;
- **Platooning**: platooning means that a number of smart vehicle move one after the other, with a small safety distance, like a train [19]; this could increase safety, while reducing gas emission and traffic;
- **Traffic Efficiency and Environmental friendliness**: this group includes the services that provide values to the infrastructure or the city, such as smart routing, green lights optimization and traffic jam information;
- **Society and Community**: this group includes the features that provide a service the society and the public, such as VRU protection and crash report.

The main constraint of the V2X in a real use case scenario is the **high reliability and low latency of the connection** between the various actors, furthermore automotive use cases usually involve large amount of data that has

to be processed and sent to the vehicles quickly [22]; the main technology used to solve all of those requirements is **edge computing**.
Edge computing refers to a series of techniques used to move the the storing and computational power used in Could computing closer to the source of the data, in order to reduce the latency [22]. This is usually done **deploying a Multi-access Edge Computing (MEC) server**, which is a cloud server located ad **the edge of the local cellular network**.

1.2 Intersection Movement Assist

The Intersection Movement Assist is a particular use case of the V2X concerning both the autonomous driving and the safety groups [13].
In this use case, a **connected vehicle** (Host Vehicle or HV), **near an intersection** receives safety messages from the infrastructure: the simpler implementation is to **send a warning messages to the driver if there is a risk of collision** performing a manoeuvre at the intersection (i.e., a vehicle from another lane could cause a car crash); a more advanced implementation could be to **suggest to the driver which manoeuvres are safe to perform** and which may cause collisions with other vehicles (for example, in a specific moment it could be safe to turn right, but not to turn left or go straight).
This use case can be divided in three simpler ones:

- if the Remote Vehicle (RV) is coming from the left, it will cause a conflict if it tries to go straight or turn left, but not if it turns right;

- if the RV is coming from the right, it will cause a conflict with any manoeuvre, if it doesn't stop;

- if the RV is coming from the front, it will cause a conflict only if it tries to turn left.

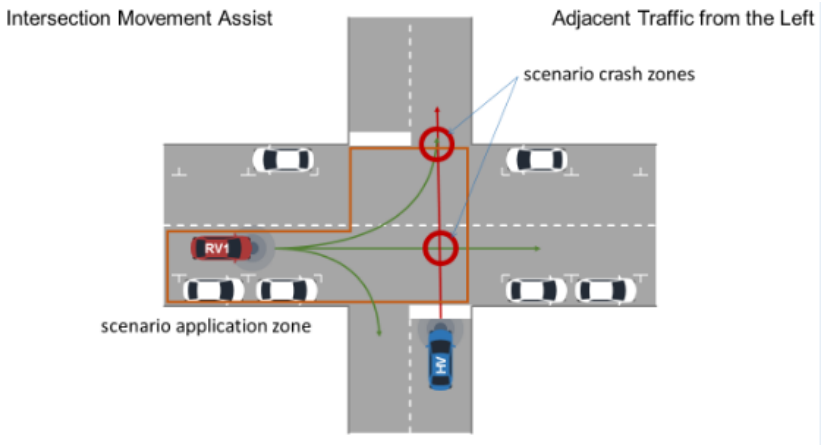


Figure 1.1: First use case [13]

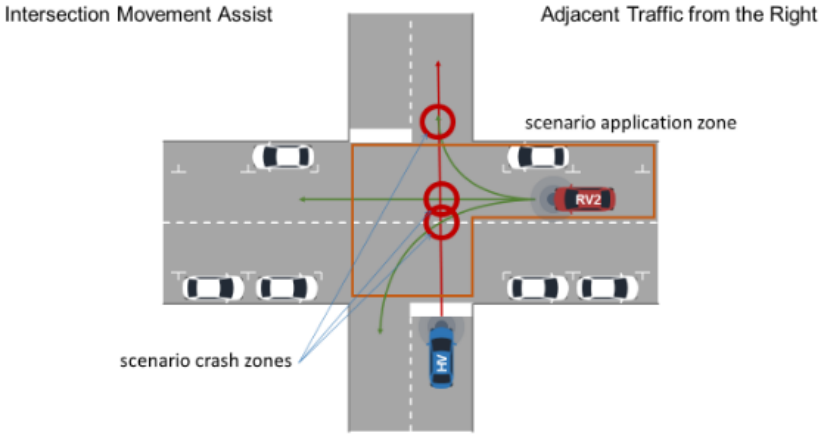


Figure 1.2: Second use case [13]

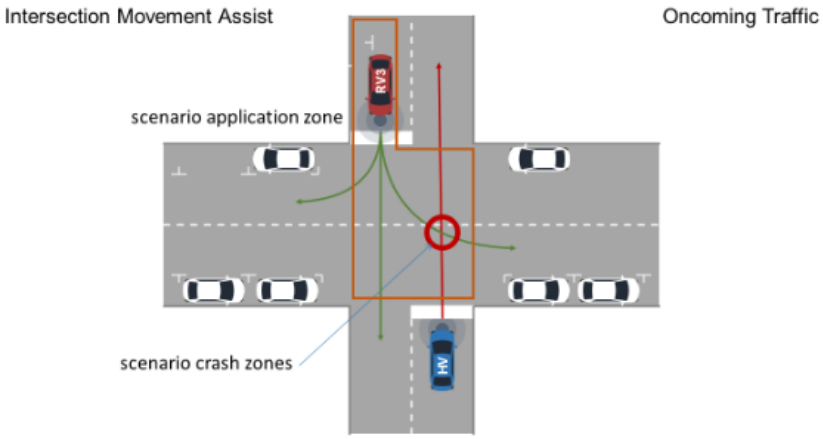


Figure 1.3: Third use case [13]

To correctly perform the IMA task, it is needed to **predict with high accu-**

racy the future movements of the RV(s); for this project, this also implies that is mandatory to **extract with high accuracy all the information** required to perform the prediction.

1.3 Thesis objective and structure

The objective of this thesis is to **implement a pipeline of containers that extracts from roadside sensors information about the vehicles** in an intersection and uses those information to **predict the vehicles' future path** and then **send manoeuvre suggestion** to the connected vehicles. The thesis is structured as follows:

- in Chapter 2 it is explained in detail the **project's use case**, first defining the **scenario** and then the **hardware available**;
- in Chapter 3 it is explained how the **two sensors (camera and LiDAR) are calibrated and synchronized**, in order to ensure the consistence of their data;
- in Chapter 4 it is explained the **high level view of pipeline**, listing all the containers and explaining what they to and how they interact;
- in Chapter 5 are explained more in detail the **implementation** of the containers that use the sensors' information to **extract knowledge about the vehicles at the intersection**;
- in Chapter 6 are explained more in detail the **implementation** of the containers that **predict the future path of the vehicles** at the intersection; in Section 6.3 there are the results of the prediction module tested both on an RSU and a simulated scenario;
- Chapter 7 **summarizes** the work done in this thesis and **suggests future improvements** that can be still performed.

Chapter 2

Project description

2.1 Use case description

The main goal of this thesis is to develop a Intersection Movement Assistance (IMA) service using the technologies presented above. The main functions of this service are:

- **Forward a video stream of the intersection to an Human Machine Interface (HMI)** in a smart vehicle connected to the system;
- **Provide live information about all detected road actors** (i.e., pedestrians and vehicles) and about the detected freespaces (i.e., road sections which are guaranteed to be free at that moment). This information will be sent to all connected vehicles in a well-defined automotive message format, compliant to the Collective Perception Service (CPS) standard.
- Predict future trajectory of the detected actors, to **generate alerts if there is a high chance of "dangerous event"**. This alert can be sent to anHuman Machine Interface (HMI) to warn the driver or directly to an autonomous vehicle.
- Suggest the driver (or the autonomous vehicle) if a specific manoeuvre is feasible without any risks or not, at any moment.

In the Figure 2.2 are shown two example of how the IMA service will work: in (a) the host vehicle can safely turn right, while turning left or going straight could cause a collision with the other vehicle; in (b) can safely turn left and right,

but cannot go straight. This suggestions are based on the trajectory prediction of the remote vehicle and are shown to the user through and HMI as colored arrows.

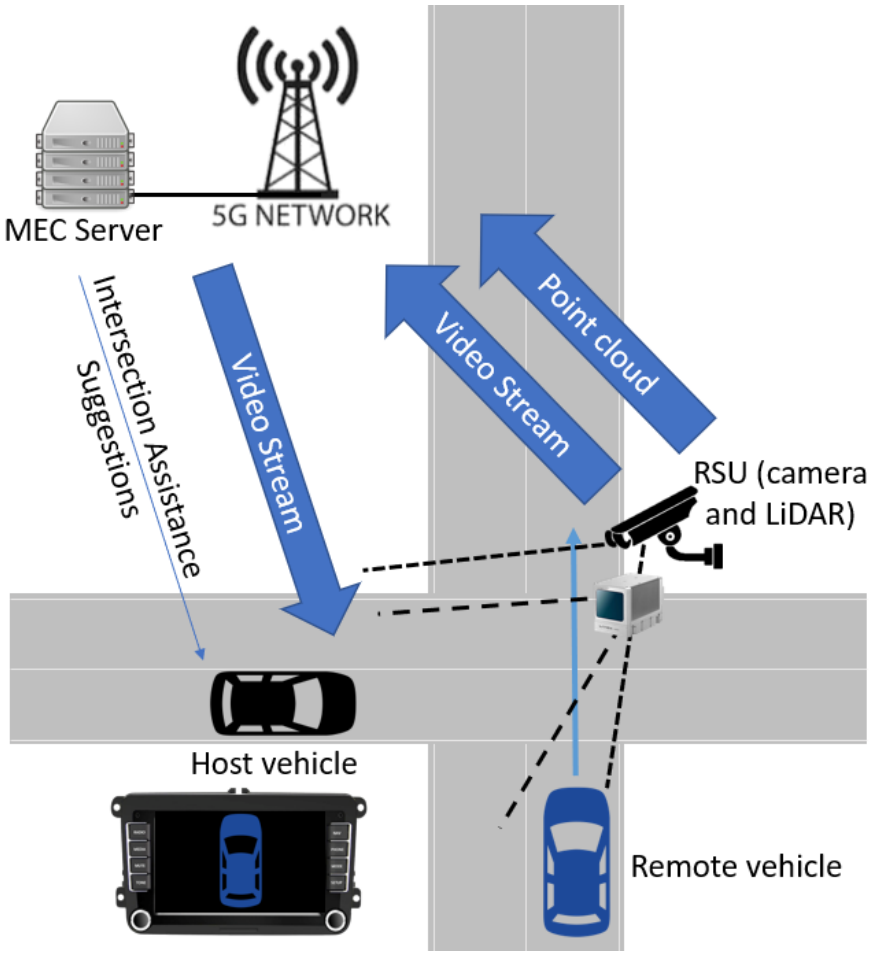
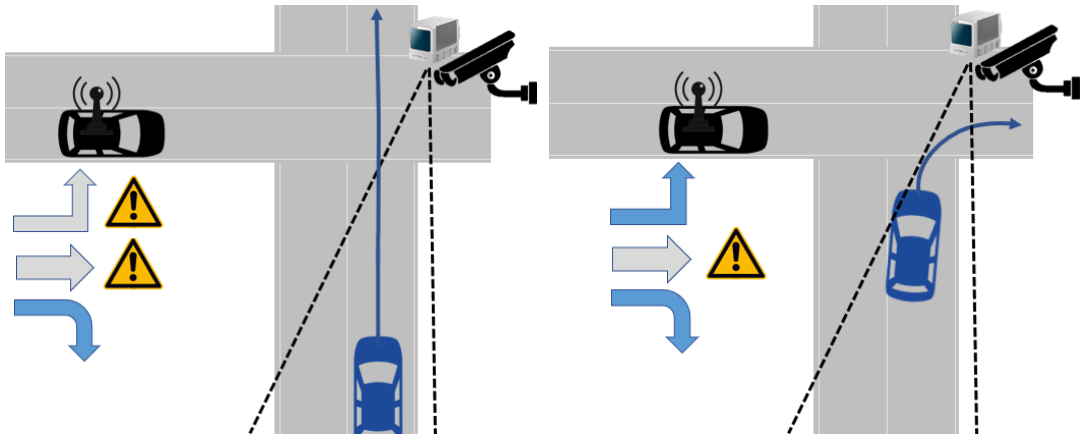


Figure 2.1: Use case scheme



(a) (b)

Figure 2.2: IMA examples

2.2 Hardware components

This project is divided into two main hardware components:

- Roadside Unit (RSU);
- MEC Server.

2.2.1 RSU

The RSU is composed of: a camera, a LiDAR, an embedded board, and connectivity components (4G and 5G antennas).

- **The camera** is a **Wisenet XNO-8080R**.
  It supports the definition of multiple RTSP streams, up to the quality of 5 megapixels. It also supports the RTCP protocol, which will be useful to compute latencies.
  Moreover, it is a **varifocal camera**, so it is possible to adjust the lens focal length to adapt the view to different scenarios without only relying on digital zoom.

- **The LiDAR** is a **Livox Horizon**.
  The Livox Horizon **sends 240.000 points per second, divided into packets of 96 points sent every 0.4ms**.
  It has a horizontal field of view of 81.7 degrees and a vertical one of 25.1

and can see up to a distance of 260 meters.
Moreover, it can also measure the object reflectivity and it can give a proximate measure of how reliable each scan is.

- **The embedded board** is the component managing the computations in the RSU.
  Since the heavy computation will be performed inside the MEC Server, it is possible to use an arm-based embedded board, that will only be used to **read sensors' data and forward them** to the server for the main computations.

What is a LiDAR?

A LiDAR is a sensor designed to compute distances. It works by flashing several light beams in different directions and computing the round trip time for each beam; knowing the direction followed by each beam and its round trip time, the LiDAR can compute easily the XYZ coordinates of the point hit by the beam.
With a sufficient number of beams, it can also be used to create 3D models of some objects.

