Study and Adaptation of the Autonomous Driving Simulator CARLA for the ATLASCAR2

Pedro Silva 72645

Supervisor: Prof. Dr. Paulo Dias

Co-Supervisor: Prof. Dr. Vitor Santos

MIECT

Deti/Dem/Ieeta - UA

19/07/2019

Outline

- Scope.
- Problems and Motivation.
- Objectives.
- Background concepts and tools.
- Software development.
- Experiments.
- Main contributions.
- Future work.

Scope

- Integration of the CARLA autonomous driving simulator within the ATLASCAR2 project.

- Detection
- Tracking
- Labelling

} LIDAR Sensors + PointGrey Cameras

Problems and Motivation.

- The task of acquiring data with the ATLASCAR is too long.
- The environment cannot be controlled.
- The algorithms can only be tested in the vehicle.
- Simulation is required.

Proposed Solution

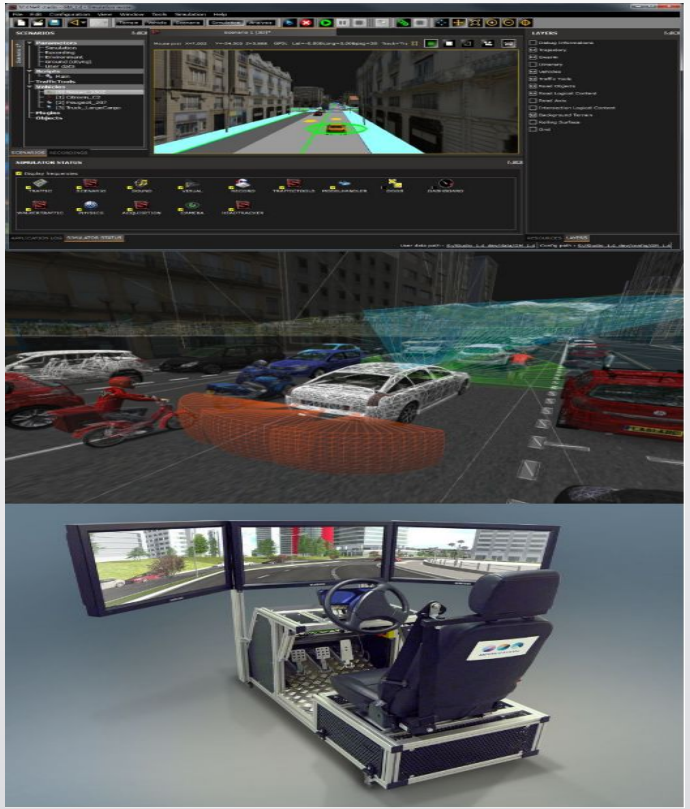
- An autonomous driving simulator can solve these issues since it is possible to control aspects of the environment and manipulate the agents involved in the autonomous driving process.
- With this simulator it is possible to replicate the sensor setup of the vehicle and test out the algorithms with the data from those sensors before implementing them in the real platform.
- CARLA was chosen as a real testbed to be used in the context of the ATLASCAR2 project.

Objectives

- Replicate the sensor setup of the ATLASCAR2 vehicle.
- Test out the results and the viability of the simulator and its controlled environments.
- Build tools and update tools for CARLA-ROS integration.
- Implement algorithms with CARLA simulated data and check if they can be used to validate the results of other algorithms currently being used on the ATLASCAR2.

Autonomous Driving Simulators

- CARLA
- COGNATA
- NVIDIA Drive
- SimDriver
- SCANeR™



Simulator Choice

- It is the simulator's job to produce scenarios that are the closest thing to the real deal.
- It is important to reflect on the cost problems and the software tools provided by the simulator and see if they fit in the context of the project.

- **CARLA** provides:
  - **Realistic Scenarios.**
  - **Client API.**
  - **ROS Bridge.**

Base Software Tools

- **ROS (Melodic).**
- **OpenCV Library.**
- **Point Cloud Library(PCL).**
- **Pygame Library.**
- **CARLA (0.9.5).**

- All of the **software tools** were **developed** in **Python** and **C++**

CARLA Project

- CARLA (Car Learning How to Act) is an open-source simulator used for autonomous driving research and it was developed by a team of researchers, programmers and artists in the Computer Vision Center in Barcelona.
- This simulator has two main **modules**:
  - CARLA Simulator Engine.
  - CARLA Python API.
- This simulator has a variety of different **sensors** which includes:
  - Camera sensors.
  - Range-based sensors.

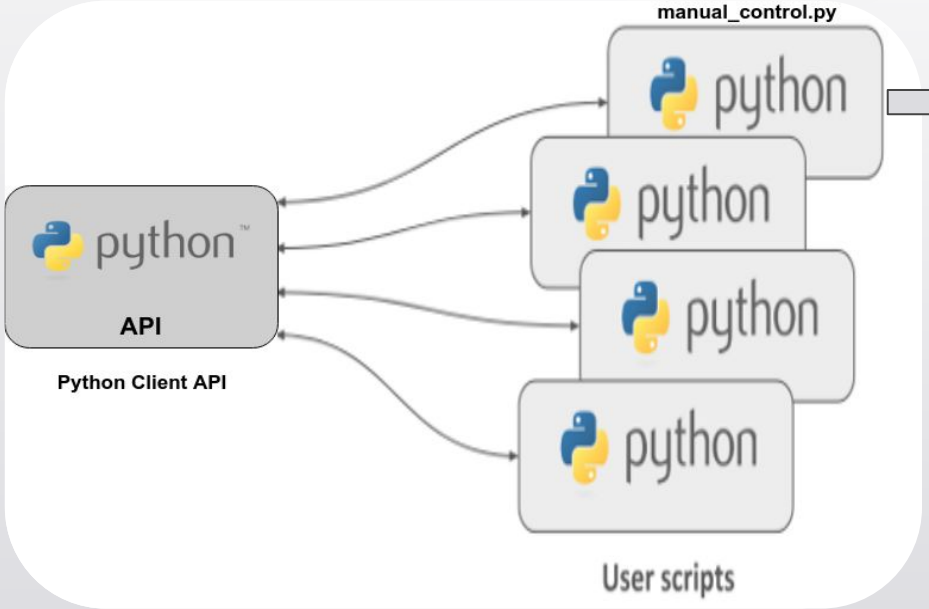
CARLA - Simulation Engine

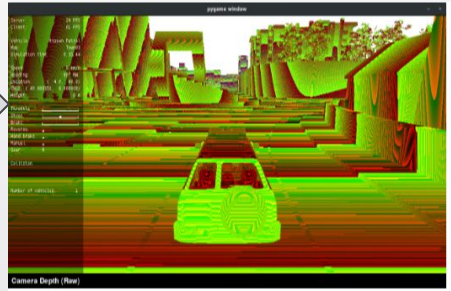
- Made in C++ and Unreal Engine 4.
- State-of-the-art rendering, realistic physics, NPC components and interoperable plugins.
- Server-client model (host:127.0.0.1).
- Establishes connection with the client and renders the scene according to certain parameters.

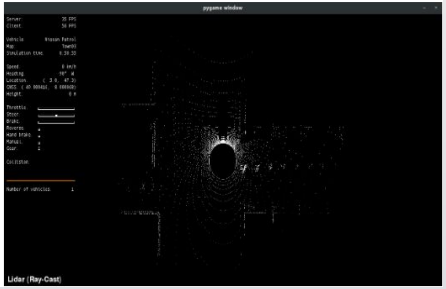

```
./CarlaUE4.sh -windowed -ResX=<image_size_x> -ResY=<image_size_y> -benchmark -fps=<number_of_fps>

e.g: ./CarlaUE4.sh -windowed -ResX=320 -ResY=240 -benchmark -fps=10
```

CARLA - Python API

- Connects to simulator via server port (e.g:2000).
- Interface used to control the simulator and retrieve data.

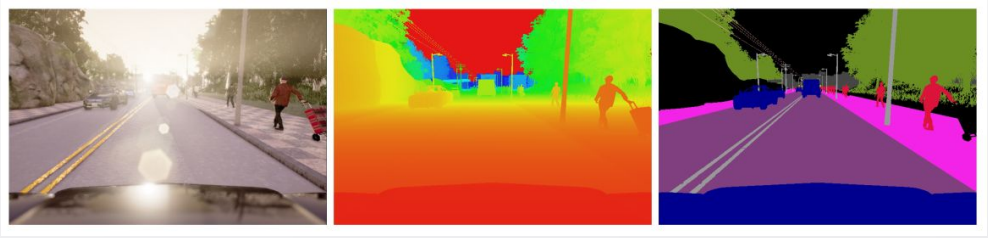


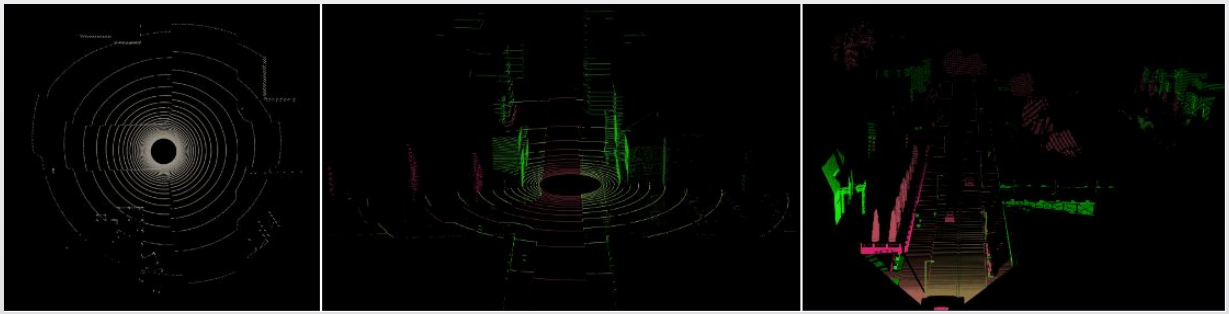




CARLA - Simulation Sensors

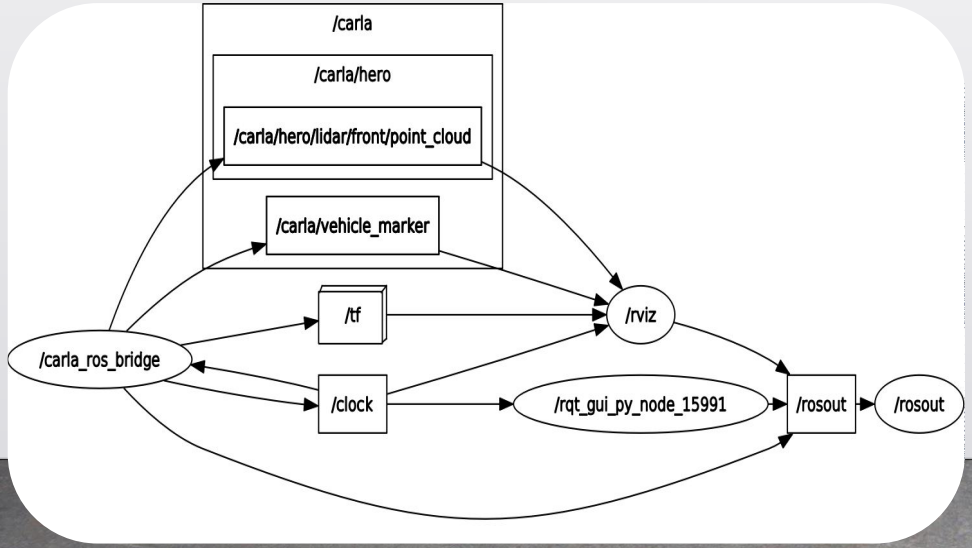
- CARLA allows for flexible configuration of the agent's sensor suite.
- This sensor suite includes:
  - 3 camera models (RGB, Depth, Segmentation).
  - Ray-cast LIDAR.
  - Odometry.
  - GPS.
  - Lane Invasion.
  - Collision Event.