2.2.2 MEC Server

The MEC server is an high-performance server located on the edge of the mobile network. This components follows the edge computing paradigm [26]: moving the server and their computation closer to the user to reduce the latency and to save bandwidth.
The MEC Server is the component performing the main heavy computations and providing services to the users.

Chapter 3

Sensors configuration

Before performing any analysis on the data, we need to ensure that all the data coming from the two sensors are consistent. This means that it was needed to perform both a **calibration** and a **synchronization**.
The first one is needed because the two sensors cannot be in the same location, or with the same rotation, and **even a small difference in the rotation** of the LiDAR and the camera **can cause** (at a high distance) **huge errors**.
The second one, instead, is important because **LiDAR and camera data are sent and travel at different speeds in the network**; it cannot be assumed that data received by the MEC server simultaneously, were also generated simultaneously.
This chapter is structured as follows:

- in Section 3.1 it is explained the calibration procedure involving only the camera;
- in Section 3.2 it is explained how the LiDAR and camera mutual position is derived with a calibration procedure;
- in Section 3.3 it is shown how the synchronization can be performed and how to retrieve timestamps from both camera and LiDAR;
- in Section 3.4 are show some other calibration, designed to map the point-cloud axis system in a more usefull one.

3.1 Camera calibration

3.1.1 Intrinsic parameters

The intrinsic parameters of a camera are a matrix that can be used to map any 3D point (in **the camera reference frame**) to the pixel in which it will be shown in the image frame. [2].

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & x \\ 0 & f_y & y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Pixel = Intrinsic * Coordinates

Where:

u, v: pixel coordinated of a point

f_x, f_y: focal lengths of the camera

x, y: half of output image size

X, Y, Z: point coordinated in the camera reference frame

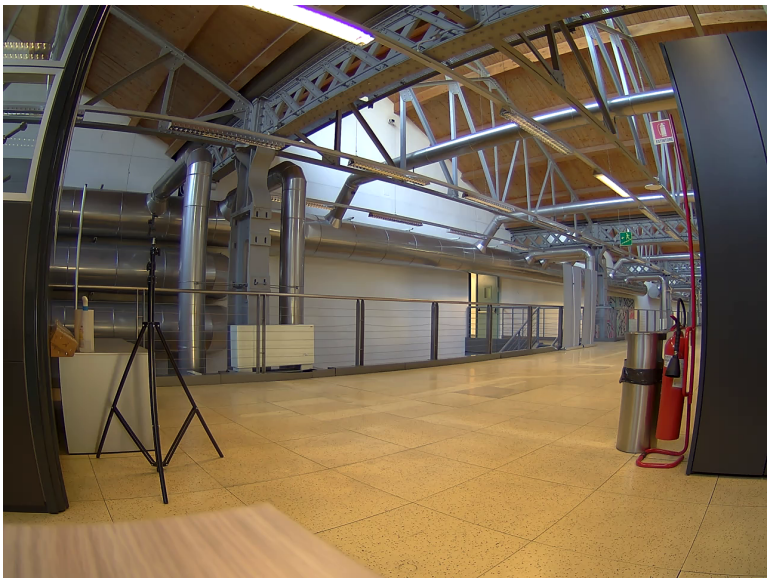
This matrix **can be computed knowing the focal length of the camera and the result image size**. However, it is not common for camera datasheets to include the focal length; if the focal length is not available, it can be computed knowing the field of view:

$$f_x = horizontal\_fov * x/2$$
$$f_x = vertical\_fov * y/2$$

3.1.2 Distortion vector

The **distortion vector is a set of 5 parameters that allows to correct the radial and tangential distortion of the image** caused by the lens [2]. The distortion vector, differently from the intrinsic matrix, is not available in the datasheet of the camera, and instead must somehow be computed. **The most efficient way to compute it is with a chessboard calibration**: different images of a chessboard are taken (very close to the camera) and the program tries to tune the 5 parameters iteratively to rectify the images. The code to perform this calibration is easily available online [25].

Results



(a) Camera frame before the distortion correction

(b) Camera frame after the distortion correction

Figure 3.1

In the Figure 3.1 it is possible to see the difference between the frame obtained from the camera lens and the one obtained after the distortion correction: in the first one, all the vertical lines (for example, the gray closet in the left) are distorted.

3.2 LiDAR-camera extrinsic calibration

The intrinsic matrix allows to map any 3D point in the camera reference frame to the correct pixel. However, the points captured by the LiDAR are not in the camera reference frame, but in the LiDAR's one; to map a LiDAR point to the correct pixel, another step is required: computing an **extrinsic matrix that can map the LiDAR reference frame in the camera's one**. This extrinsic matrix is the physical transformation (rotation and translation) needed to transpose a reference frame into another [1].

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_1 \\ R_{21} & R_{22} & R_{23} & T_2 \\ R_{31} & R_{32} & R_{33} & T_3 \end{bmatrix} \begin{bmatrix} X_l \\ Y_l \\ Z_l \\ 1 \end{bmatrix}$$

Camera coordinates = Extrinsic * LiDAR Coordinates

Where:

X_c, Y_c, Z_c: 3D coordinates in the camera reference frame

X_l, Y_l, Z_l: 3D coordinates in the LiDAR reference frame

R: rotation component of the transformation

T: translation component of the transformation

This can be combined with what is known about the intrinsic parameters, to map any 3D point in the corresponding pixel.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & x \\ 0 & f_y & y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_1 \\ R_{21} & R_{22} & R_{23} & T_2 \\ R_{31} & R_{32} & R_{33} & T_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Pixel = Intrinsic * Extrinsic * LiDAR Coordinates

To compute the extrinsic parameters, it is mandatory to know 3D coordinates and image pixels of some points and the intrinsic matrix of the camera. To store coordinates and pixels, and then calibrate the system, a **semi-automatic pipeline** was developed.

The pipeline is structured as follows:

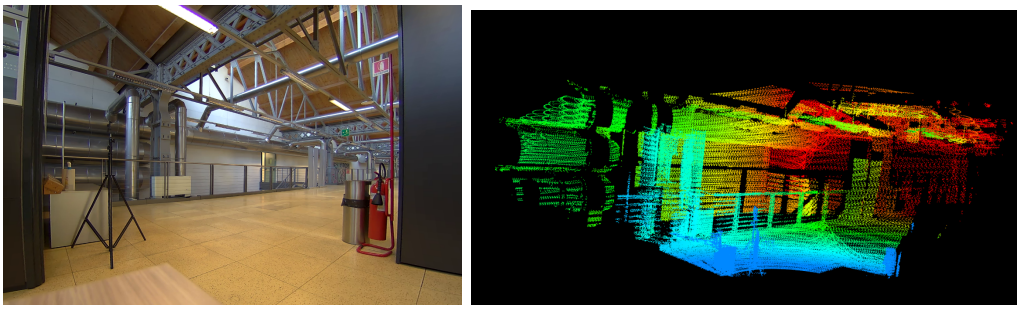
- First, in a still scene, a program **takes a long exposure LiDAR scan and an image from the camera and stores those in 2 files**. "Long exposure" in this case means "around 5 seconds", in order to have a very dense cloud (1.200.000 points); this scan is saved in a pointcloud file (pcd extension).

- Manually, the user opens the image in any image editor and **selects around 15/20 key points**, and stores their pixel coordinates in a file.
  Then, the user opens the stored pointcloud in the pcl_viewer provided by the pcl-tools package, picks the same points (in the same order), and stores their 3D coordinates in another file.
  This manual operation requires around 20 minutes to complete.

- Lastly, the user **passes the pixels, coordinates and intrinsic matrix files to a program that tries to optimize the extrinsic matrix**.

The program in the last section of the pipeline exploits the **ceres_solver library [23]** (an open source C++ library designed to solve least square problems) **to optimize the error of the pixel predictions of the key points**, tuning the extrinsic matrix slightly for each pixel-coordinate pair.
The extrinsic matrix is initialized as $\begin{pmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$, which is the "same position and rotation" transformation.

This program is very sensitive to any error in the pixel-coordinate pairs, so is always better to double-check when storing the points.
Also, more points on a straight line don't provide a lot of information, is better to have fewer points, but all around the scene.



(a) Camera view (b) LiDAR view

Figure 3.2

Results

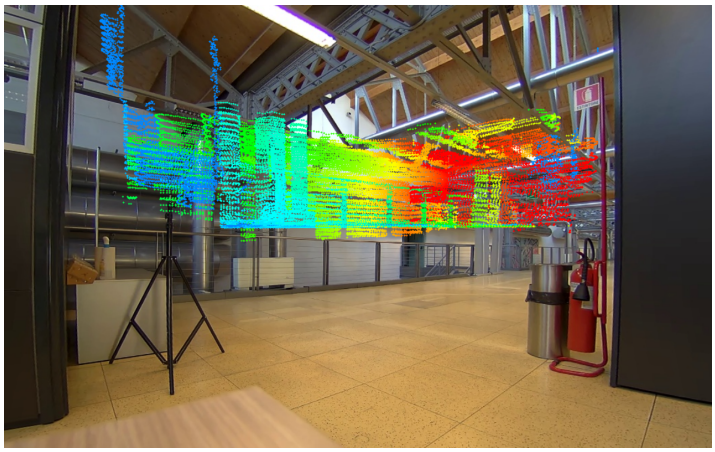


Figure 3.3: LiDAR and camera with no calibration

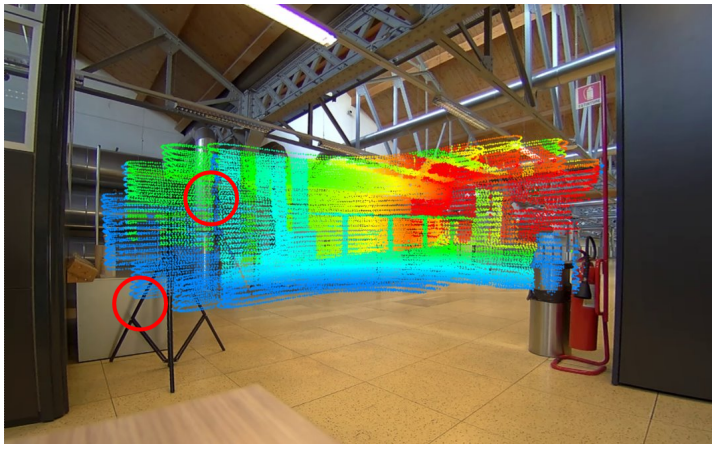


Figure 3.4: LiDAR and camera after adding a static positional offset

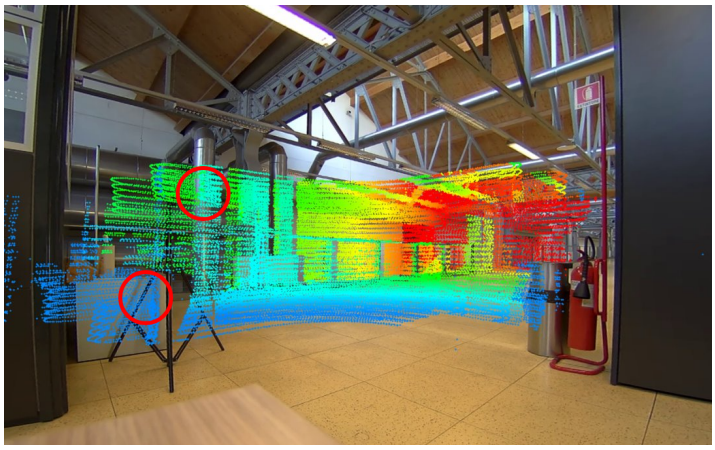


Figure 3.5: LiDAR and camera after full calibration

In the Figures 3.3, 3.4 and 3.5 it is possible to see that, even if adding a static offset improves the alignment of the two sensors, it does not solve the calibration problem: looking at the 2 highlighted sections, **it is clearly visible that the positions of the gray table and the one if the vertical pipe are estimated correctly in the Figure 3.5**, while they are not in the Figure 3.4. Furthermore, from a practical point of view, **it is simpler to perform a 20-minutes calibration procedure** from a remote location **than having to manually measure with complete accuracy the relative positions** of camera and LiDAR on the RSU.

3.3 Synchronization

The synchronization between the two sensors is needed in order to ensure the consistence of their data before merging them. Working with two totally different types of sensors, there are a lot of factors that may change the time synchronization of the messages, for example:

- message size of the different payloads, since heavier payload are slower to transmit (and so, are older when they are received);
- different internet protocol used (eg. transport-level protocol);
- eventual camera internal frame buffer, that may introduce delays.

3.3.1 Approach

The previous list shows that assuming that the last received pointcloud and frame were generated at the same time will not work: data generated at the same time, but sent with different network protocols and different payload sizes, will not arrive simultaneously to the receiver.
The solution is to keep a buffer of images and a buffer of pointclouds (with their corresponding timestamps), and always be at the slower sensor's pace; this may cause some (small) delays, but guarantees to always have consistent information.

3.3.2 The timestamps

LiDAR

When the LiDAR sends a scan (every 0.4ms, in a UDP packet) it also sends some information, one of those is an **Network Time Protocol (NTP) synchronized Unix epoch timestamp**. This means that when we generate a pointcloud it's trivial to associate it with a timestamp.

Camera

Retrieving a timestamp from an Real Time Streaming Protocol (RTSP) stream is, instead, much harder.
The **Real time Transport Protocol (RTP) packets** that compose the stream contain a timestamp, but is a **32-bit timestamp initialized at a random number**: it is useful only when used to compare two frames, but not if you need an absolute timestamp.
If the camera supports it, you could have also an Real Time Control Protocol (RTCP) stream. RTCP is a control protocol over RTSP and, **in the RTCP Sender Report packet (RTCP SR), the source sends also an NTP timestamp**.

However, even if the RTCP stream is available, **the code implementation to read its data is not trivial**. The OpenCV implementation of the VideoCapture class reads only the RTSP stream and it is not possible in OpenCV to read the RTCP packets. Even in the FFmpeg C library, there is no method

to read the RTCP timestamp. But the RTCP timestamp is indeed read by the FFmpeg library, and it is stored in a private data structure. To read, this structure, you need to compile the program including some private headers that are not present in the built library, but only in the FFmpeg source code. After the RTCP timestamp is read, the last step is to convert the NTP timestamp in milliseconds since UNIX Epoch, since they are different:

- the NTP timestamp is composed by 64 bits; the first 32 bit count the seconds from the January 1 1900, while the seconds counts the fractional seconds (with a resolution of 232 picoseconds) [3]

This did not totally solve the problem: **the RTCP SR packet** is not associated to each frame, but instead **is sent every 5 seconds**. This can be useful to estimate the network stability and latency, but not to associate a precise timestamp to each frame.

At the moment, there are two possible solutions:

- Assume that the network delay remains constant between the RTCP packets, and use this information to associate each frame to a timestamp;

- **Store the image in another protocol (such as AMQP) in order to manually add a timestamp** to the message. However, RTSP is designed to transfer images, and internally has some optimization to reduce the payload's size and message's latency; this means that using AMQP comes with the **tradeoff of adding latency**.

At the moment in this project is adopted the second solution, since is mandatory to have the timestamp of both camera and LiDAR in order to correctly synchronize them. Further tests will be performed in order to find an optimal solution.

3.4 Additional LiDAR calibrations

There are two other transformations performed on the LiDAR scan that have not yet been mentioned.

Floor z align

For some computation that will be seen in the next chapters, **I need the pointclouds to be aligned to the floor, with the floor points having z = 0**.
If the LiDAR were installed horizontally in the RSU, this would be as simple as adding an offset to the coordinates of the points. Otherwise, if the LiDAR is tilted toward the floor (for example, to have a better view of the vehicles on the road) the calibration is more complex.
The solution to this problem is to pick three floor level points on the pointcloud (with the same pcl_viewer tool used in the extrinsic calibration) **and use them to rotate the pointcloud** in a way that maps the plane defined by those 3 points into the z=0 plane.

North align

Because of the requirements of the project, **we need the pointcloud to have a specific set of axes: y positive to the north, x positive to the east, z positive upward**.
The LiDAR scan already has the positive z upward, so it is easy to rotate the XY plane in order to fulfill this requirement: similarly to what is done in the floor z align, **a point directly to the north is picked and is used to rotate the XY plane accordingly**.

Chapter 4

Software architecture

Each software module in this project is deployed as a Docker container; this approach provides a lot of benefits, such as **avoiding dependency errors and increasing the system flexibility**, since any container can run in any hardware component. Furthermore, this approach could allow creating replicas of some containers in order to scale or increase the performances.
The **whole system** that starts from the RSU sensors and ends with the path prediction **consists in a pipeline of containers**.
This chapter focuses on listing all containers and summarizing what each one does and how communicates with the other container, while Chapter 5 and Chapter 6 will focus more on detail in the implementation of some containers.

In the Figure 4.1 it is shown an high level view of the software pipeline. Some low level details are hidden, but is possible to see the high level flow of the information:

- the detection and tracking components use the camera data flow to detect all the road actors and associate them with an unique ID;

- the 3D bounding boxes component merges the information of the tracking with the points received from the LiDAR, in order to estimate the vehicles' positions;

- the vehicles' positions are used to predict their future manoeuvres.

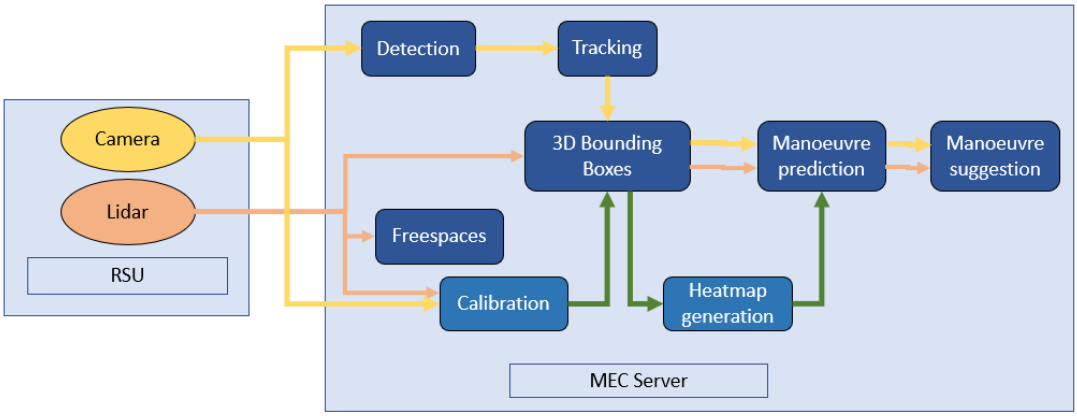


Figure 4.1: High level view of the container pipeline

This chapter is divided as follows:

- in Section 4.1 it is explained the container pipeline that sends the sensors' data to the MEC server;
- in Section 4.2 are shown the visualization containers used to provide visual information to testers and users;
- in Section 4.3 it is explained the pipeline that analyze the sensors' data to create high level information;
- in Section 4.4 are there are the container that perform the IMA service.

4.1 Sensors data

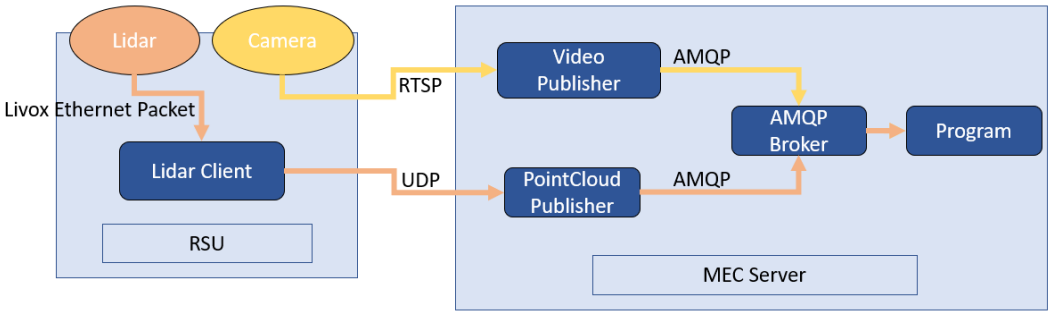


Figure 4.2: Sensors containers schema

Sensors data need to be sent from the RSU to the MEC server, where any program can use them.

The approach used is to have an **AMQP broker inside the MEC Server**, and perform all the inter-program communication through the broker. This approach simplifies the development, avoiding creating point-to-point communication.

4.1.1 Lidar data flow

Lidar Client

The Lidar Client container uses the LiDAR SDK to connect to the LiDAR and **receive all the scans, and then forward the point to the PointCloud Publisher**.
The main objective of this container is to **separate the logic of the PointCloud Publisher from the low-level SDK**, which may change depending on the LiDAR brand, model and firmware version.

PointClound Publisher

The **PointCloud Publisher container receives the points from the Lidar Client and accumulates them, before publishing to the AMQP broker**.
Before sending the AMQP packet, it set a property named "timestamp" into the packet, allowing the programs that will use the pointcloud to synchronize.
From a configuration file, **this container receives the list of topics in which it has to publish the pointcloud; different topics define different publishing frequencies** and different pointcloud sizes, so that any program can subscribe to the most suitable topic.
This container also performs the calibrations explained in chapter 3.4, floor z align and north align.

4.1.2 Camera data flow

Video Publisher

The camera publishes an RTSP stream, but, for the reasons discussed in chapter 3.3, we decided to wrap the images in an AMQP packet.
This container reads the image from the RTSP stream, corrects the distortion, encodes the image, and publishes it into the AMQP broker (adding the "timestamp" property).
To have a reliable timestamp, this container should be placed into the RSU;

however, this could add a huge delay if the network cannot provide enough bandwidth, so at the moment the configuration used for this project is the one of the Figure 4.2. Further tests will be performed in when the final hardware will be available to understand if the architecture in the Figure 4.3 is a viable solution.

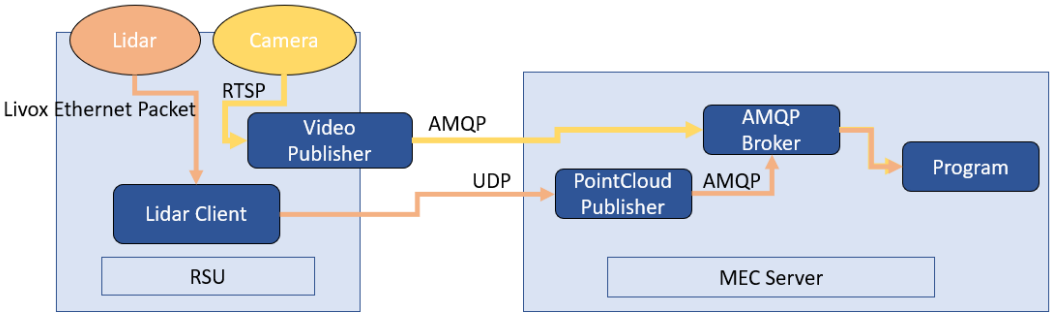


Figure 4.3: Alternative position of the Video Publisher container

4.2 Visualization containers

This section contains the containers that are used to provide video informations to an external user.

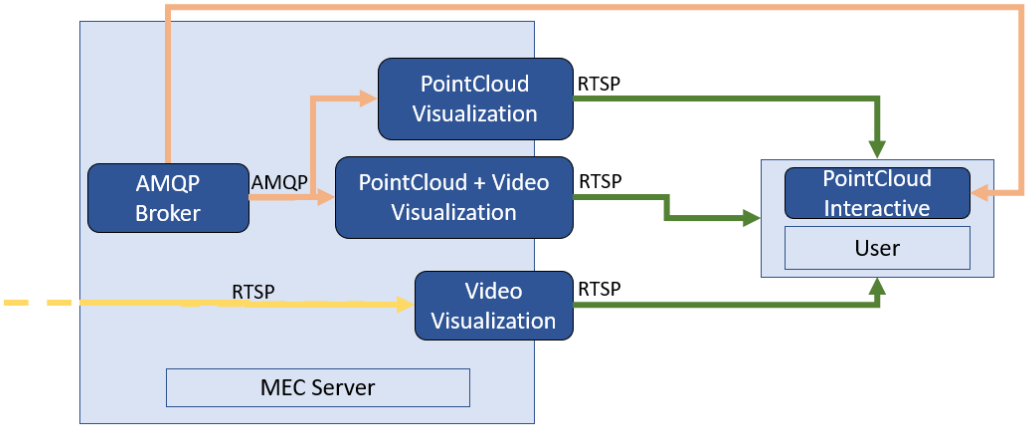
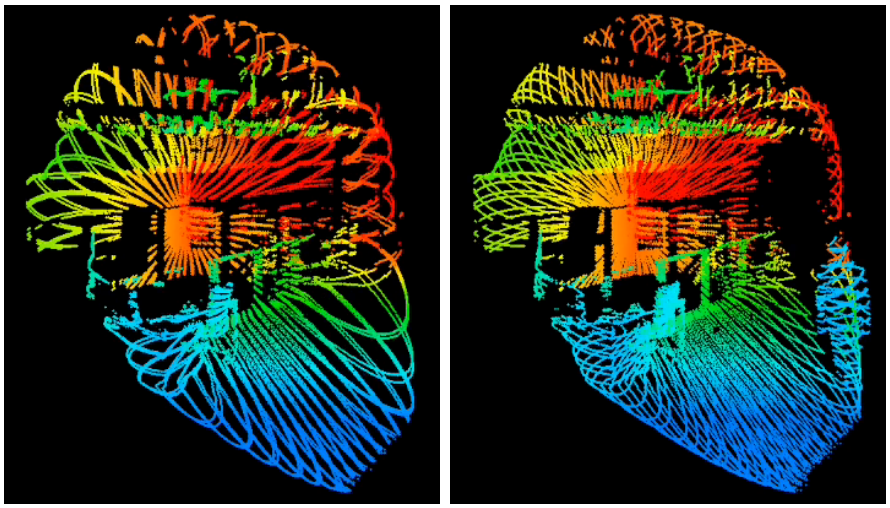


Figure 4.4: Visualization containers schema

4.2.1 Pointcloud Visualization

This container reads the pointcloud from the AMQP broker, and **creates an RTSP stream of the live 3D pointcloud, colored by distance, as viewed from a fixed position and orientation**.
Internally, this point of view modification is performed by creating another extrinsic matrix that emulates a different position of the camera. Color gradient and point of view are configurable using the configuration file.



(a) (b)

Figure 4.5: Pointcloud result RTSP stream

4.2.2 Pointcloud and Video Visualization

This container receives from the AMQP broker both the images and the pointclouds and provides their information merged in an easily understandable way: it **publishes and RTSP stream showing the camera video, adding to the images the points of the pointcloud in the corresponding pixel** (math in Section 3.2), **colored by distance**.
The distance color gradient and other parameters are configurable through a configuration file.