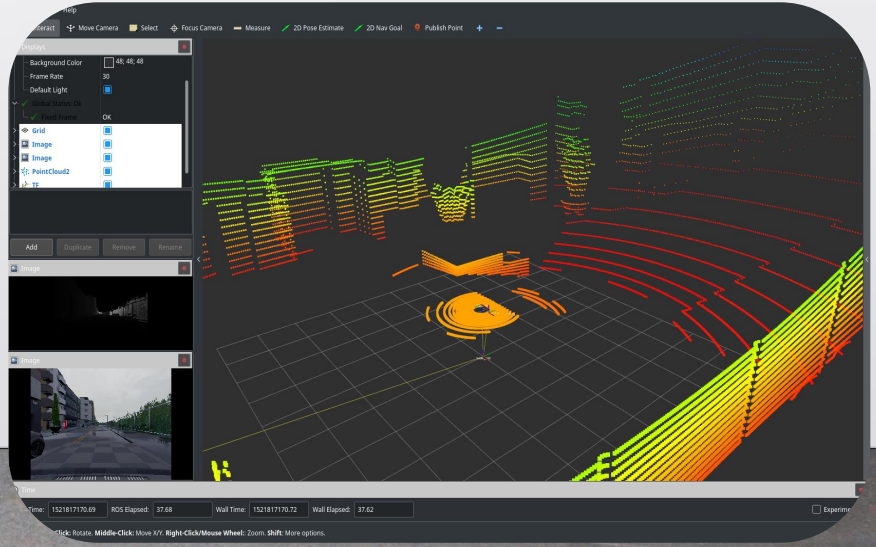




CARLA - ROS Bridge

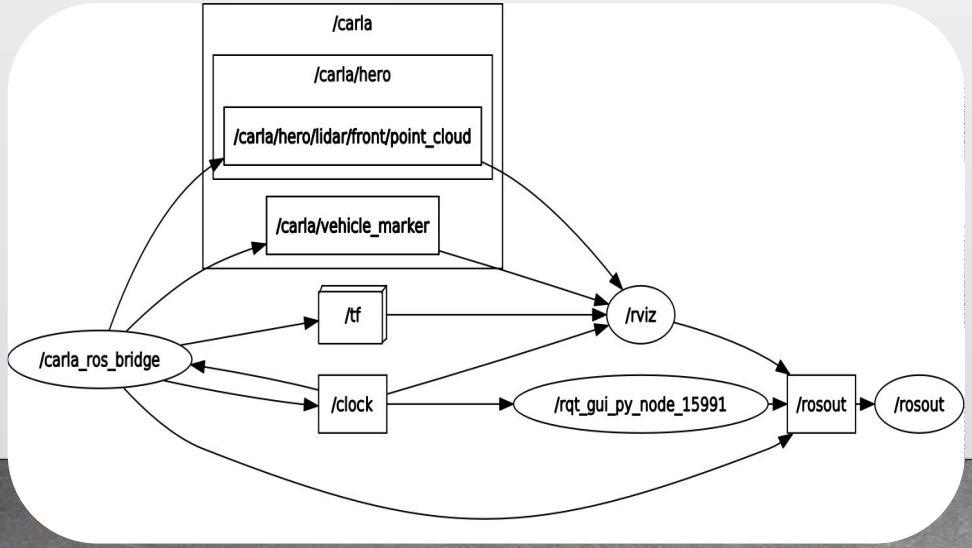
- The **CARLA** and **ROS** integration is achieved via the **carla_ros_bridge** package.
- This package serves as an **interface** that connects **CARLA** to the **ROS framework** showing the results present in the CARLA world with **RVIZ**.

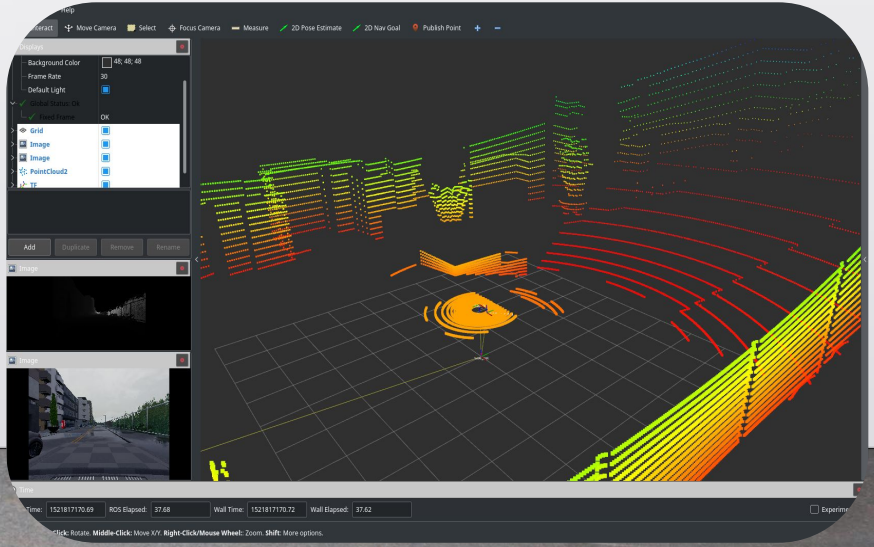




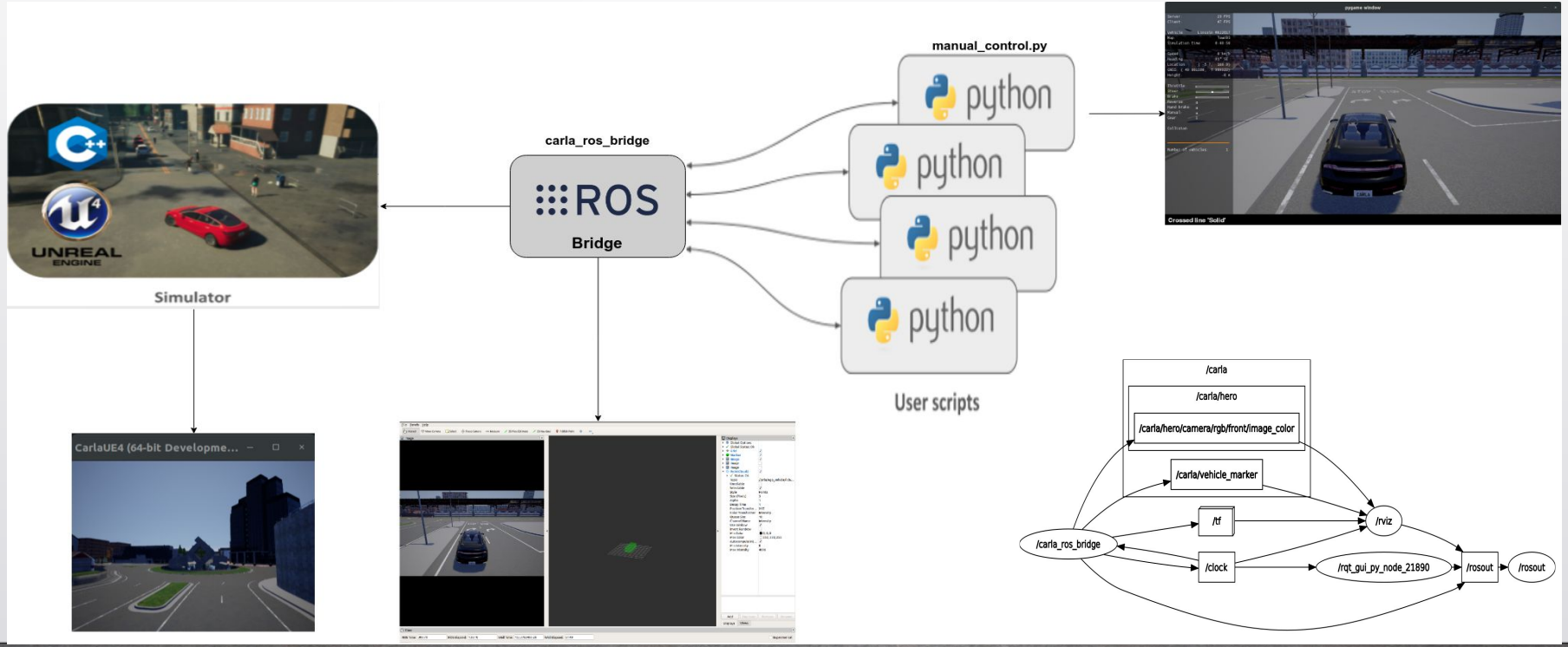
CARLA - ROS Bridge

- The **CARLA** and **ROS** integration is achieved via the **carla_ros_bridge** package.
- This package serves as an **interface** that connects **CARLA** to the **ROS framework** showing the results present in the CARLA world with **RVIZ**.