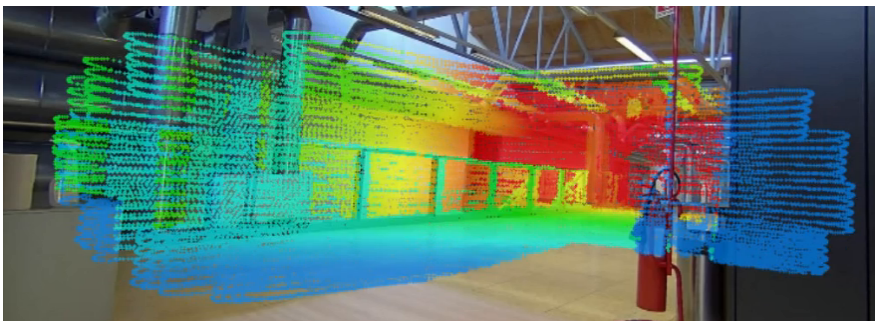
4.2.3 Video Visualization

The camera publishes an RTSP stream in the private network between the RSU and the MEC Server.

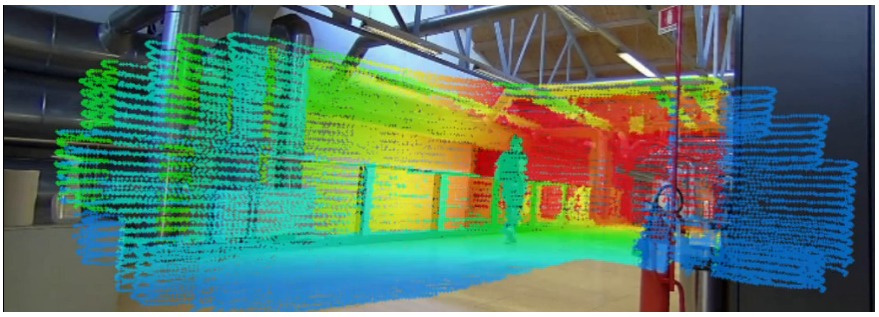
This container hosts a simple RTSP server and launches an FFmpeg command to **redirect the RTSP stream sent from the camera into an open port of the MEC Server**.

4.2.4 Pointcould interactive Visualization

This container runs inside the user's device and allows the user to directly connect to the AMQP broker and **create an OpenGL 3D view of the live pointcloud in his pc**.
The main advantage of this program is that allows a **live interaction with the pointcloud, using the mouse and the keyboard** to change zoom and point of view, impossible to do in an RTSP stream.



(a)



(b)

Figure 4.6: Pointcloud and Video result RTSP stream

4.3 Data analysis containers

This section contains the programs in charge of analyzing sensor data and providing high level knowledge about the position of the road actors at the intersection to the IMA service. The Detection and Tracking module detect road actors from

the video information, while the 3D Bounding Box module merge this information with the LiDAR points in order to find the actors' position.

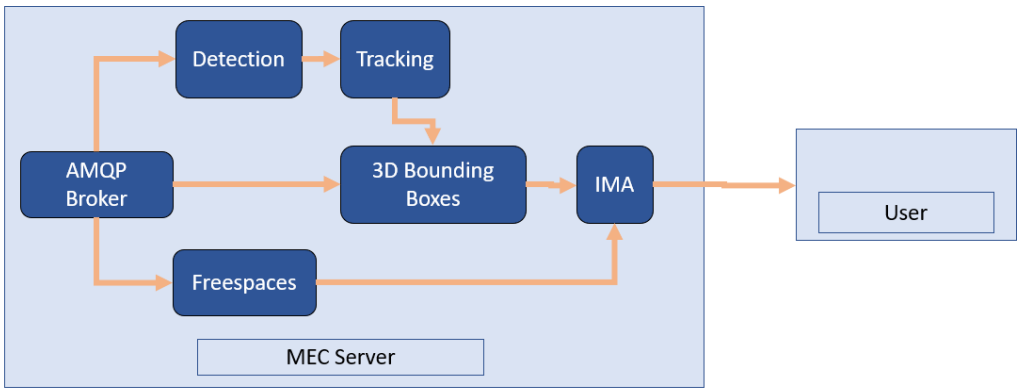


Figure 4.7: Data analysis containers schema

4.3.1 Detection

This container reads the video stream from the AMQP broker and **publishes a JSON with the detected objects** and their positions in the image.
Since other containers in the pipeline may work with a version of the same image with lower resolution, this containers provide also the possibility to scale the coordinates of the detection to be consistent with the lower resolution image.
More details about the implementation are available at Section 5.1

4.3.2 Tracking

This program reads the video and the result of the detection and **associates each detected object to an ID**, which needs to be constant between frames. Furthermore, it is also capable to learn some features of the objects in order to **track them even during the frames in which there is no detection available** (i.e. if, the detection is slow and takes several frames to send a detection, the tracking tries to compensate for those frames).
It also tries to map the tracking box positions into geographic coordinates.
Its result is published as a JSON to the AMQP broker.
The implementation of this program comes from the thesis work "Multiple Object Tracking and trajectory prediction for safety enhancement of autonomous driving" [16] previously developed at LINKS Foundation.

4.3.3 3D Bounding Boxes

This container uses the LiDAR pointcloud and the result from either the detection or the tracking to **compute a 3D bounding box around any object**. This allows us to have, for each detected object, its **3D position and volume occupation**.
Its result is published as a JSON to the AMQP broker. More details about the implementation are available at Section 5.3

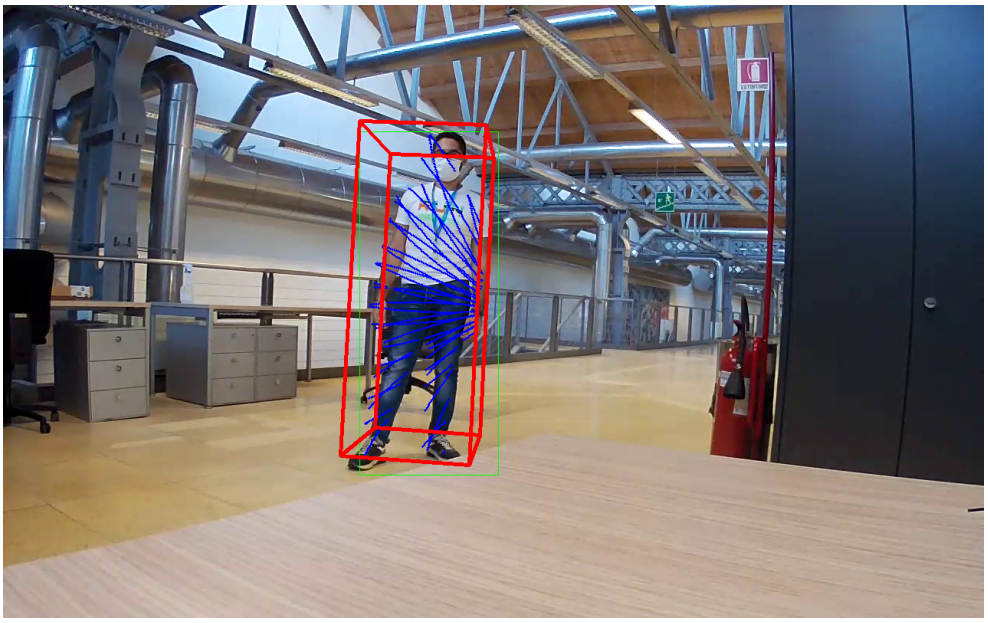
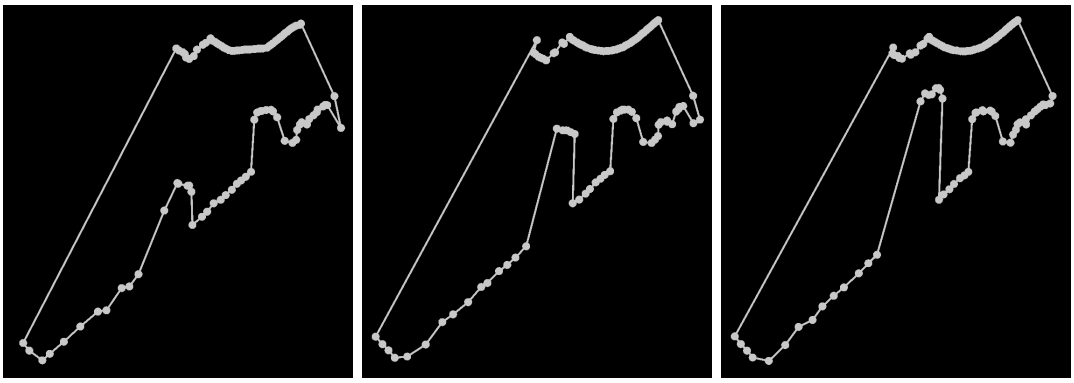


Figure 4.8: Bounding box (red) compared to Detection (green)

4.3.4 Freespaces

A freespace is defined as a section of the space that is known to be not occupied by any object.
This container **uses the LiDAR data to compute what freespaces (0 or more) are available at ground level**. To do so, this is the only container that truly requires the floor z align calibration Section 3.4).
Its result is published to the AMQP broker as a vector of polygons, since an obstacle near the RSU can cause the freespace to be divided in two separated areas; a polygon is defined by a list of points that mark the border of a freespace. More details about the implementation are available at Section 5.2



(a) (b) (c)

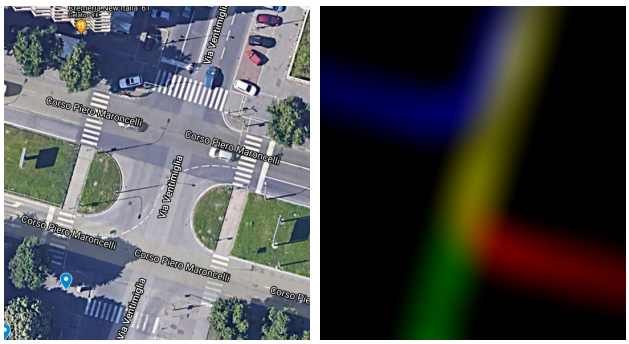
Figure 4.9: Different freespaces with a moving object and a still one

4.4 IMA containers

This section contains the programs that use the objects' information extracted by the previous containers to **provide high-level data that will be used for the IMA services**.

4.4.1 Heatmap generator

This container uses historical data of cars' positions in the intersection to **generate an RGB heatmap of the path made by a car to perform a specific manoeuvre**. Each possible manoeuvre (turn left, turn right, go straight) is mapped in a specific channel of the heatmap, as 3 separated heatmaps.
This heatmap will then be used by the prediction container to associate a predicted position to a specific manoeuvre. More details about the implementation are available at Section 6.1.



(a) (b)

Figure 4.10: An intersection and its heatmap, generated on simulated data

(a) (b)

Figure 4.11: An intersection and its heatmap, generated on real data

4.4.2 Prediction

This container **predicts the future positions of any detected vehicle**, reading the data receiver from the tracking and 3D bounding box and **applying to them a Kalman filter**.
Then, the container uses the heatmap to estimate, for each vehicle, the **probability that it will perform any of the 3 possible manoeuvres**.
This knowledge will be used to send warnings to a connected vehicle if a detected manoeuvre could cause any conflicts. More details about the implementation are available at Section 6.2

(a) (b)

Figure 4.12: Prediction example

4.4.3 Intersection movement assistant

This container receives the position of a connected vehicle and reads from the Prediction module the future manoeuvres of the other vehicles. Then, knowing the relative position of the vehicles and the topology of the intersection, sends to the vehicles a suggestion about which manoeuvres are safe to perform.

(a) (b)

Figure 4.13: Manoeuvre suggestion example

Chapter 5

Data analysis algorithms

This chapter contains the programs that use the sensors' data to extract meaningful knowledge to understand what is happening at the intersection.
This chapter is structured as follows:

- in Section 5.1 it is explained how the detection works, with a focus on the performance optimization on a limited hardware;
- in Section 5.2 it is show the logic that extract a freespace polygon from the raw pointcloud;
- in Section 5.3 it is explained how the 3D bounding boxes are computed from the detections and the pointcloud.

5.1 Detection

Usually, it is not too difficult to implement an object detection task using a neural network, thanks to the huge amount of works and researches performed in the last years. However, this project arose a problem that usually is not a concern in the detection task scope: **the MEC Server in which the detection** (and all the other containers) should run, **does not provide a Graphics Processing Unit (GPU)**, which is usually a mandatory requirement for any machine learning application. For this reason, any machine learning algorithm that will be used, needs to be able to run on a CPU.
Furthermore, since the system is designed to avoid car crashes and other dangerous situations, **this whole system needs to run as fast as possible**, in order to send alerts with enough notice to the drivers.

5.1.1 Yolo

Since speed and optimization are fundamental requirements, it was choosen to use the You Only Look Once (YOLO) approach [6]. YOLO is a family of neural networks designed to perform both detection and classification in one step, with the tradeoff of reducing the detection accuracy if compared to other solutions, such as two-stage object detectors [12].

5.1.2 YoloV3 and YoloV4

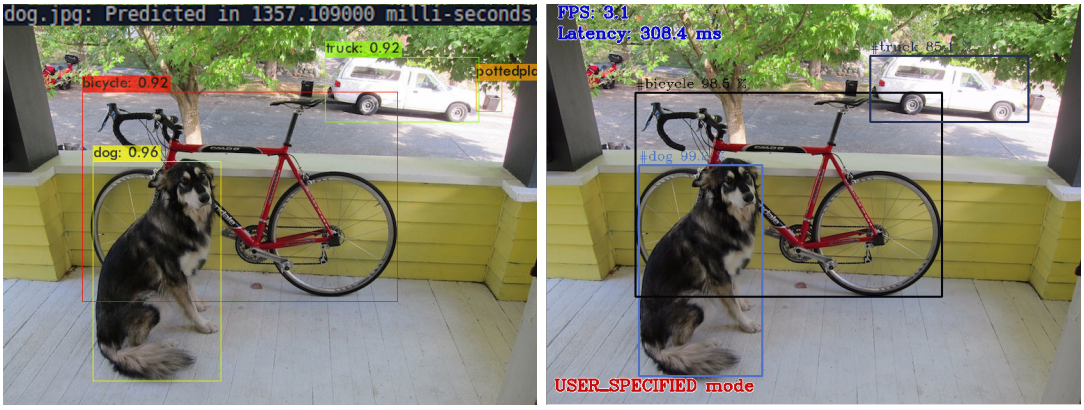
The first approach was to use YoloV3 [10], subsequently updated to YoloV4 [17]. The **C and C++ implementations are based on Darknet, which is a C framework to create and manage neural networks**. Darknet also allows for some hardware optimization, both for CPU and CUDA.
After some work, both networks were discarded due to their poor performances on CPU: **the detection could take up to 1.3 seconds per frame when optimized** for the hardware (AlexeyAB's implementation [24]), 3.5 seconds when not (Pjreddie's original implementation [32]).