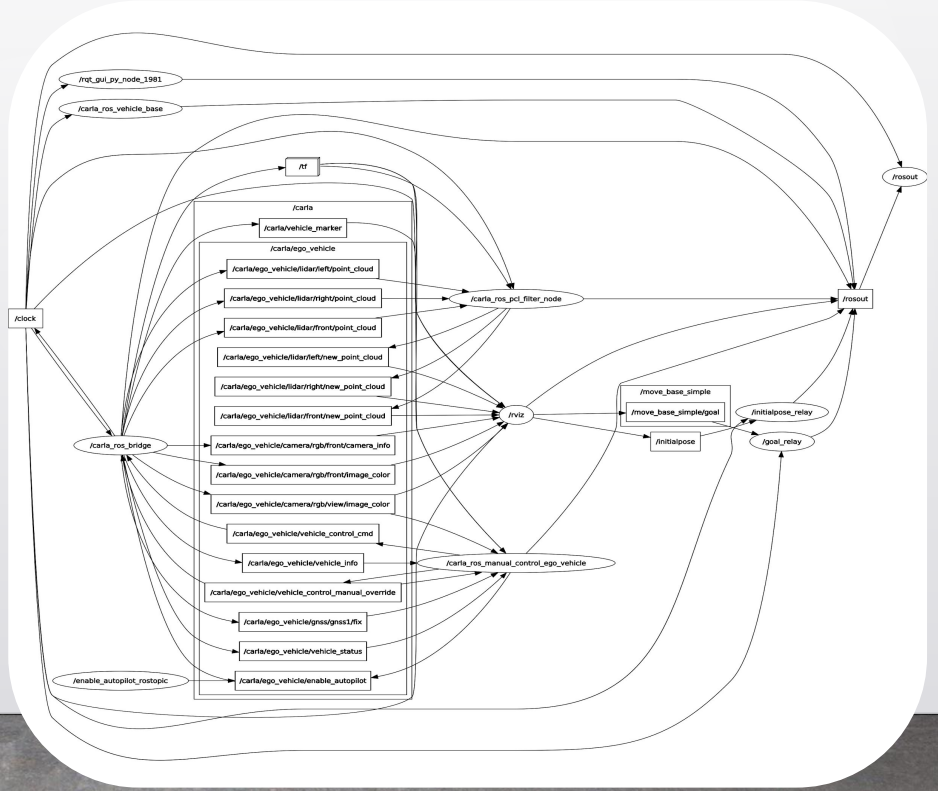


CARLA - ROS Bridge



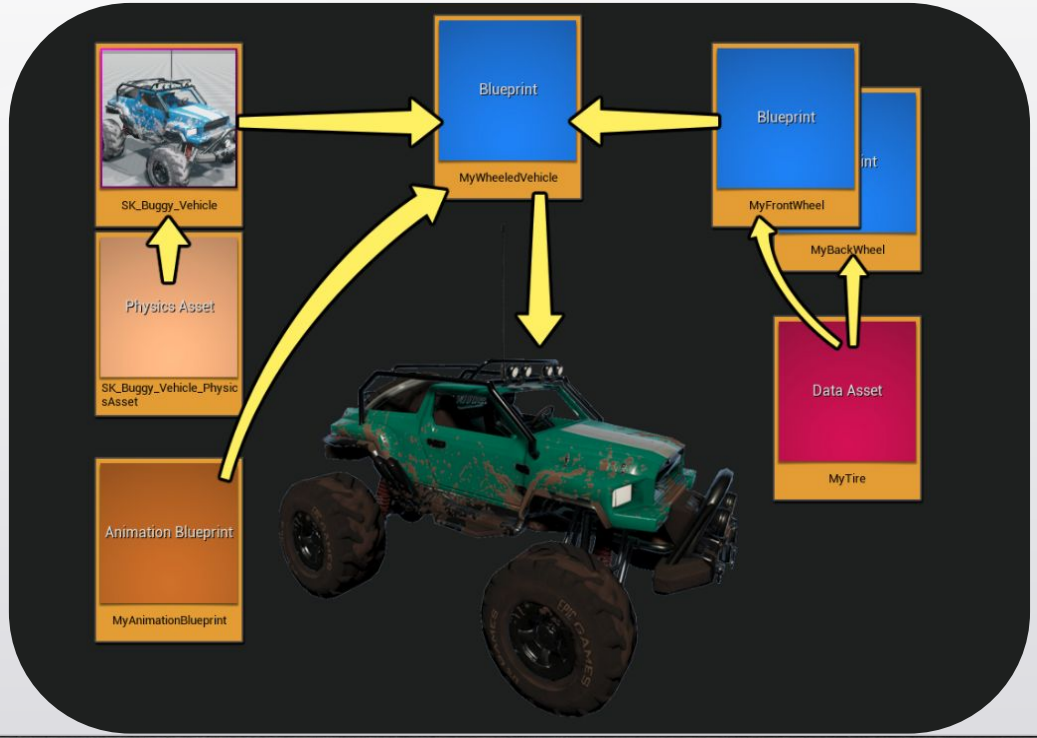
Adaption of ROS bridge with new architecture

- **New** or **updated ROS nodes**:
  - **carla_ros_bridge**.
  - **carla_ros_vehicle**.
  - **carla_ros_manual_control**.
  - **carla_ros_pcl_filter**.



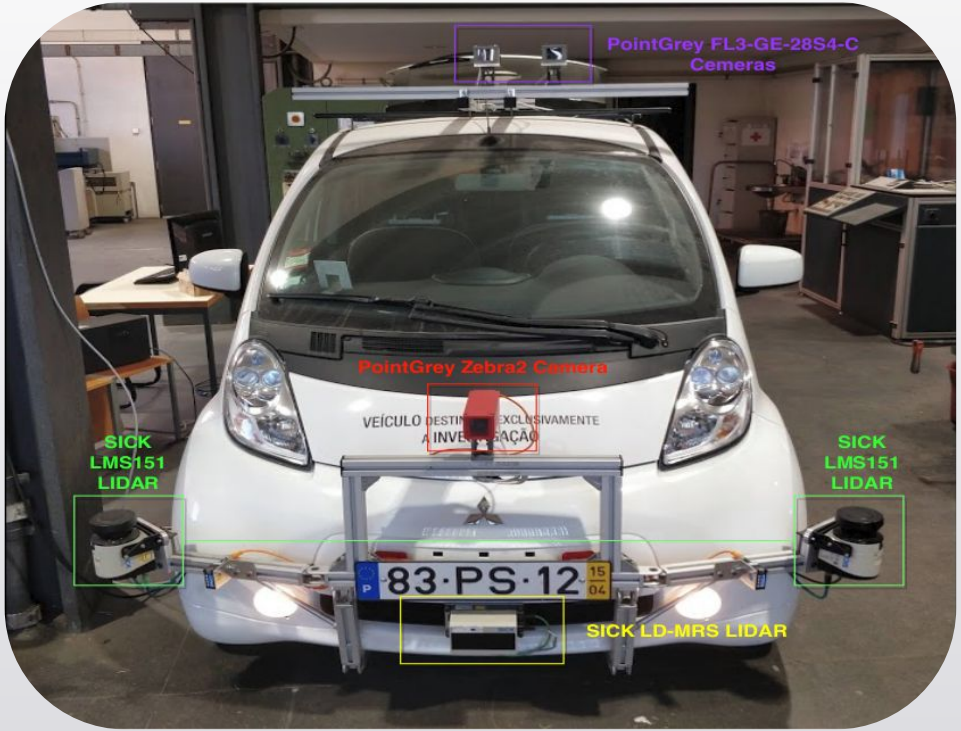
Blueprints in CARLA

- In **CARLA**, all **objects** are defined in **blueprints** which contains all the necessary **information** to create the object and spawn it as new **actor** in the **simulation**.
- The **blueprints** have an **ID** that **uniquely** identifies the **blueprint** and **all instances** of this blueprint. (e.g: “vehicle.nissan.patrol”, “sensor.camera.rgb”).
- The **sensor setup** is attached in the **vehicle blueprint** when you **design** it in **Unreal Engine 4**.
- This **blueprint model** includes:
  - **Skeletal Mesh.**
  - **Physics Asset.**
  - **Animation Blueprint.**
  - **Tire Configuration Data Asset.**
  - **1/+ Wheel Blueprints.**



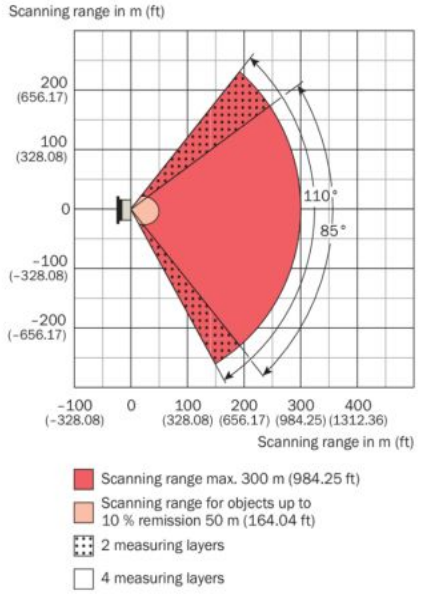
Experimental Infrastructure

- **ATLASCAR2:**
  - 1 Camera PointGrey Zebra2.
  - 2 Camera PointGrey FL3-GE-28S4-C.
  - 1 LIDAR SICK LD-MRS.
  - 2 LIDAR SICK LMS151.

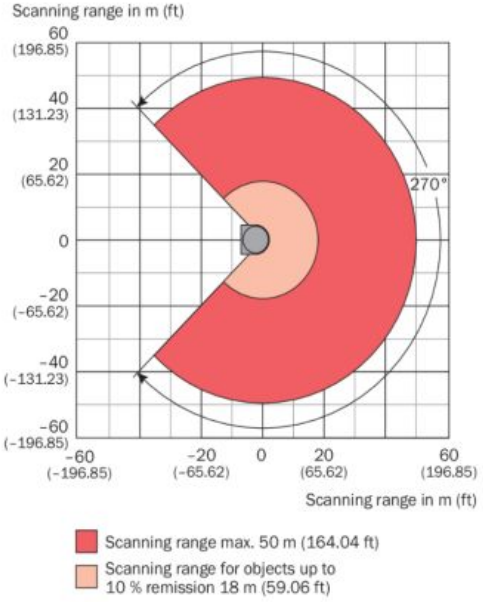


LIDAR Sensors

- SICK LD-MRS



- SICK LMS151



PointGrey Cameras

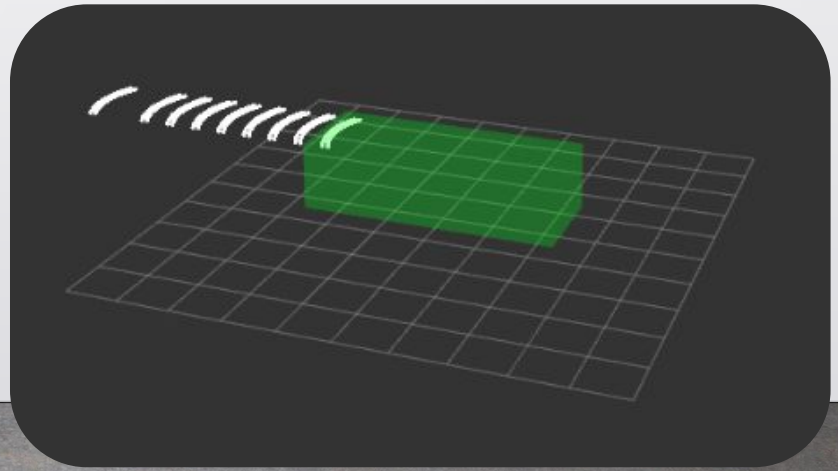
- PointGrey Zebra2

- PointGrey FL3-GE-28S4-C

ATLASCAR2 Sensor Setup Implementation in CARLA

- The **carla_ros_vehicle** node is the one **responsible** for **implementing** the **ATLASCAR2 sensor setup** in a **vehicle** and spawn this **vehicle** in the **CARLA simulation**.

- The **sensor setup list** is **defined** using a **.JSON configuration file**.

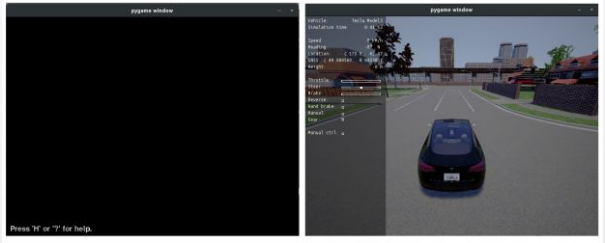


```
{
  "sensors": [
    {"type": "sensor.camera.rgb",
      "id": "front",
      "x": 2.0, "y": 0.0, "z": 1.0, "roll": 0.0, "pitch": 0.0, "yaw": 0.0,
      "width": 800,
      "height": 600,
      "fov": 100},
    {"type": "sensor.camera.rgb",
      "id": "view",
      "x": -4.5, "y": 0.0, "z": 2.8, "roll": 0.0, "pitch": -20.0, "yaw": 0.0,
      "width": 800,
      "height": 600,
      "fov": 100},
    {"type": "sensor.lidar.ray_cast",
      "id": "front",
      "x": 2.0, "y": 0.0, "z": 2.4, "roll": 0.0, "pitch": 0.0, "yaw": 0.0,
      "fov": 50}
  ]
}
```

Vehicle Control using ROS and Pygame Window

- The **carla_ros_manual_control** node **controls** the **vehicle** in the **simulation** via **sockets** and **ROS messages.**
- Based on the **manual_control.py** script.