5.1.3 OpenVINO

OPEN Visual Inference and Neural network Optimization (OpenVINO) [9] **is a toolkit developed by Intel that allows the optimization of machine learning models** by means of an inference engine that exploits the possibles hardware optimizations of Intel CPUs.
OpenVINO performs different types of optimization on the model, such as:

- hardware optimization exploiting the CPU instruction set;
- fusing consecutive primitive operations in a single convolution, when possible;
- reducing the network size and complexity, preferring a 16bit precision floating point representation instead of the default 32bits one;
- quantizing the model to perform low precision 8bit interference (reducing significantly the accuracy).

Looking at Figure 5.2 it is possible to see that, when running the same network (YoloV4) with the OpenVINO's optimization, it was **found out that the inference time decreased down to almost 300ms per frame**, keeping the same result quality.



(a) AlexeyAB's optimized YoloV4 (b) OpenVINO's YoloV4

Figure 5.1: YoloV4 results

5.1.4 YoloV5

At first, YoloV5 [21] was discarded because of the need to write all the implementation from scratch, but, given the poor performance of the alternatives, at last, it was decided to try it.

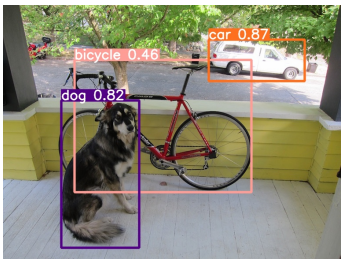
YoloV5 is a **new implementation of the Yolo approach: it leaves Darknet to instead use Pytorch**; however, to optimize for performance, in this project the detection code was written using Libtorch, a Pytorch C++ frontend.

The YoloV5 network family is composed of 5 different networks: nano, small, medium, large and X.

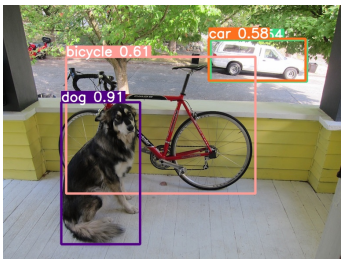
In Table 5.1 and Figure 5.2 is is possible to see he performance of the networks, running on CPU, on a 768 x 576px image.

	nano	small	medium	large	X
inference time	25.6 ms	48.8 ms	100.5 ms	199.9 ms	333.9 ms
network size	3.77 MB	14 MB	40.7 MB	82.9 MB	166 MB

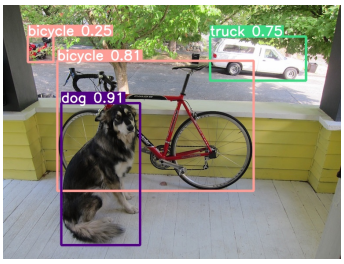
Table 5.1: YoloV5 performances



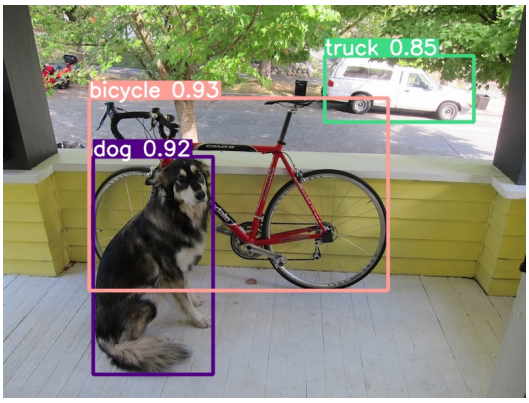
(a) V5 nano



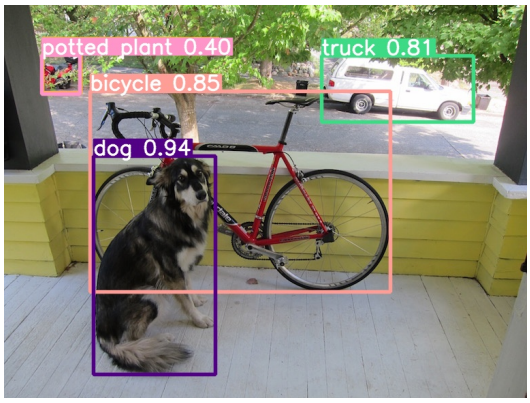
(b) V5 small



(c) V5 medium



(d) V5 large



(e) V5 X

Figure 5.2: YoloV5 results

The YoloV5 family of networks is capable to provide high performances and reasonable accuracy even on CPU, so it was decided to use those networks for the detection task in this project.

Note

This performance tests were performed on a Nvidia Jetson AVX Xavier, running only the detection in the machine.
In real use cases, the detection will run along with all the other containers in the system, so the performance will be poorer.

5.1.5 Other solutions

Detection inside the RSU

There is another solution to this performance issue.
It is possible to upgrade the RSU's embedded board with a GPU; then, the detection could be performed on the RSU itself and sent to the MEC Server afterward.

Pointcloud-based detection

It is possible to perform the detection task directly on the data generated by the LiDAR, without using the camera. In this case detection and volume estimation are merged in a single task. There are different networks that allows that, such as **Voxelnet** [7] and **Complex-YOLO** [11].
However, not only the result are poorer than the one performed on an image, but **the training often is LiDAR-dependant**: different LiDAR models can have different scanning patterns, and a network trained with a specific scanning pattern often does not work with another. This is a problem for this project, since Livox LiDARs often have non-traditional scanning patterns.

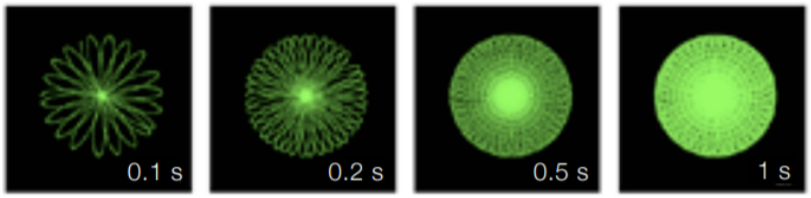


Figure 5.3: Livox Mid-40 scanning pattern [30]

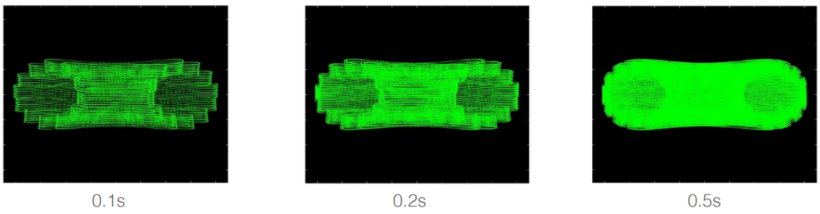


Figure 5.4: Livox Horizon scanning pattern [29]

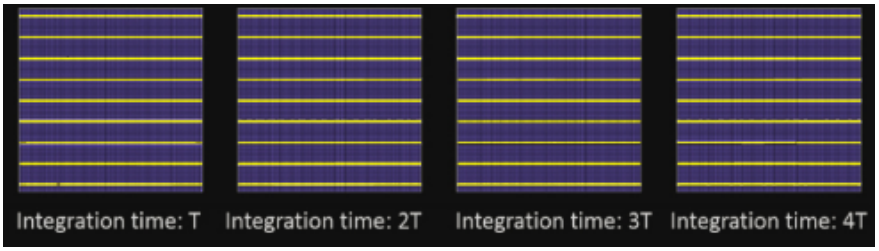


Figure 5.5: Standard LiDAR scanning pattern [30]

5.2 Freespace

The freespaces algorithm, given a pointcloud, needs to be able to compute one (or more) polygon(s) representing the floor area known to be not occupied at that moment. The area of the space not inside a freespace is either occupied by an object or not known (i.e. out of the scope of the LiDAR or behind some object that blocks the LiDAR view).
Since it is known that the freespace must be at floor level and the polygon must be flat, the pointcloud needs to be rigidly transformed in order to have the floor plane mapped in the z=0 plane (Section 3.4).
The algorithm to detect freespaces is a simple ray-tracing algorithm:

- **select a height threshold and an angular resolution** (both automatically read from a configuration file);

- for each point in the pointcloud, if its z coordinate is higher than the threshold, consider it as an obstacle and store its distance; **store the minimum obstacle distance** for each degree;

- for each point in the pointcloud, if its z coordinate is lower than the threshold, consider it as a floor point and store its distance; **store the minimum floor point distance** for each degree;

- **connect all the border points** (minimum floor points and minimum obstacle points) to have the freespace polygon;

- if, for either the obstacle or the floor points, some points in the path are not defined, the algorithm **splits the freespace in more separated polygons**.

In the Figure 5.6 in white are shown the floor points, in red the minimum floor points, and in green the minimum obstacle points, in order to help the understanding of the freespace algorithm.

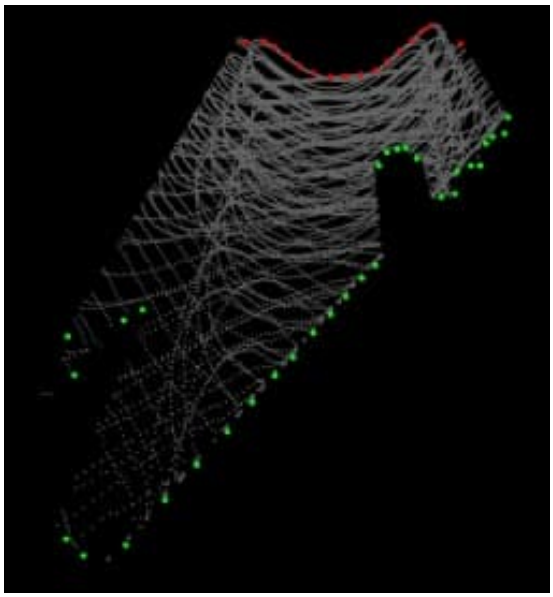


Figure 5.6: Freespace points

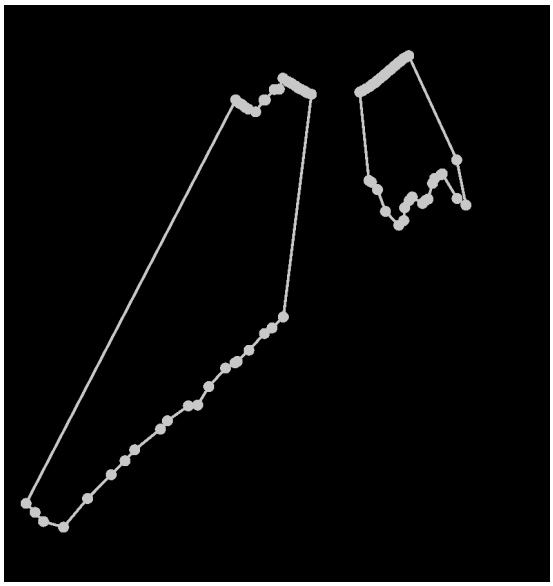


Figure 5.7: Separated freespaces

While in the Figure 4.8 is shown the normal behaviour of the freespace module, in the Figure 5.7 is shown what happens if an obstacle is placed in front of the RSU: in this case the algorithm generates more than one freespace.

5.3 3D Bounding Boxes

The bounding boxes algorithm **takes as input a pointcloud** generated from the LiDAR **and the detection** of all the objects in the image (in JSON format) and **merges those information to compute**, for each of those objects, **the 3D position and volume**; the result is published in JSON format.

For this algorithm to perform well, it is mandatory that there are no errors in the previous containers in the pipeline. Since objects can be far, it is important that the **calibration between camera and LiDAR is accurate**, because even a slight error in the calibration could cause a wrong align of the LiDAR pointcloud and camera image, causing the program to miss the real object position. Moreover, since the objects may move very fast, also the **synchronization between the camera and the LiDAR must be precise**, because using a more recent pointcloud with an older image detection will cause the program to look for an object in the wrong place.

The main algorithm works as follows:

- receive both a detection and a pointcloud;
- remove floor points from the pointcloud;
- for each object in the detection, select the points in the pointcloud that will enter in that region of the image (math in Section 3.2);
- if the point quantity is above a threshold, random sample to reduce the point quantity;
- apply a statistical outlier removal [34] to the points, in order to remove noise or background points;
- use a simple Axis-Aligned Bounding Box (AABB) algorithm to generate the bounding box.

5.3.1 Noise and Statistical Outlier Removal

Even with a perfect LiDAR-camera calibration and synchronization, there are **a lot of factors that can degrade the position** and volume **estimation**. The

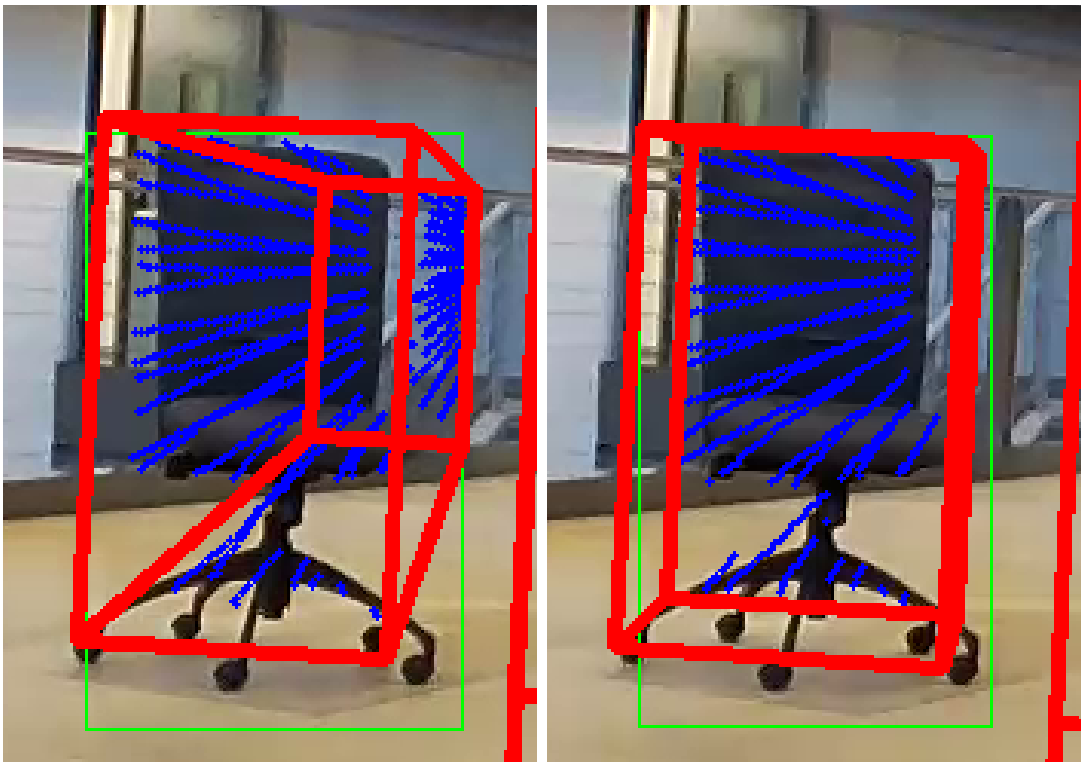
first cause of noise can be the **LiDAR itself**: **some rare faulty points** can be caused by atmospheric dust or an error in the LiDAR behavior. Another, more frequent, cause of errors is a **partial occlusion of the object**: if the object is partially covered by another object, the detection module will still generate a correct rectangle, but the bounding box will have no tool to understand the inside the rectangle there are two different objects, and the interesting one is composed only by a portion of the points in the rectangle. The last cause of errors are the **background points**: if the detection box is bigger than the real object, the program has no way to know that some of the points in the rectangle are not part of the real object, but part of the background.

To solve all of those problems, it has been applied to the interesting portion of the pointcloud a **Statistical Outlier Removal**: this algorithm **removes points from the pointcloud based on their density** [34]. Given a neighborhood size and a threshold, for each point in the pointcloud it is computed the mean distance from its neighbors. Then, the neighbors distant more than $mean\_distance * threshold$ are removed from the pointcloud.
The only drawback of this approach is that the Outlier Removal can be slow if there are a lot of points; to solve this issue, a random sampling is performed on the interesting section of the pointcloud before applying the Outlier Removal.

In the Figure 5.8, are highlighted in green the detection boxes, in red the 3D bounding boxes, and in blue the points used to generate those 3D boxes. It is possible to see that in the first picture (the one that does not use the Statistical Outlier Removal) the program uses some points in the background to generate the chair's 3D box, while in the second one **correctly understand that, even though the green box is bigger, the object does not contain those points**.

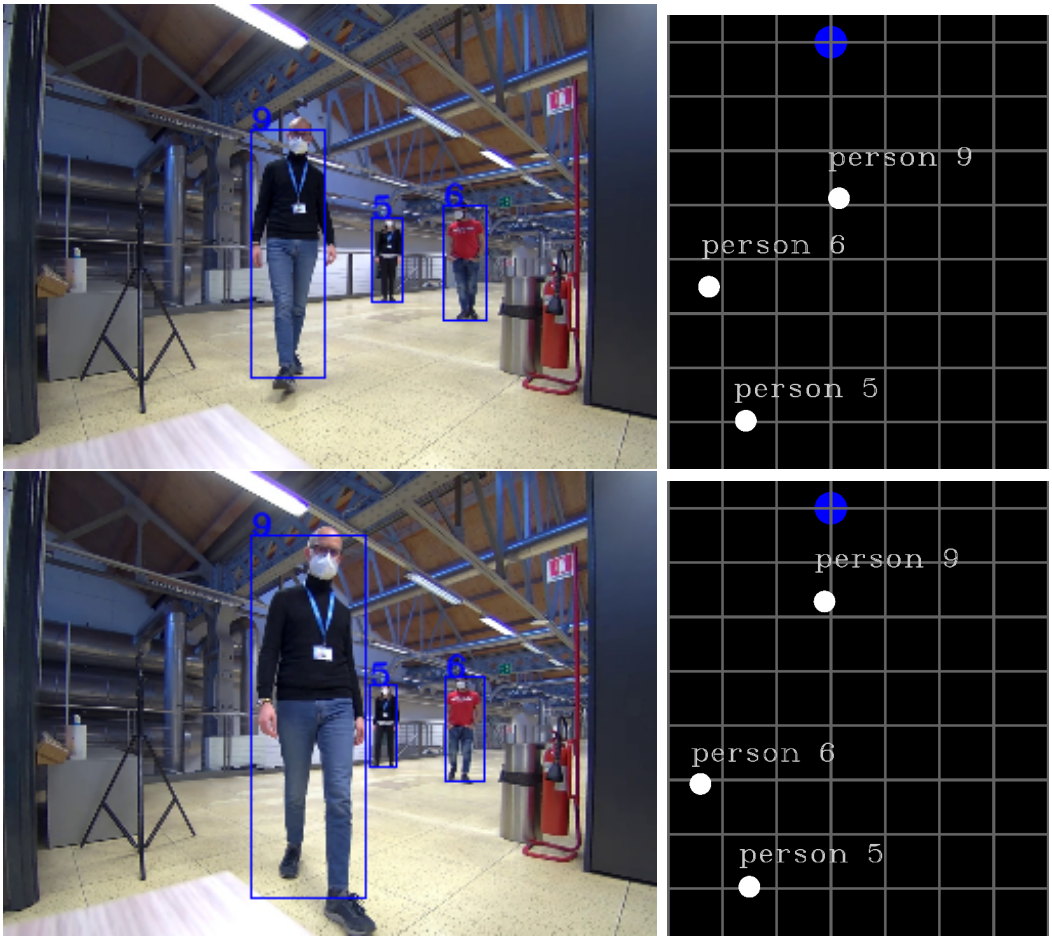
5.3.2 Results

For this project, the main goal of the bounding box is not to estimate the vehicle volumes occupation, but to **track the position of the vehicles in the 3D space** in order to allow the IMA to predict their future trajectory.



(a) Bounding box without outlier removal (b) Bounding box with outlier removal

Figure 5.8: 3D bounding box examples



Frame Position's top view

Figure 5.9: Position estimation

In the Figure 5.9, the first column of images shows the tracking of the actors in the video, while **the second column shows their estimated position on a top view grid**: the grid resolution is one meter, and the blue point at the top represent the LiDAR position.
It is possible to see that **the bounding boxes correctly represent the scene**: position of the actors is correct compared to what can be seen in the real scenario. Furthermore, the program **correctly estimates the movements** between the frames: person 9 moved a meter forward, while person 6 moved slightly backward and person 5 remained in the same place.

5.3.3 Future improvements

The axis-aligned bounding box is easily computed by taking the **minimum and the maximum of each coordinate** axis component.
The main advantage is that this algorithm is easy to implement and really fast at runtime, but **the result may be bigger than the object**, if the object is not parallel to any axis.
For future developments, is recommended to implement an algorithm with the Oriented Bounding Box (OBB), in order to have more accurate results.

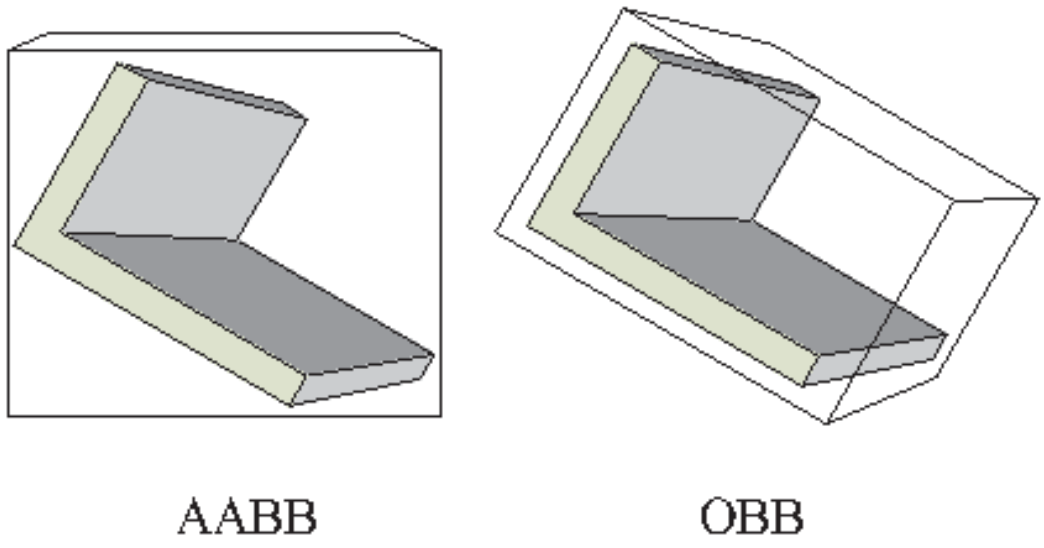


Figure 5.10: Difference between AABB and OBB [27]

Chapter 6

Intersection Movement Assistant

This chapter explains the process that uses the knowledge provided by the data analysis containers to provide the IMA service to the user.
The chapter is structured as follows:

- in Section 6.1 it is explained how the path heatmap is generated, to map each position with a manoeuvre;
- in Section 6.2 it is explained how the path prediction is performed;
- in Section 6.3 are shown the results of the prediction module in two different scenarios;
- in Section 6.4 it is shown how the manoeuvre suggestion works in a simulated scenario.

6.1 Heatmap generator

The heatmap generator is a relatively simple program that **receives as inputs the positions of several cars performing different (labeled) manoeuvres** in an intersection and generates an RGB heatmap.
Each channel of the RGB image represents the path usually taken by a car that wants to perform a specific manoeuvre (turn left, turn right, go ahead). The algorithm works as follows:

- the heatmap starts as a black image;

- for each frame, for each car, depending on the labeled manoeuvre, a circle of color is added to the image. The color depends on the manoeuvre, and the intensity is maximum in the detected position and fades with the distance. To avoid errors in the generation of the heatmap, the algorithm ignores a car if it moved less than 0.2 meters from the last frame.

6.1.1 Heatmap interpretation

For each pixel, the ratio between the channels describes the probability of a car in that position to perform any of the three manoeuvres.
For example, if the value of a pixel is (50, 200, 0) a vehicle in that position has 20% probability of wanting to go left, 80% of going ahead and 0% of turning right.

$$\%_{left} = \frac{100*r}{r+g+b}$$
$$\%_{center} = \frac{100*g}{r+g+b}$$
$$\%_{right} = \frac{100*b}{r+g+b}$$

6.1.2 Parameters

The only parameters for this container are the maximum intensity and the radius of the circle to add to each frame.
These parameters may have different optimal values, depending on the framerate of the positions, the road width and the precision needed.

6.1.3 Other solution

Another possible solution consists in the manual generation of "main paths": it would be possible to create, for each intersection to monitor, the three paths that it is reasonable for a vehicle to follow in order to perform each of the three manoeuvres, and then generate the heatmap based on the distance of each point from the three paths.
Even though this approach is reasonable, it was decided that the data-generated approach was more interesting for this project.

6.2 Manoeuvre Prediction

The goal of the Prediction component is to estimate the probability of any vehicle to perform any manoeuvre at the intersection.
To do so, it first predicts the future position of the vehicle (some seconds ahead) and then uses this prediction and the heatmap associated with that intersection to compute the probabilities.
So, the algorithm is as follows:

- **read the positions** of each vehicle from the 3D bounding box module;
- for each car, use a Linear Kalman filter to **clean possible measurement errors**;
- for each car, use an Unscented Kalman filter (with Constant Turn Rate and Acceleration (CTRA) model) to **predict the position 1 second ahead**. The choice of this model is based on Andrea Mancini's master thesis [20], done previously at LINKS Foundation. From the thesis, models and results have been used, while code has been translated from MATLAB to C++ (to allow integration with other components);
- for each car, **read the heatmap** in the predicted position and compare the intensities in each channel **to estimate manouvres probabilities**.

6.2.1 Kalman filters

A Kalman filter is a **recursive filter that tries to estimate the real state of a dynamic system subjected to noise** [28]. The first version of the Kalman filter works for linear systems, while the **Unscented Kalman filter** can also work with **highly non-linear models**, if the user provides the mathematical model's state function.
Internally, a **Kalman filter stores both a mathematical model and the uncertainties of both the model and the mesurements**; it uses those uncertainties to estimate if a new value received is a realistic one or is noise and, in that, case, tries to correct it with the value generated by the internal model.
Those uncertainties are not easy to tune:

- the one associated with the measurements sometimes can be found in the sensor's datasheet or **computed performing multiple measurements on a known value**;
- the one associated to the mathematical model is vaguer and measures **how much** the user thinks that **the model will estimate correctly the real system behaviour**.