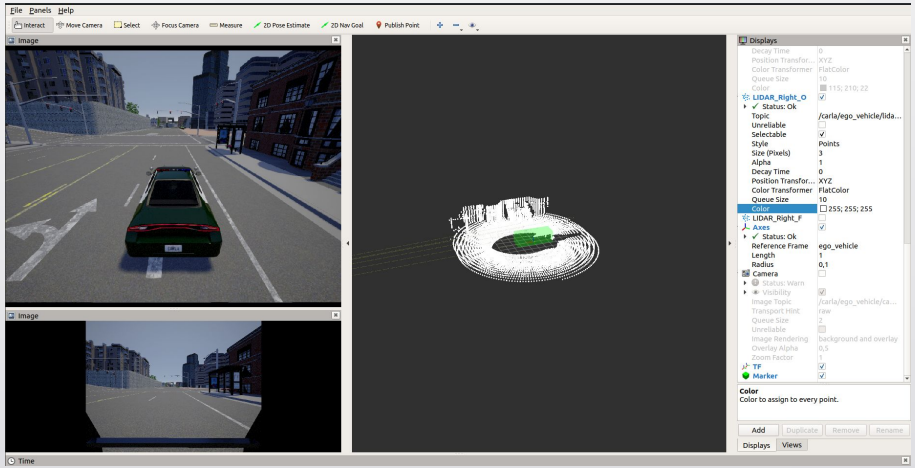
- This **spawned vehicle** is based on **random blueprints** from **CARLA** each one with **different sizes** and **different sensor positions.**



ATLASCAR2 Sensor Setup Configuration - #1

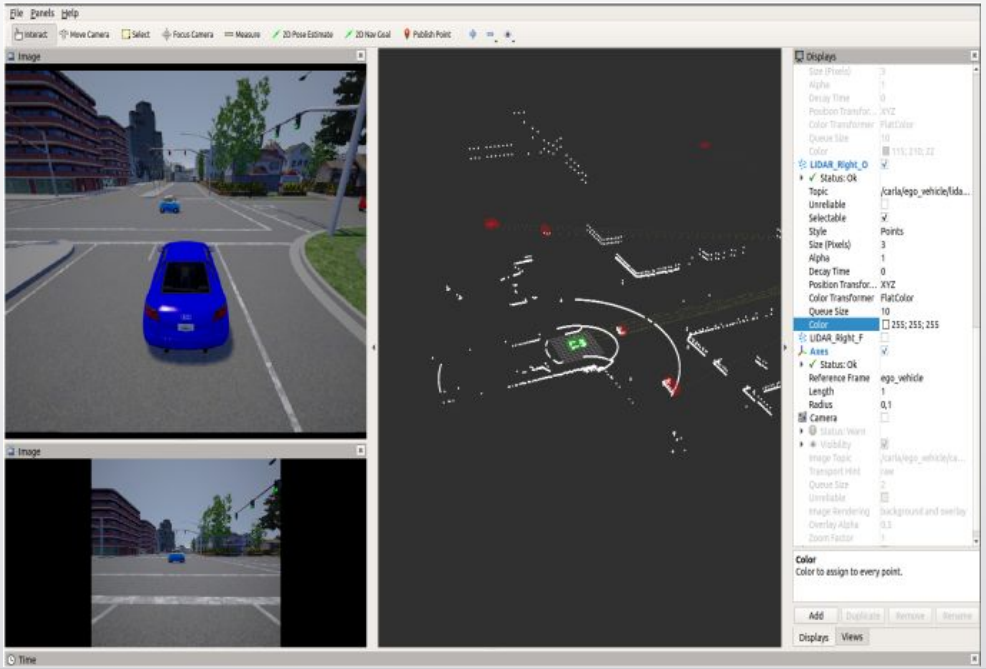
$$ray\_cast = \frac{points\_per\_second}{frames\_per\_second \times number\_of\_channels}$$

LIDAR Blueprints Attributes	Types	Default Values	Description
channels	int	32	Number of lasers
range	float	1000	Maximum measurement distance in centimeters
points_per_second	int	56000	Points generated by all lasers per second
rotation_frequency	float	10.0	Lidar rotation frequency
upper_fov	float	10.0	Angle in degrees of the upper most laser
lower_fov	float	-30.0	Angle in degrees of the lower most laser
sensor_tick	float	0.0	Seconds between sensor captures (ticks)

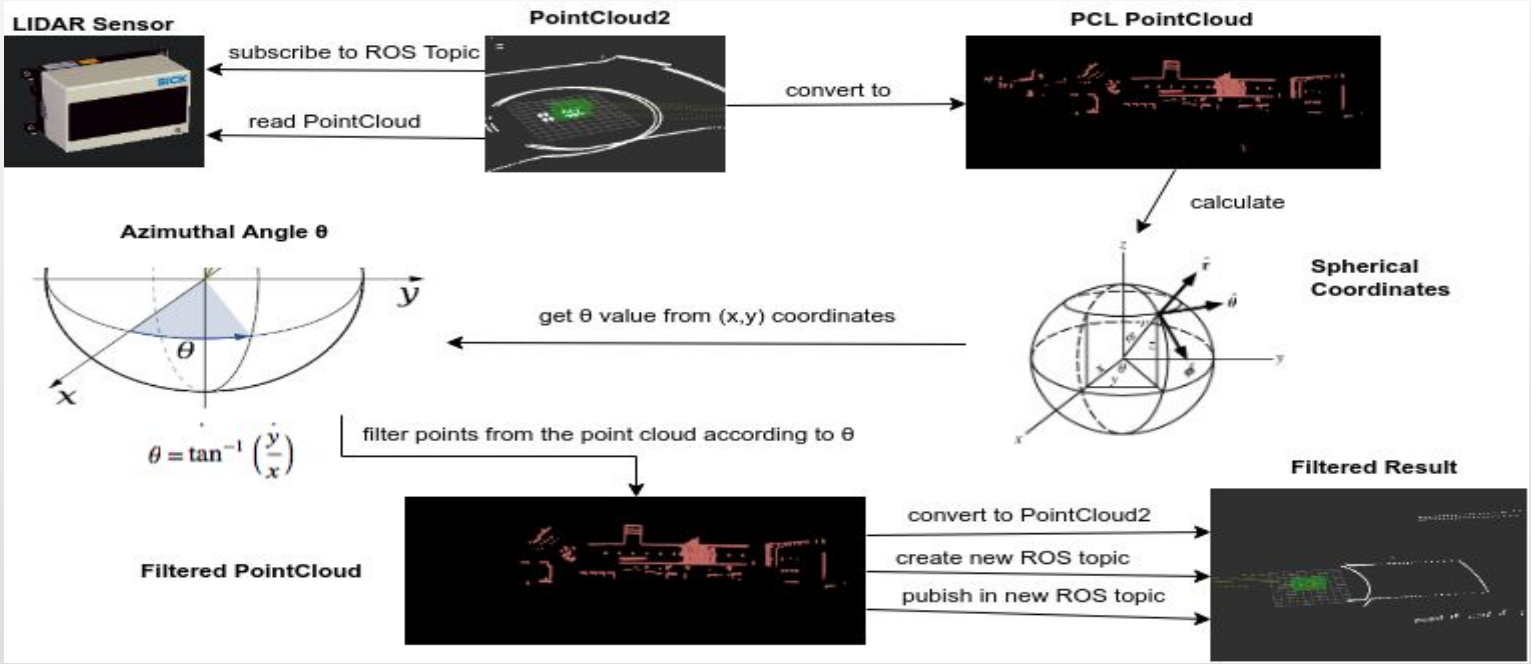


ATLASCAR2 Sensor Setup Configuration - #2

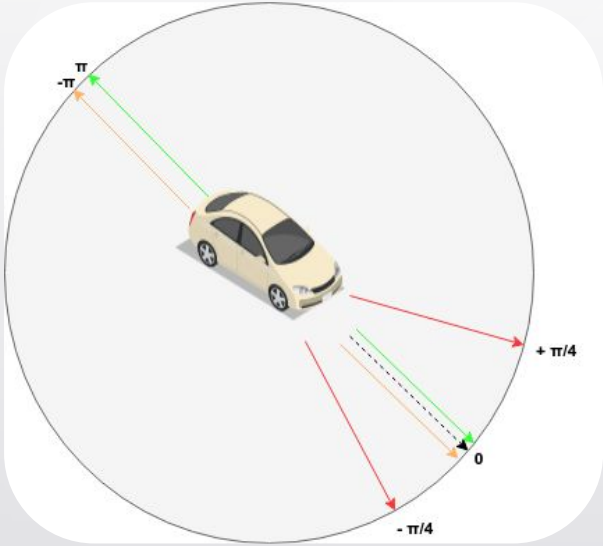
LIDAR Blueprints Attributes	Types	LD-MRS	LMS151	Description
channels	int	4	1	Number of lasers
range	float	25000	5000	Maximum measurement distance in centimeters
points_per_second	int	40000	27000	Points generated by all lasers per second
rotation_frequency	float	20.0	25.0	Lidar rotation frequency
upper_fov	float	1.6	0.0	Angle in degrees of the upper most laser
lower_fov	float	-1.6	0.0	Angle in degrees of the lower most laser
sensor_tick	float	0.02	0.02	Seconds between sensor captures (ticks)

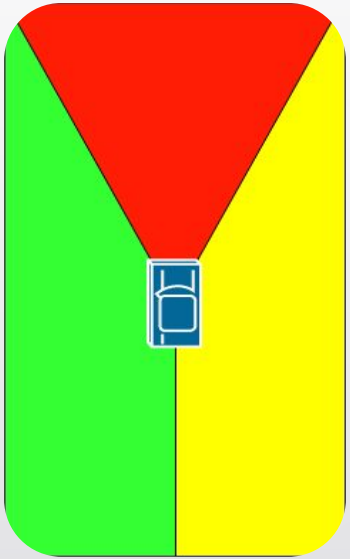


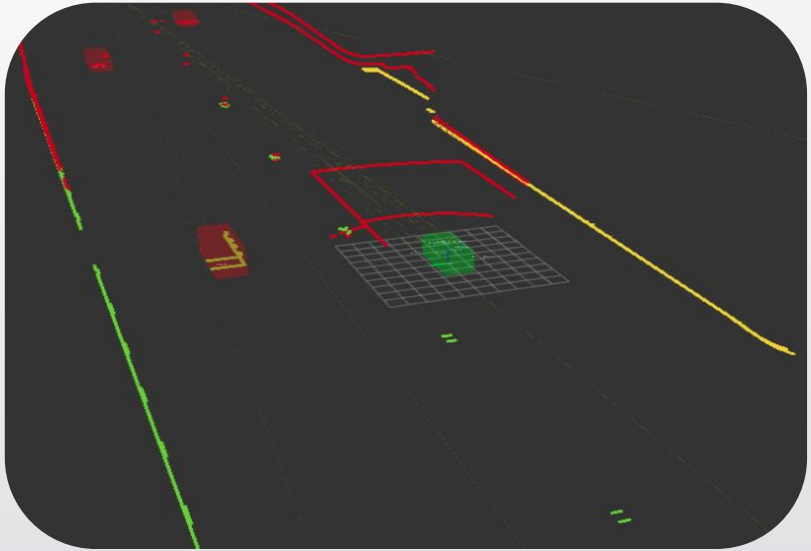
Point Cloud Filtering Algorithm



Filtering with Spherical Coordinates

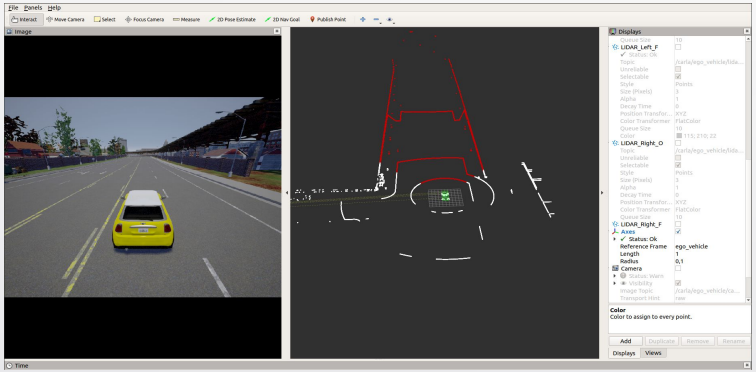


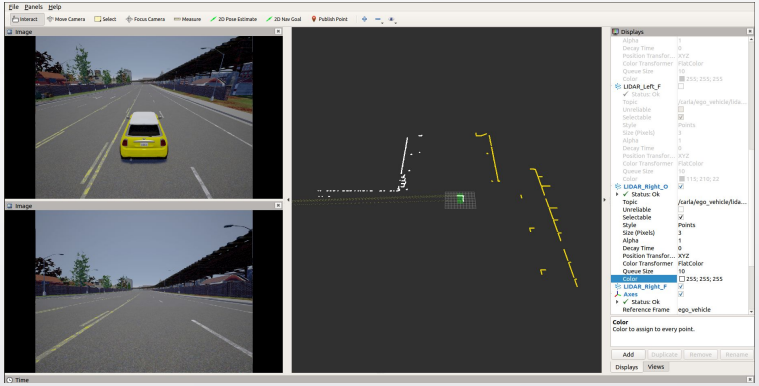


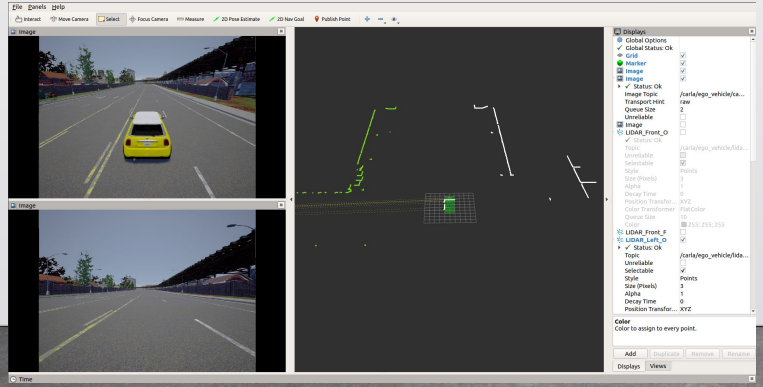


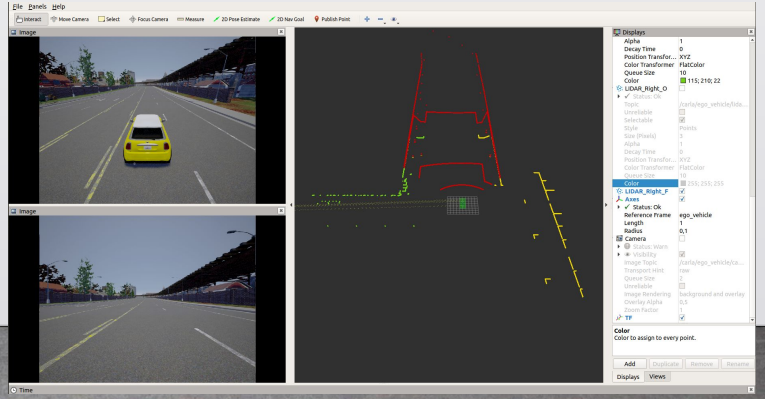
Sensor Point Cloud	Azimuthal Angle $\boldsymbol{\theta}$ Range Values in Radians
Front LIDAR (Red)	$-\pi/4 \leq \boldsymbol{\theta} \leq \pi/4$
Left LIDAR (Green)	$0 \leq \boldsymbol{\theta} \leq \pi$
Right LIDAR (Yellow)	$-\pi \leq \boldsymbol{\theta} \leq 0$