6.2.2 Constant Turn Rate and Acceleration model

The model chosen to estimate the vehicles movement is the **CTRA model**. The CTRA model is a motion model that internally stores **six states: x position, y position, yaw, velocity, acceleration and turn rate**.
Using this model, the Unscented Kalman filter can **use the positions of the vehicle (measurement) to estimate velocity, heading, acceleration and turn rate (internal states)** [20].
Then, this state information will be used to **predict the future of the vehicle using the model's state transition function** (shown in the Figure 6.1), with the assumption of constant acceleration and turn rate; this assumption limits the effectiveness of the prediction to only a few seconds, since in a general case it is not true that acceleration and turn rate are constant.

6.2.3 Other solutions

There are a lot of alternatives solutions, divided in mainly two different approaches:

- keep the Unscented Kalman filter, changing the vehicle mathematical model;
- remove the Unscented Kalman filter, and use another algorithm to predict the future positions of the vehicle.

Change mathematical model

It has been shown [20] that **the CTRA model has some drawbacks**: it cannot be used to predict the long-term future since its assumptions can only be true in a short-term analysis, and it is only physics-based, so it does not have

$$\vec{x}(t+T) = \begin{pmatrix} x(t+T) \\ y(t+T) \\ \theta(t+T) \\ v(t+T) \\ a \\ w \end{pmatrix} = \vec{x}(t) + \begin{pmatrix} \Delta x(T) \\ \Delta y(T) \\ wT \\ aT \\ 0 \\ 0 \end{pmatrix}$$

where:

$$\begin{aligned} \Delta x(T) =& \frac{1}{w^2}[(v(t)w + awT)sin(\theta(t) + wT) + acos(\theta(t) + wT) \\ & -v(t)wsin\theta(t) - acos\theta(t)] \end{aligned}$$

and

$$\begin{aligned} \Delta y(T) =& \frac{1}{w^2}[(-v(t)w - awT)cos(\theta(t) + wT) + asin(\theta(t) + wT) \\ & +v(t)wcos\theta(t) - asin\theta(t)] \end{aligned}$$

Figure 6.1: State function of the CTRA model [20]

any knowledge about road manouvres.
Despite its faults, however, in this project it proved to be a useful tool for a short-term prediction.
Changing the mathematical model is a matter of defining the states in which the filter is interested in and how they update (state function).

Long Short-Term Memory

An Long Short-Term Memory (LSTM) is a special type of Recurrent Neural Network (RNN). An RNN is a neural network that, differently from a normal one, stores a state internally, and its outputs depend both on the inputs and the stored state. The main application of RNNs involves speech recognition, where its ability to remember the past words and context is fundamental [5].
Recently, it was studied the **use of LSTMs to predict vehicle motions** [18]; this approach could create a **manouvre-aware model** instead of a physics-based one, and so have a longer span of accuracy.

However, this solution requires running a neural network for each detected object: **this could be very expensive from a computational point of view**, and should be tested if it will still make possible for the whole system to run at an acceptable speed.

6.3 Application and tests

Due to physical installation constraints, **the LiDAR and camera setup did not get on the field in time for this thesis to be published**, so the last part of this project (the one concerning the prediction of vehicle movement) could not be tested in conjunction with the first part of data processing.
To test the prediction module, two different approaches have been taken:

- test on an **RSU** already installed, using only a camera;
- test on **simulated data**.

6.3.1 Test on RSU

LINKS Foundation installed an RSU with an embedded board and a camera, **mounted right after a turn and before a "T" intersection**.
Even though the intersection is not clearly visible from the camera point of view, the path of the vehicles allows us to understand if they are going to turn left or not.
The RSU does not contain a LiDAR, but the tracking implementation also provides a feature that tries to associate each detected object with its geographic coordinates. Then, knowing the geographic coordinates of the RSU, is possible to estimate the distance between the RSU and the vehicle. **This measure may not be as accurate as the one generated by a LiDAR**, but associated with a Kalman filter to remove noises, was good enough to allow the prediction module to work.

(a) (b)

Figure 6.2: RSU intersection map and heatmap

Figure 6.3: RSU point of view

The Figure 6.4 shows the predictions of some cars in the intersection: the white points show the trajectory prediction in the next 1 second, while the percentages near the vehicle show the manoeuvre prediction computed using the trajectory prediction and the heatmap. It is easy to see that, in a real use case, vehicles can follow very different paths: **a vehicle trying to turn will take a wider turn and position itself in the left section of the road**, in order to perform a correct manoeuvre. This different approach allows the algorithm to understand what the driver wants to do various seconds before the turn occurs.

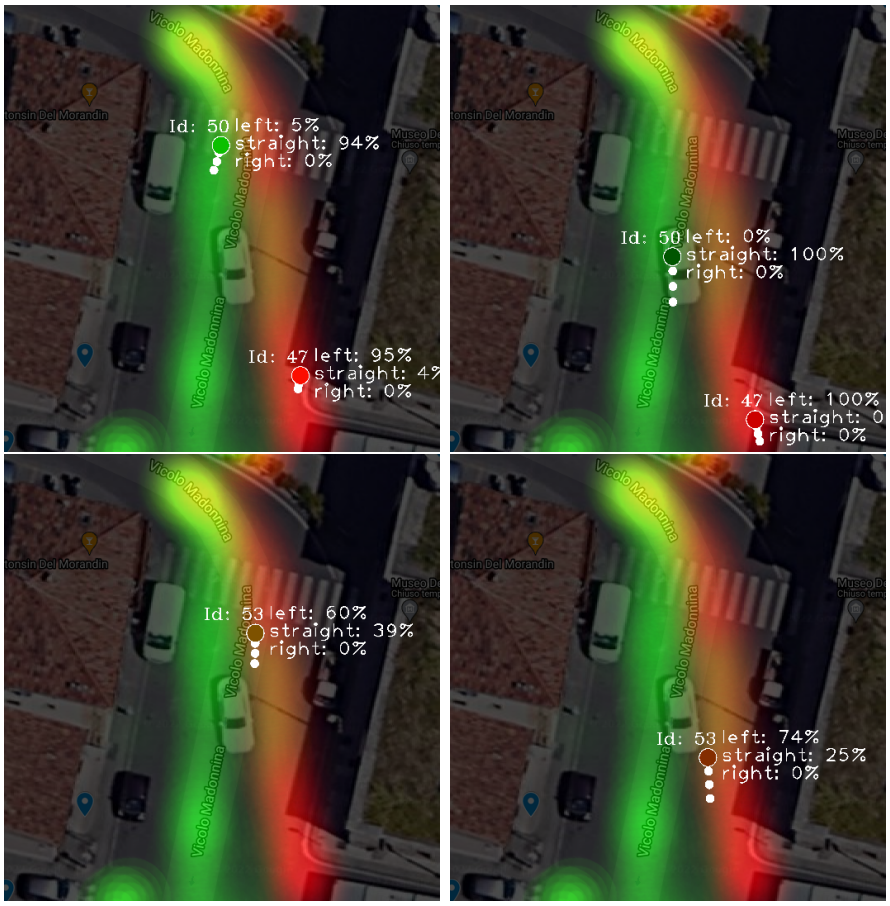
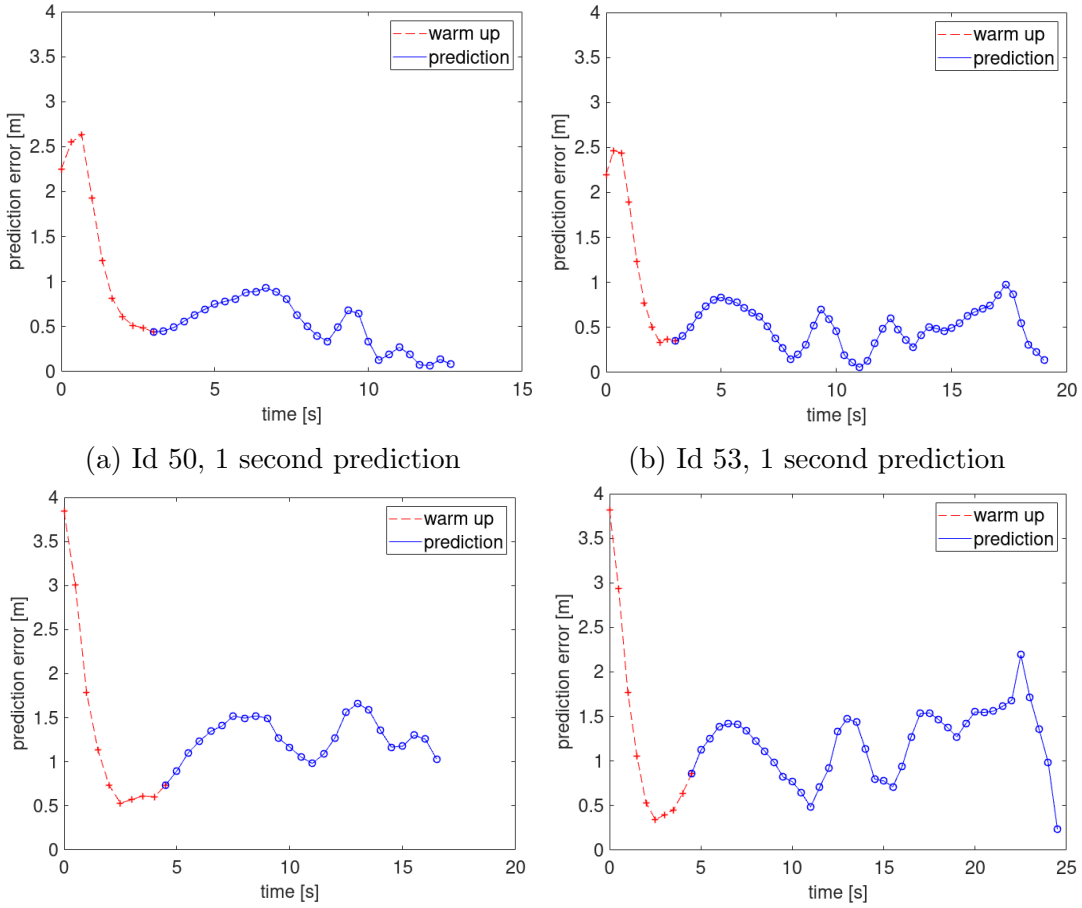


Figure 6.4: Prediction results

In the Figure 6.5, in (a) and (b) are shown the prediction errors over time of the vehicles in the Figure 6.4. The first red section is characterized by an high error, this is because the Unscented Kalman filter needs a few runs in order to correctly align himself with the real value. Then, it is possible to see that the error in **the prediction error stabilizes below one meter**; this is an acceptable result, since our goal is not to predict with absolute accuracy the trajectory of the vehicle, but to predict its intersection manoeuvre. In the Figure 6.5 in (c) and (d) are shown the errors of the same vehicles when trying to predict their positions two second in the future; as expected, it is possible to see that the error is bigger, with peaks ad 1.5 and 2 meters.
More information about the accuracy of the prediction performed by the Unscented Kalman filter are available in the original thesis [20].



(a) Id 50, 1 second prediction

(b) Id 53, 1 second prediction

(c) Id 50, 2 seconds prediction

(d) Id 53, 2 seconds prediction

Figure 6.5: Prediction errors

6.3.2 Test on simulated data

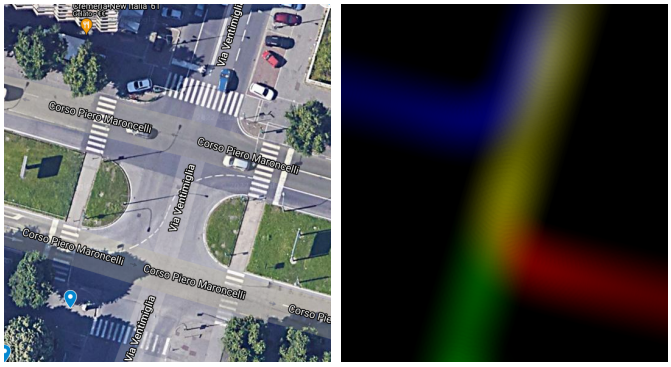
To perform tests in a **more complex scenario**, it was decided to use some artificially generated data. Artificial data also allow to test the prediction with **more reliable positional information** than the one generated by a camera.

SUMO and TRaCI

Simulation of Urban MObility (SUMO) is an open-source traffic simulator designed to handle large networks [8]. It allows the user to **generate its own road network** or to import an existing one from different file formats; for this project, a map of Turin was exported from OpenStreetMap (OSM) and then converted into a SUMO map using the netconvert tool [31].

SUMO also allows to **define any number of actors**, each with its own starting

point, destination, maximum velocity, allowed path and more parameters. After both a network and a list of actors are defined, SUMO will **start a real-time simulation of the behavior of those actors**, taking care also of eventual mutual interaction between actors, and of the interaction between actors and infrastructures (for example, traffic lights).
Traffic Control Interface (TraCI) is an **interface that allows SUMO to act as a server**, waiting for commands through a TCP connection from any client. To automatize the simulations, **TraCI is available as a library** for multiple programming languages; for this project, the python interface [35] was used in order to generate simulated data of vehicles moving in a Turin intersection.



(a) (b)

Figure 6.6: Turin intersection map and heatmap

Data generation

Using a simple python program, it was possible to **generate JSON messages with the same structure as the ones generated by the real RSU**, and use them to test the prediction module.
However, after a few tests, it was clear that the generated data was unrealistically clear. **To reproduce working conditions similar to the ones of the real RSU, some white noise was added** to all measurements: a random value between -0.5 and 0.5 meters was added both to the x and y coordinates of every single measurement, creating an uncertainty of maximum 0.71 meters on the vehicle positions.
The heatmap, instead, was generated from data without noise, to test an approach more similar to the one proposed in Section 6.1.3.