Filtering Results

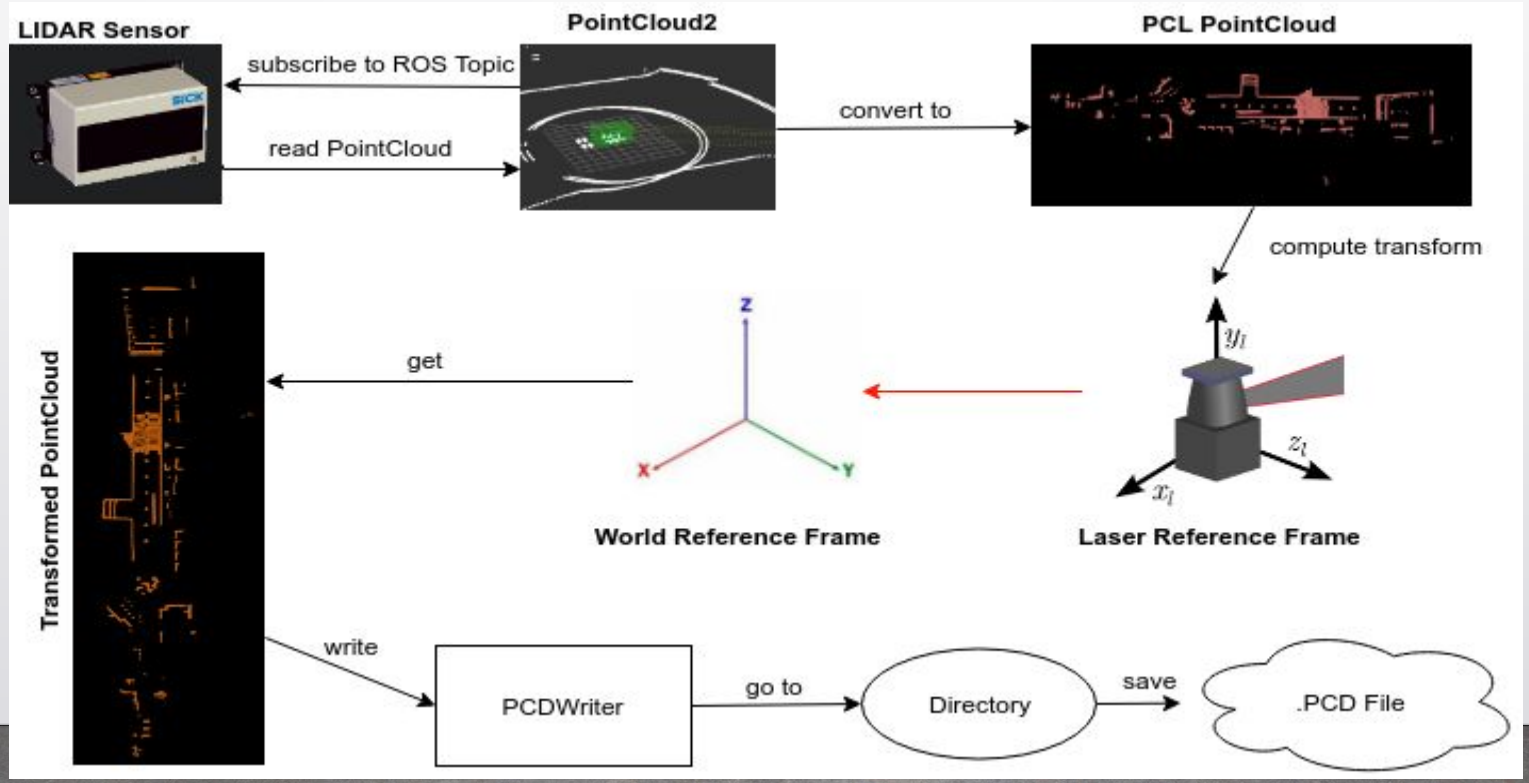






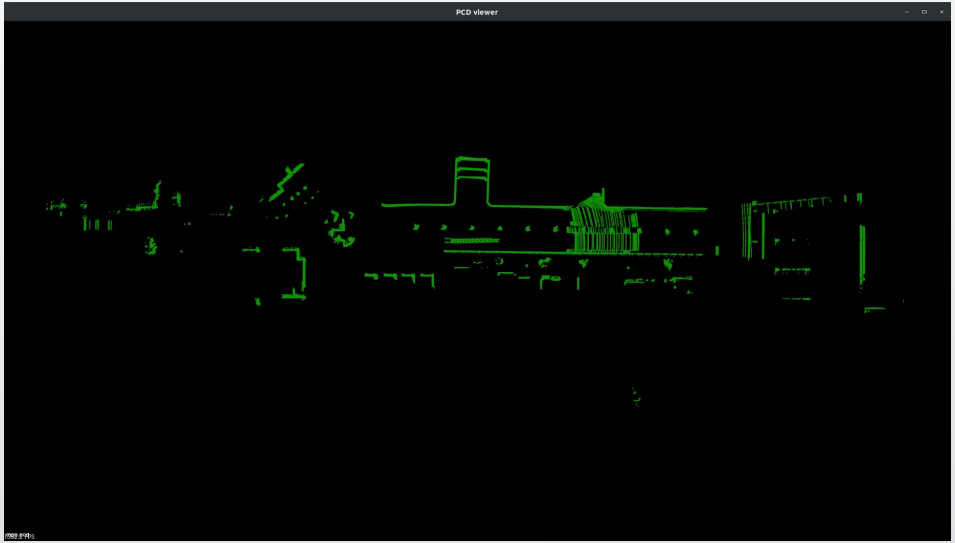


PCL Recording Package

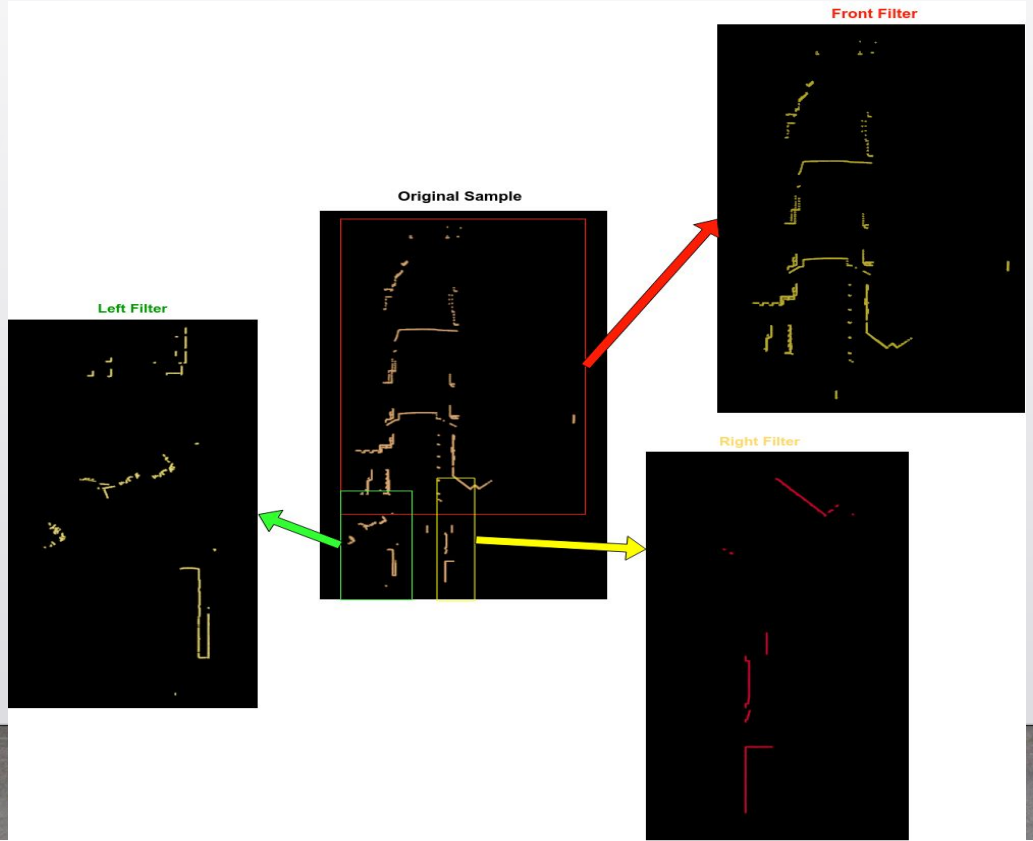


PCL Visualizer Package

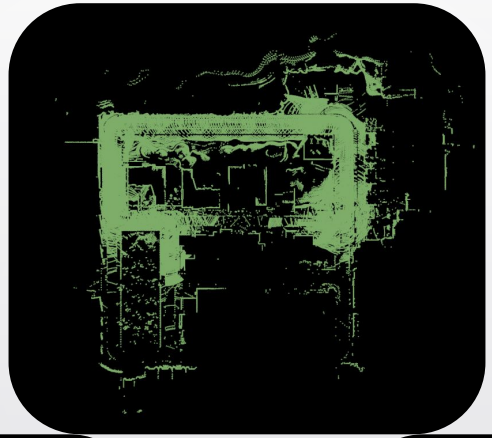
- Scans the recorded point clouds files.
- Concatenates the points to make sure the number of points in each cloud stays the same.
- Filters the duplicated results.
- Shows map result.

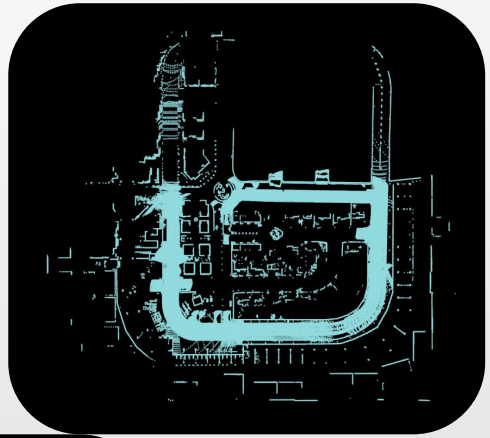


PCL Point Cloud Filtering Results

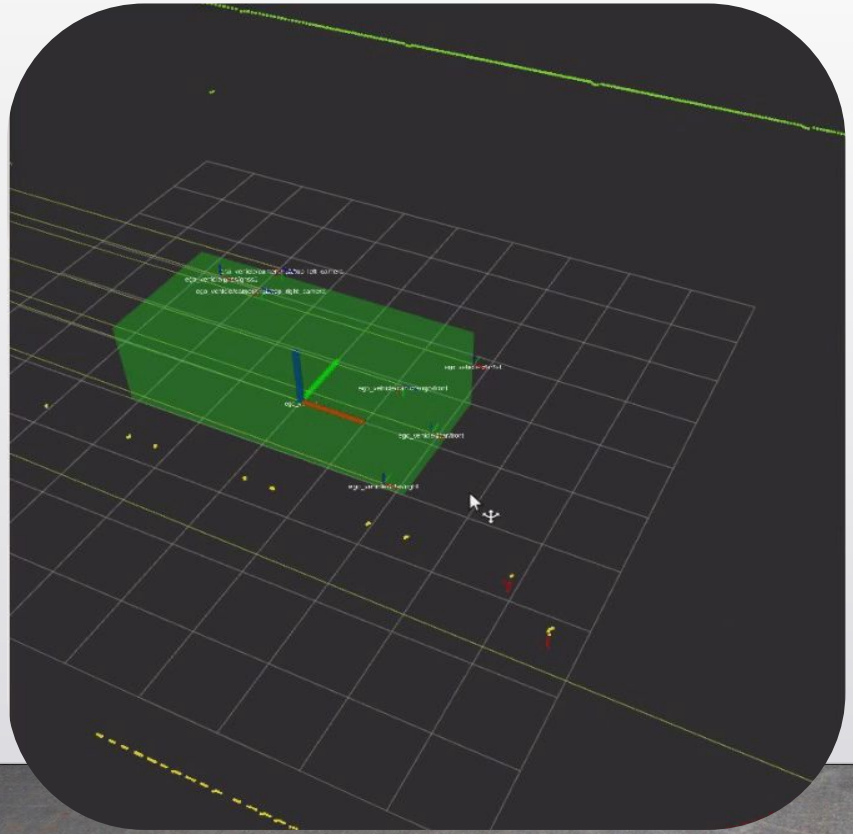


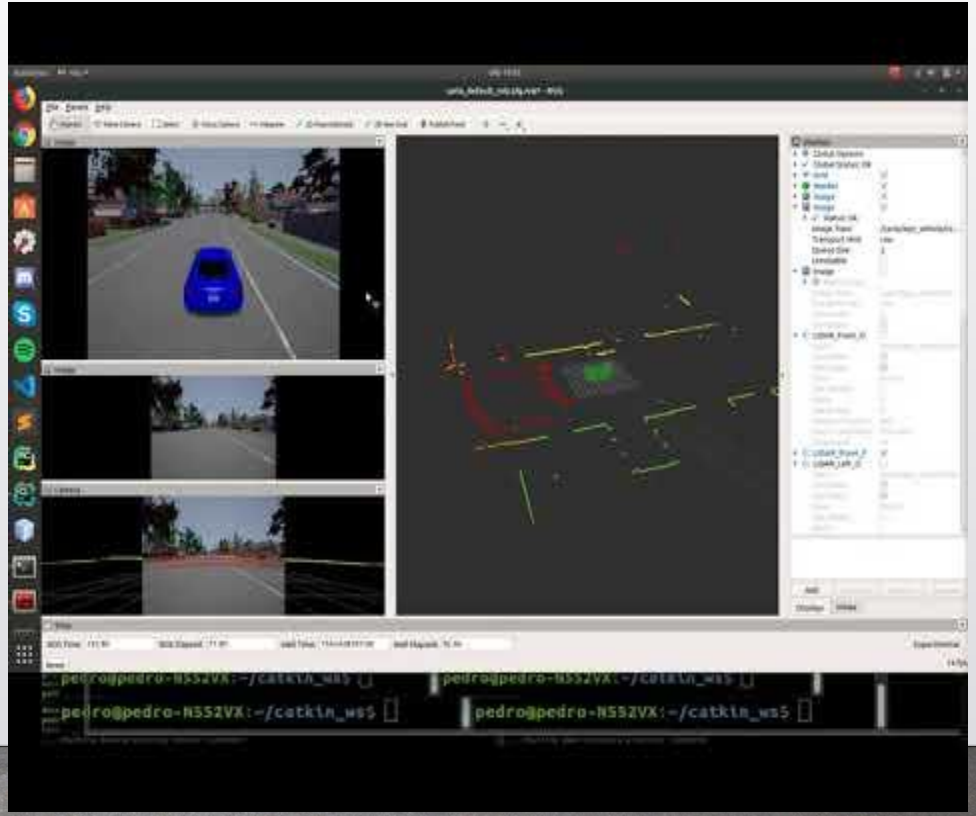
Map Results





Results





Experiments with CARLA simulated data

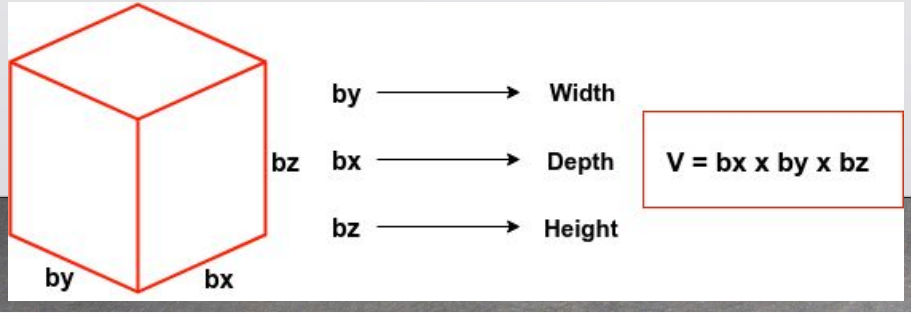
- **Simulated data** replicates the **ATLASCAR setup**.
- **Algorithms** driven by **real world data** may be **tested** with **CARLA**.
- **Validation testbeds** within **CARLA** as a **tool** to **evaluate** algorithms.
- **Performance measurement** of **algorithms** previously **used** on the **ATLASCAR**.
  - **Semi Automatic Detection**.
  - **Object Detection**.
  - **Road Visual Perception**.

Bounding Boxes in CARLA

- Each **object** in **CARLA** has a **bounding box** that **defines** the **limits** of the **object** and this **box** can be **activated** using **functions** present in the **CARLA** module.
- With the **bounding box information** it is **possible** to **create .JSON datasets** with the:
  - **(x,y,z) → position coordinates.**
  - **yaw → rotation.**
  - **(bx,by,bz) → extent of the bounding box.**
  - **model → blueprint ID.**
  - **classification → label.**

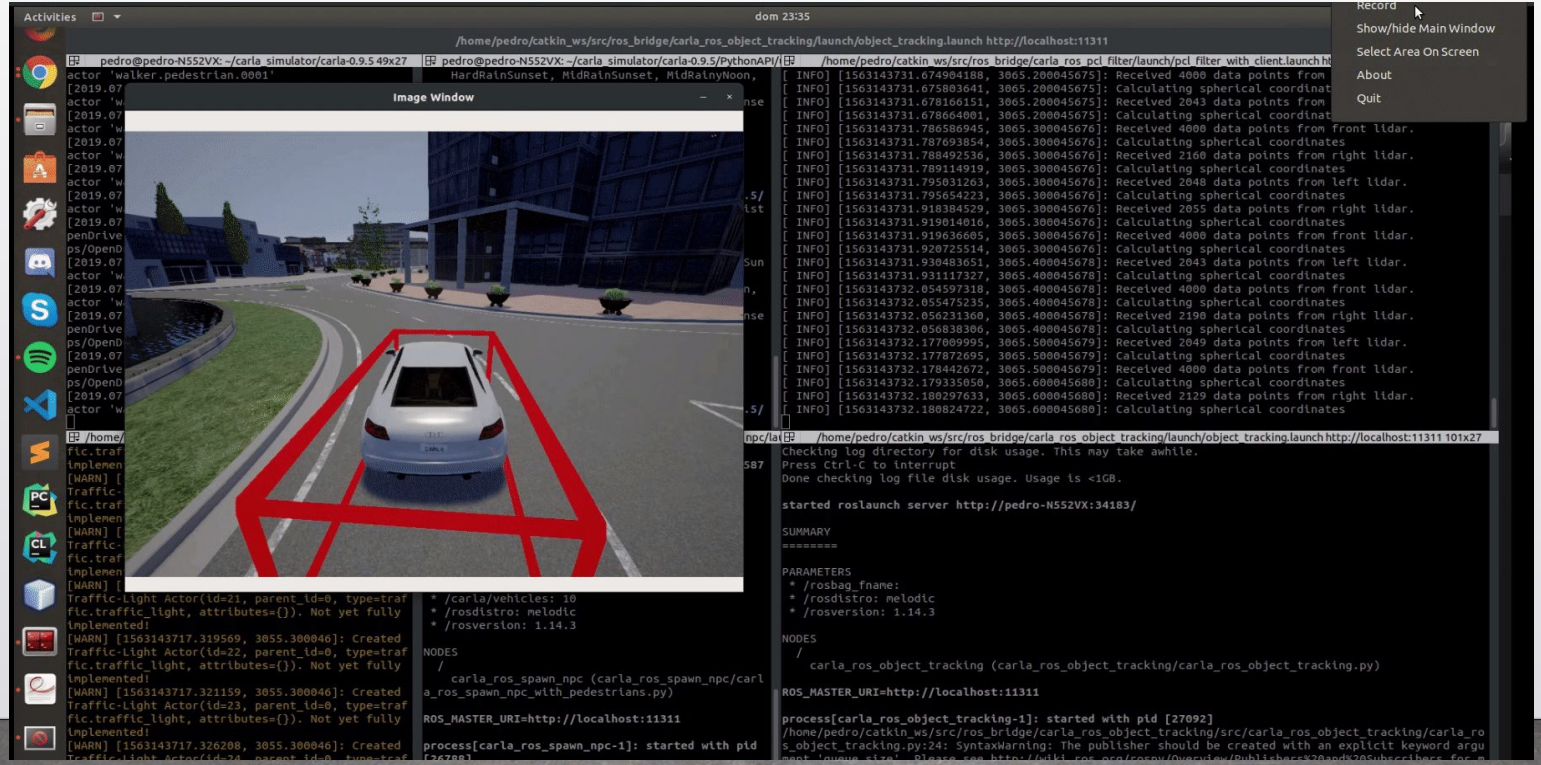
Bounding Box Volume	Label
$\leq 0.18\text{cm}^3$	Bicycle
$> 0.18\text{cm}^3$	Motorcycle

Bounding Box Volume	Label
$\leq 2.17\text{cm}^3$	Car
$> 2.17\text{cm}^3$	Truck

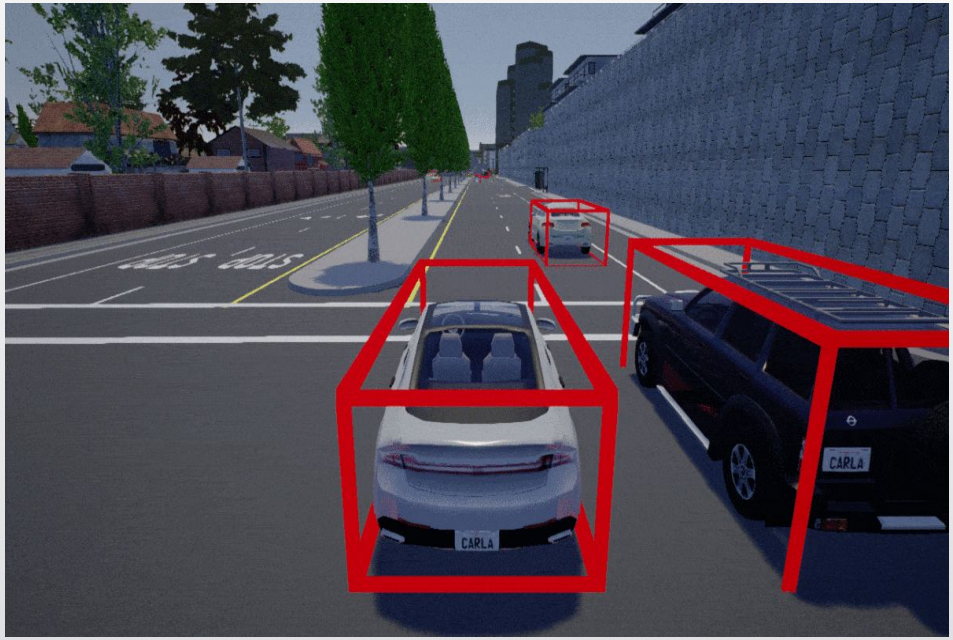


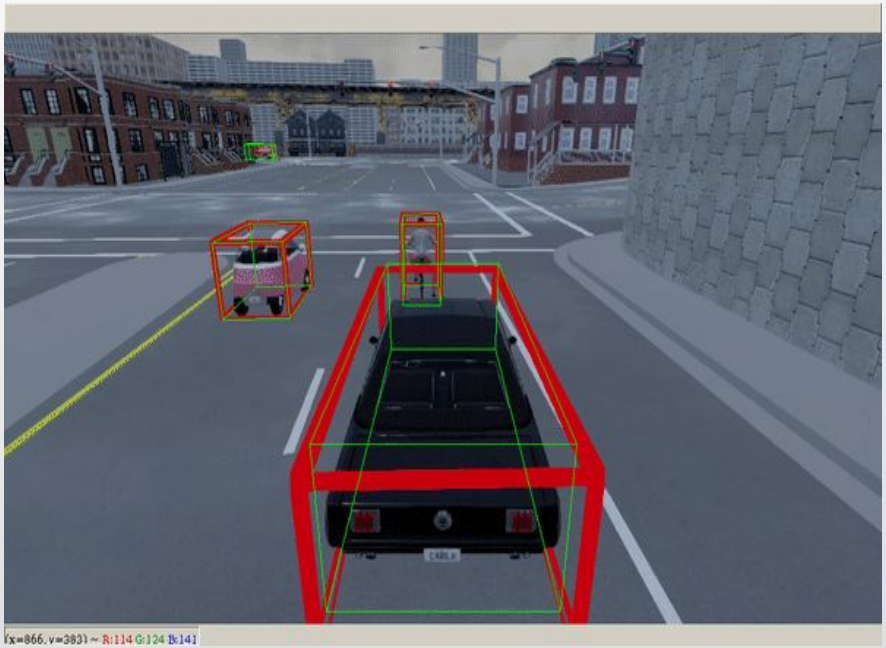

```
{
    "pedestrians": [
    {"yaw": 89.9999847412, "y": -13.9457063675, "x": -0.910356402397, "model":
    'walker.pedestrian.0001', "z": 1.04959559441, "bx": 0.340000003576, "class":
     'pedestrian', "bz": 0.879999995232, "by": 0.340000003576},
    {"yaw": 89.9999847412, "y": 12.9256744385, "x": 42.900844574, "model": '
    walker.pedestrian.0001', "z": 1.05390000343, "bx": 0.340000003576, "class":
    'pedestrian', "bz": 0.879999995232, "by": 0.340000003576},
    {"yaw": 89.9999847412, "y": 41.3421363831, "x": -95.6055526733, "model": '
    walker.pedestrian.0001', "z": 1.38003540039, "bx": 0.340000003576, "class":
    'pedestrian', "bz": 0.879999995232, "by": 0.340000003576},
    {"yaw": 89.9999847412, "y": 131.629211426, "x": 231.2709198, "model": '
    walker.pedestrian.0002', "z": 1.84568417072, "bx": 0.340000003576, "class":
    'pedestrian', "bz": 0.879999995232, "by": 0.340000003576},
    ...
    ]
}
```

Bounding Boxes in CARLA



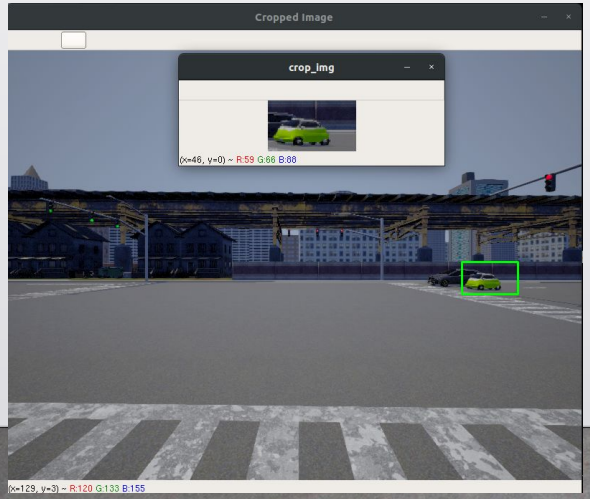
Bounding Boxes in CARLA





Template Matching

- **Finds areas** in an **image** that **match** (are **similar**) to a **template image** (**patch image**).
- The **template images** are from **template objects cropped** from the **CARLA world**.

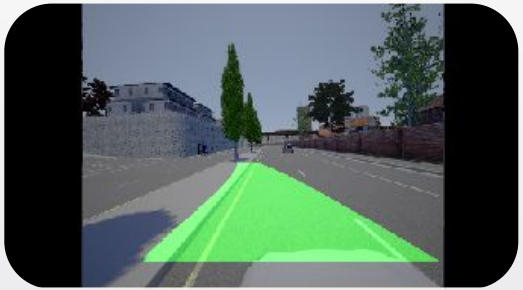


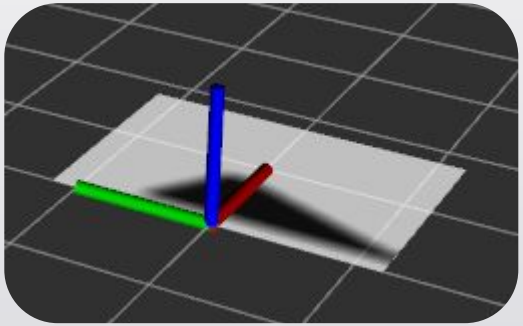
Road Visual Perception

- The **road visual perception algorithm** implemented by **Tiago Almeida** uses the **data** provided by the **two cameras** present on **top of the vehicle** to perform **lane detection** on the **road**.

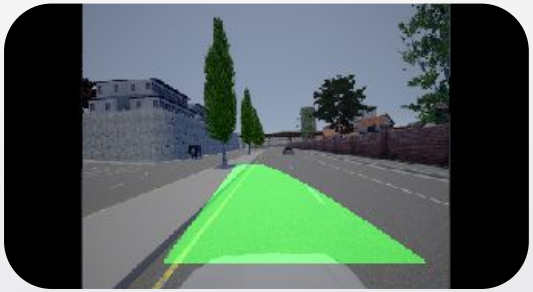
- The **probabilistic maps** determines the **probability** of having **road markings** on the **road.**
- Probabilistic results determine the **limits of the lane** and defines the **road overlay map**.