Results

Figure 6.7: Prediction results

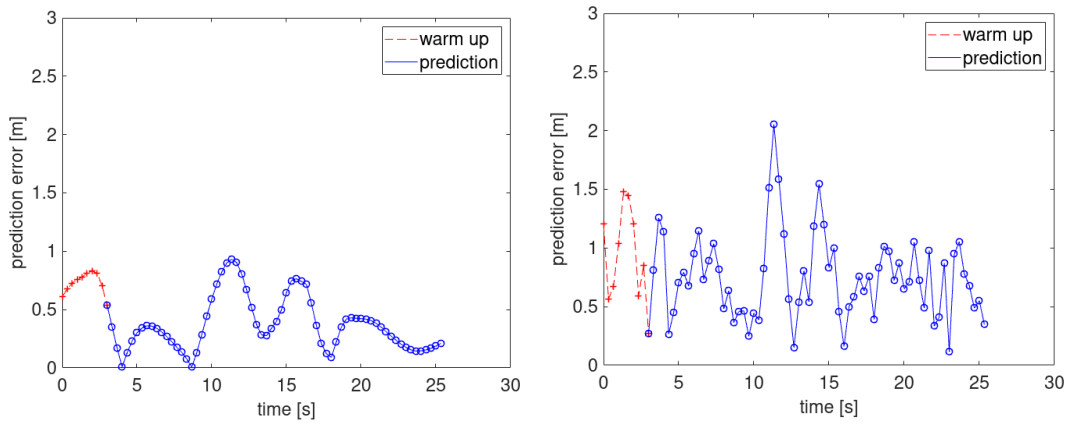
In the Figure 6.7 it is possible to see that, even with the added noise, the **Kalman filter correctly predicts the path of the three simulates vehicles**:

- The **vehicle with ID 0** moves in the right lane of the road, this increases from its probability to turn right way before it reaches the intersection; then, it starts turning, and this manoeuvre is correctly detected.

- The **vehicle with ID 1** does not turn; this is detected in the center of the intersection because the prediction estimates that the vehicle will go ahead too far away to allow a turning manoeuvre.

- The **vehicle with ID 2**, instead, turns left; this is detected because, in the center of the intersection, it starts to slow down a turn.
  Both for this vehicle and the previous one, it's difficult to predict if they are

going to go straight or turn right before a certain point of the intersection is reached.



(a) ID: 2, 1 second prediction (b) ID: 2, 1 second prediction, added noise

Figure 6.8: Prediction errors

In the Figure 6.8 it is shown the prediction error of the vehicle with ID 2 from the Figure 6.7. The first image shows the error without the added noise, using only the SUMO simulation, while the second also include the random noise. It is possible to see that the first red section the error has a peak, due to the warm up of the Kalman filter. Then, there are other two peaks in both the images: **the second peak is caused by the deceleration** of the vehicle, while the **third by the beginning of the turning manoeuvre**; since the CTRA model assumes constant acceleration and turning rates, both those sudden changes cause a peak in the prediction error.

It is worth noticing that this **simulation does not consider some parameters** that could be important in a real scenario; for instance, in this test the 3 vehicle moved freely, since **there was no traffic** on the road: it is possible that in a congested road vehicles will move slower and follow different paths in order to avoid hitting other vehicles.

6.4 Manoeuvre suggestion

The goal of this component is to generate and provide manoeuvre suggestion for a vehicle, based on the the vehicle's position and the prediction of the other

vehicles.
The module works knowing the topology of the intersection and the relative position of the vehicles, and using those information to find which manoeuvres can cause a dangerous situation, in a similar way to what is shown in the Figure 1.1, 1.2 and 1.3.

6.4.1 Vehicle connection

For the same reasons of the other module (the Prediction one), it was not possible to test this program in a real scenario, but only on a simulated one.
In a real scenario, this module will need to communicate to a connected vehicle, in order to send the manoeuvre suggestion. This communication has not been implemented yet, but there are different solutions to implement it:

- the vehicle communicates its position sending a Cooperative Awareness Messages (CAM) [15] to the MEC server; the MEC server sends the manoeuvre suggestion using a non standard message;
- the vehicle communicates its position sending a CAM to the MEC server; the MEC server sends the manoeuvre suggestion using a Maneuver Coordination Message (MCS) [14];
- expose the service as a web server, the vehicles sends its position in a GET request and receives the manoeuvres suggestion as a response.

6.4.2 Results

In the Figure 6.9 it is shown how the manoeuvre suggestion module works in a simulated scenario:

- in (a) the connected vehicle can only turn left, since turning right or going straight could cause a crash with the vehicle with **ID 1**;
- in (b) the connected vehicle can turn both left and right, but cannot go straight since it could cause a crash with the vehicle with **ID 2**; the program does not consider the velocities of the vehicle, so it considers going straight not safe even if in this case it could be.

(a) (b)

Figure 6.9: Manoeuvre suggestion results

6.4.3 Future improvements

The module has to be enhanced in order to receive the position from a connected vehicle and respond with a manoeuvre suggestion.
Furthermore, the current implementation follows a basic approach and does not consider the velocity of the vehicles when detecting dangerous manoeuvres; this could be added in future updates.

Chapter 7

Conclusion

Summarizing, in this thesis it has been developed a **complete system that can provide the vehicle path prediction and use it to generate manoeuvre suggestions**. The hardware infrastructure is composed of an RSU (mounting a LiDAR, a camera and an embedded board) and a MEC server, that performs the computation.

To ensure the coherence of the different sensors, it was developed a **procedure that allows the extrinsic calibration** of camera and LiDAR in any scenario; this procedure requires the selection of some keypoints both on an image and on a pointcloud, and computes the rigid transformation that makes the keypoints of both sensors coincide.
The software system starts with some containers that **interface to the hardware using some low-level SDK** and then publishes the retrieved data to an AMQP broker in a more simple format.
Then, using the camera stream, it is performed an **object detection** to identify all the road actors. A lot of effort was devoted in the **optimization** of this task, since it will run on a CPU instead that a GPU.
Another task performed by the system is the **3D bounding box estimation around any detected object**; this is performed by merging the information received by the LiDAR to the detection performed on the video. This merge is possible only thanks to the accuracy of the calibration procedure.
Lastly, the **real time positions of the road actors are used to predict their future path** using an Unscented Kalman filter, and a data-driven approach allows to **map the prediction to one of the three possible manoeuvres**

available at the intersection (left turn, right turn, go straight).

Once that is possible to predict the future path of any vehicle at the intersection, it is possible to **send alerts or manoeuvre suggestions to the vehicle**s that are able to connect to the system, in order to avoid conflict between the vehicles trajectories and **increase the road safety**.

The **high modularity** of the project allows the different components to be replaced with future updates or reused in other projects, provided that the communication interface remains constant. Furthermore, this software architecture is suitable for different hardware configurations, since there is no restriction on the physical allocations of the running containers.

7.1 Future improvements

This project shows the feasibility of the manoeuvring assistance in an intersection based on the information of a camera and a LiDAR, but there are a lot of improvements that can still be performed.

Since the LiDAR and camera setup was not mounted in the intersection before the publication of this thesis, **it was not possible to test the whole pipeline at once**: the data analysis containers were tested on real data while the IMA module was tested on generated data. So, the first improvement is to test the whole system together, once the complete RSU is available, developing also the connection with the vehicle.
Furthermore, **it has not yet been a performance test on the final hardware**, to measure some KPIs such as processing time and communication latencies.

Once those tests are performed, **any of the single modules can be updated** to have better performance or accuracy, for example:

- it could be studied if an LSTM-based approach for the prediction could provide better long term results without degrading too much the performance

(as explained in Section 6.2.3);

- it could be developed a more precise approach on the creation of the bounding boxes, that now overestimate a little the spacial occupation (as explained in Section 5.3.3);
- update the Detection and the Tracking module if new solutions with better performances are found.

References

[1] Zhengyou Zhang. "A Flexible New Technique for Camera Calibration". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22 (Dec. 2000). MSR-TR-98-71, Updated March 25, 1999, pp. 1330–1334. URL: https://www.microsoft.com/en-us/research/publication/a-flexible-new-technique-for-camera-calibration/.

[2] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2003.

[3] David L. Mills. "The NTP Era and Era Numbering" (2012).

[4] José Santa et al. "Experimental evaluation of CAM and DENM messaging services in vehicular communications". *Transportation Research Part C: Emerging Technologies* 46 (Sept. 2014), pp. 98–120. DOI: 10.1016/j.trc.2014.05.006.

[5] Xiangang Li and Xihong Wu. *Constructing Long Short-Term Memory based Deep Recurrent Neural Networks for Large Vocabulary Speech Recognition*. 2015. arXiv: 1410.4281 [cs.CL].

[6] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].

[7] Yin Zhou and Oncel Tuzel. *VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection*. 2017. arXiv: 1711.06396 [cs.CV].

[8] Pablo Alvarez Lopez et al. "Microscopic Traffic Simulation using SUMO". In: *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, Nov. 2018, pp. 2575–2582. URL: https://elib.dlr.de/127994/.

[9] *OpenVINO toolkit*. 2018. URL: https://docs.openvino.ai/latest/index.html.

[10] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement*. 2018. arXiv: 1804.02767 [cs.CV].

[11] Martin Simon et al. *Complex-YOLO: Real-time 3D Object Detection on Point Clouds*. 2018. arXiv: 1803.06199 [cs.CV].

[12] Petru Soviany and Radu Tudor Ionescu. "Optimizing the Trade-off between Single-Stage and Two-Stage Object Detectors using Image Difficulty Prediction". *CoRR* abs/1803.08707 (2018). arXiv: 1803.08707. URL: http://arxiv.org/abs/1803.08707.

[13] 5GAA. *C-V2x Use Cases Methodology, Example and Service Level Requirements*. 2019.

[14] Alejandro Correa et al. "Infrastructure Support for Cooperative Maneuvers in Connected and Automated Driving". In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. 2019, pp. 20–25. DOI: 10.1109/IVS.2019.8814044.

[15] *ETSI EN 302 637-2 V1.4.1*. Tech. rep. 2019. URL: https://www.etsi.org/deliver/etsi_en/302600_302699/30263702/01.04.01_60/en_30263702v010401p.pdf.

[16] Francesca Pacella. "Multiple Object Tracking and trajectory prediction for safety enhancement of autonomous driving". PhD thesis. Politecnico di Torino, 2019.

[17] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934 [cs.CV].

[18] André Ip, Luis Irio, and Rodolfo Oliveira. "Vehicle Trajectory Prediction based on LSTM Recurrent Neural Networks". In: *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*. 2021, pp. 1–5. DOI: 10.1109/VTC2021-Spring51267.2021.9449038.

[19] Seolyoung Lee, Cheol Oh, and Gunwoo Lee. "Impact of Automated Truck Platooning on the Performance of Freeway Mixed Traffic Flow". *Journal of Advanced Transportation* 2021 (2021).

[20] Andrea Mancini. "Vehicle path prediction for safety enhancement of autonomous driving". PhD thesis. Politecnico di Torino, 2021.

[21] Xingkui Zhu et al. *TPH-YOLOv5: Improved YOLOv5 Based on Transformer Prediction Head for Object Detection on Drone-captured Scenarios.* 2021. arXiv: 2108.11539 [cs.CV].

[22] 5GAA. *MEC for Automotive in Multi-Operator Scenarios.* Tech. rep. URL: https://5gaa.org/wp-content/uploads/2021/03/5GAA_A-200150_MEC4AUTO_Task2_TR_MEC-for-Automotive-in-Multi-Operator-Scenarios.pdf.

[23] Sameer Agarwal, Keir Mierle, et al. *Ceres Solver.* http://ceres-solver.org.

[24] AlexeyAB. *Darknet.* https://github.com/AlexeyAB/darknet.

[25] "Calibrate fisheye lens using OpenCV" (). URL: https://medium.com/@kennethjiang/calibrate-fisheye-lens-using-opencv-333b05afa0b0.

[26] "Edge computing" (). URL: https://en.wikipedia.org/wiki/Edge_computing.

[27] "How Collision Detection works" (). URL: https://devdept.zendesk.com/hc/en-us/articles/360011559320-How-Collision-Detection-works-v2020-.

[28] "Kalman filter" (). URL: https://en.wikipedia.org/wiki/Kalman_filter.

[29] "Livox Horizon datasheet" (). URL: https://www.livoxtech.com/horizon/downloads.

[30] "Livox Mid-40 datasheet" (). URL: https://www.livoxtech.com/mid-40-and-mid-100/downloads.

[31] "Netconvert documentation" (). URL: https://sumo.dlr.de/docs/netconvert.html.

[32] Pjreddie. *Darknet.* https://github.com/pjreddie/darknet.

[33] SAE. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles.*

[34] "Statistical outlier removal" (). URL: https://pcl.readthedocs.io/projects/tutorials/en/latest/statistical_outlier.html.

[35] "TraCI python documentation" (). URL: https://sumo.dlr.de/docs/TraCI/Interfacing_TraCI_from_Python.html.

Ringraziamenti

A conclusione di questo elaborato, desidero menzionare tutte le persone, senza le quali questo lavoro di tesi non esisterebbe nemmeno.
Ringrazio i colleghi della Fondazione LINKS in cui ho svolto questo interessante lavoro di tesi. In particolare, ringrazio Daniele Brevi e Edoardo Bonetto per i preziosi consigli e suggerimenti.
Ringrazio i miei genitori, perché senza di loro non avrei mai potuto intraprendere questo percorso di studi.
Ringrazio Silvia, la mia ragazza, per essermi stata vicino tutto questo tempo.
Infine ringrazio tutti gli amici, troppi per elencarli qui, che mi hanno accompagnato in questo percorso.