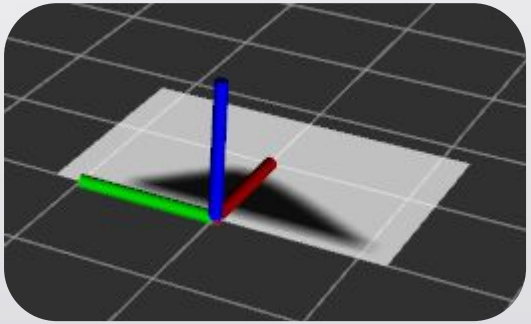
- **Top Left Camera**



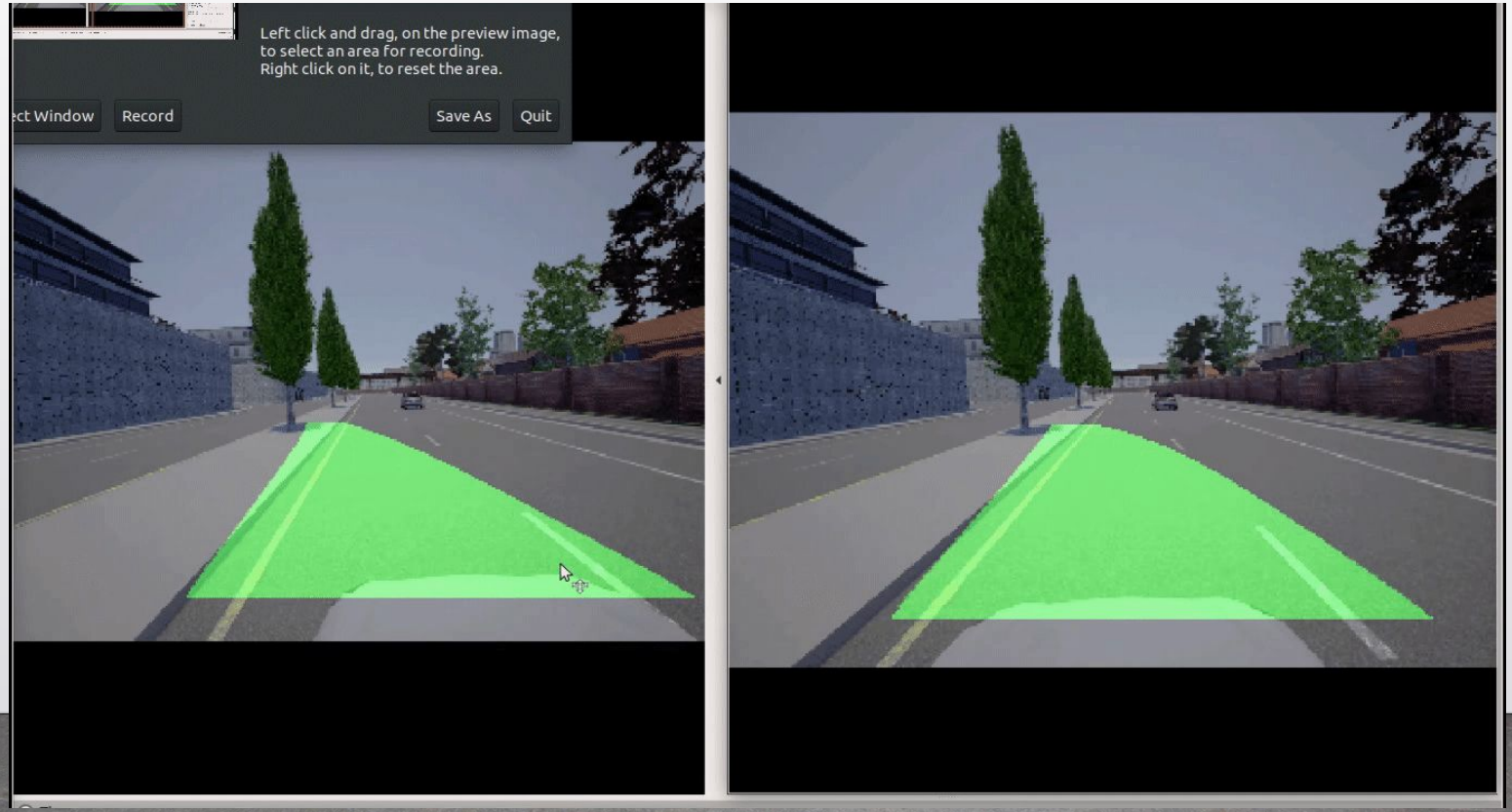


- **Top Right Camera**





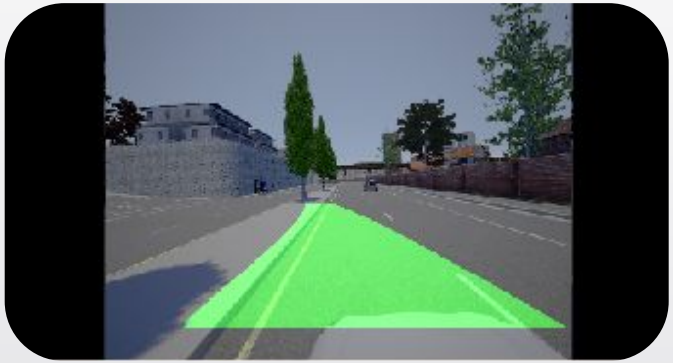
Road Visual Perception with Rosbag



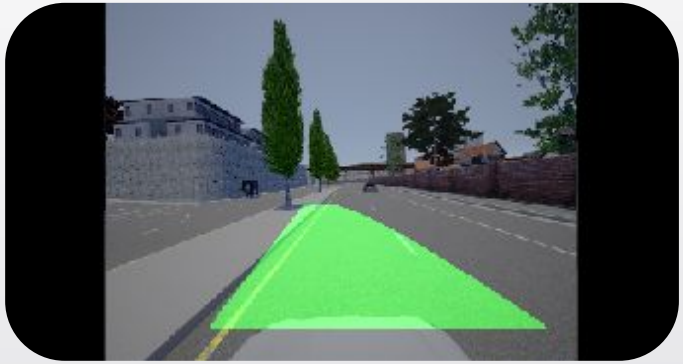
Road Visual Perception Comparison

CARLA

- **Top Left Camera**

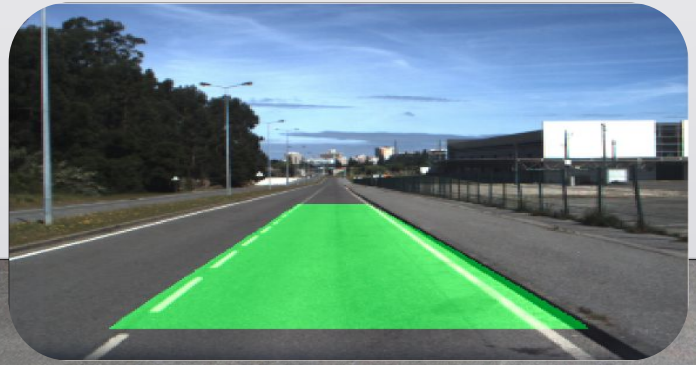


- **Top Right Camera**

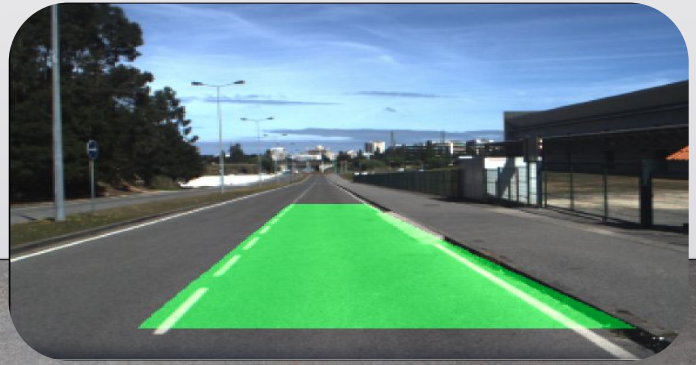


ATLASCAR

- **Top Left Camera**



- **Top Right Camera**



Main Contributions and Conclusion

- ROS nodes were developed to replicate the sensor setup of the ATLASCAR in the CARLA simulation environment.
- Improvements were made in the CARLA-ROS integration.
  - Flexible vehicle setup configuration.
  - Point Cloud Filtering, Recording and Visualization.
- Validation strategies with CARLA simulated data were applied in order to fit in the context of the ATLASCAR.
  - Bounding Boxes.
  - Template Matching.
  - Road Visual Perception.
- **CARLA can now be used to evaluate and perform tests with the algorithms of the ATLASCAR!**

Future Work

- Create datasets using fusion between camera and LIDAR sensor data.
- Create a hardware interface to mirror the ATLASCAR2.
- Evolve to an automatic evaluation of the point clouds.
- Work on a navigation method for the vehicle to follow a specific object around the CARLA world.

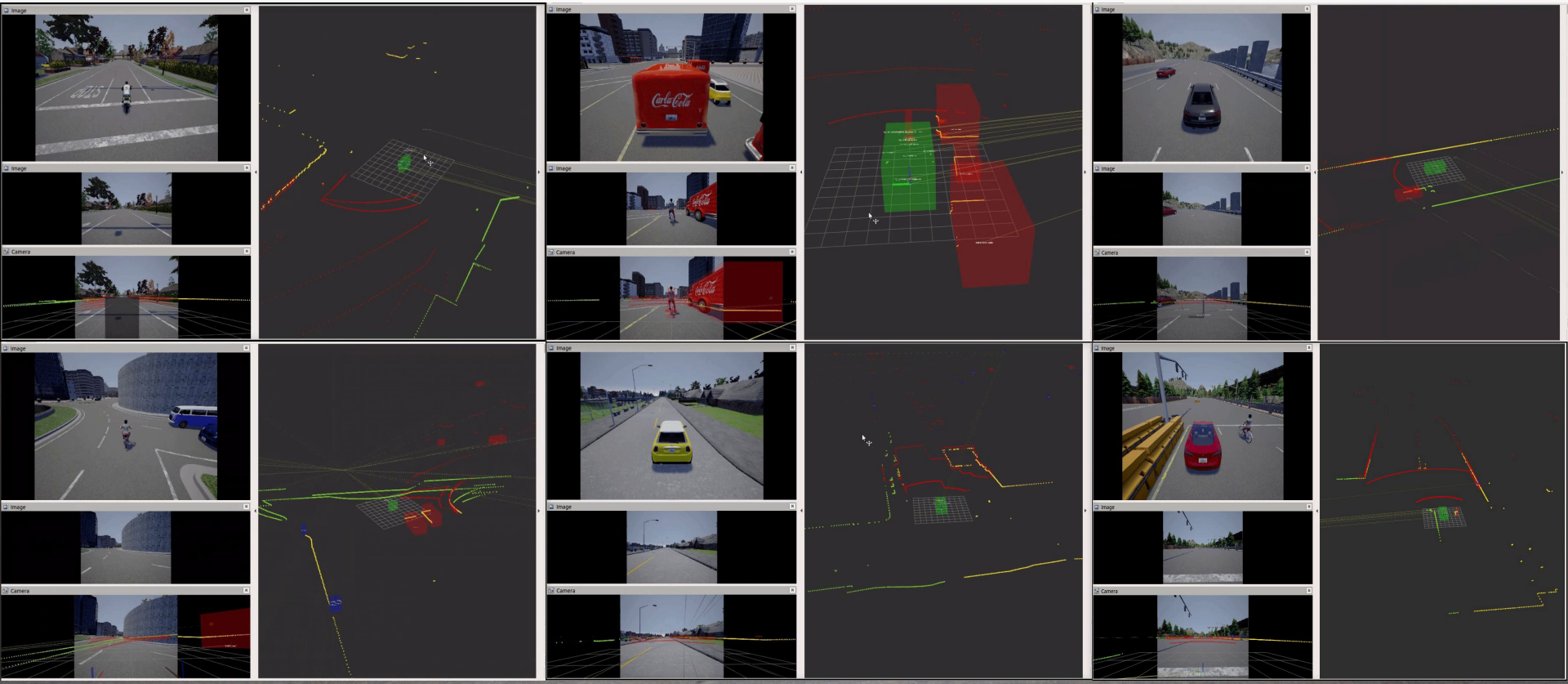


Image
Camera
Coca-Cola