Overview

This project was used to test how different approaches of creating critical scenarios play out when running the scenarios and evaluating the scenarios utilizing Autoware as the ADS inside the Carla simulator.

The methods used where CAT, activeDoE, and reactiveCAT.

For a more in depth explanation on how the methods work, please refer to the Masters Thesis document that explains everything extensively.

Set up the Bridge for Autoware to work in the Carla simulator

To set up the Carla-AW Bridge please follow the instructions provided here (best idea is to use the quickstart video that they have): https://github.com/TUMFTM/Carla-Autoware-Bridge

What needs to be changed is the docker image of AW to **ghcr.io/autowarefoundation/autoware:humble-2024.01-cuda-amd64** since the location of the image changed.

In case you are having troubles with the GPG keys of ROS (since this base image seems to use an outdated key) you can create a patched Dockerfile like this to update the key and then use this patched dockerfile in the next steps instead of the base one.

FROM ghcr.io/autowarefoundation/autoware:humble-2024.01-cuda-amd64

USER root

 # Replace the expired GPG key

*RUN curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key *

*| gpg --dearmor -o /usr/share/keyrings/ros-archive-keyring.gpg && *

*echo "deb [signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu jammy main" *

> /etc/apt/sources.list.d/ros2.list

Run Carla-Autoware

To now be able to use the Carla-Autoware setup together with the mats-trafficgen Code you need to change a few things.

- After starting the container of the Bridge you need to add 'spawn_sensors_only': 'True', to the launch_arguments inside carla_Autoware_ego_vehicle.launch.py in

tum/src/carla_autoware_bridge/launch

```
def generate_launch_description():
    ld = launch.LaunchDescription([
        launch.actions.DeclareLaunchArgument(
            name='objects_definition_file',
            default_value=get_package_share_directory(
                'carla_autoware_bridge') + '/config/objects.json'
        ),
        launch.actions.IncludeLaunchDescription(
            launch.launch_description_sources.PythonLaunchDescriptionSource(
                os.path.join(get_package_share_directory(
                    'carla_spawn_objects'), 'carla_example_ego_vehicle.launch.py')
            ),
            launch_arguments={
                'objects_definition_file': launch.substitutions.LaunchConfiguration(
                    'objects_definition_file'),
                'spawn_sensors_only': 'True',
            }.items()
        )
    ])
    return ld
```

- In tum/src/carla_autoware_bridge/config there is an objects.json file.
  You might need to change the id of the ego to “ego_vehicle” there if its not like this already. And the type to “vehicle.volkswagen.t2_2021” since this is what the mats-traffic-gen project uses.

----------To now run the complete setup you first need to start **CARLA**----------

Best case is to run it with docker like this:

docker run --privileged --gpus all --net=host -e DISPLAY=$DISPLAY carlasim/carla:0.9.15 /bin/bash ./CarlaUE4.sh

In case this fails you can also install Carla from scratch and just run the .sh file like this: *./CarlaUE4.sh*
Documentation can be found here: https://carla.readthedocs.io/en/latest/start_quickstart/

----------Next we need to start the Bridge----------

IF you have setup everything as described in the Carla-Autoware_Bridge repository you can start the docker container like this:

docker run -it -e RMW_IMPLEMENTATION=rmw_cyclonedds_cpp --network host tumgeka/carla-autoware-bridge:latest

To start the Bridge **manually** (**not needed** for executing with mats-traffic-gen setup) you can then call:

ros2 launch carla_autoware_bridge carla_aw_bridge.launch.py port:=2000 passive:=True register_all_sensors:=False timeout:=180

in the next step.

----------Next we need to start Autoware----------

For this you either start the container like this:

rocker --network=host -e RMW_IMPLEMENTATION=rmw_cyclonedds_cpp -e LIBGL_ALWAYS_SOFTWARE=1 --x11 --nvidia --volume /work/Valentin_dev/tumgeka_bridge --ghcr.io/autowarefoundation/autoware:humble-2024.01-cuda-amd64

Or build the patched image in case you had troubles with the GPG keys and run the container like this:

docker build -f docker/Dockerfile.autoware-patched -t autoware-humble-patched .

→

*rocker --network=host *

*-e RMW_IMPLEMENTATION=rmw_cyclonedds_cpp *

*-e LIBGL_ALWAYS_SOFTWARE=1 *

*--x11 --nvidia *

*--volume /work/Valentin_dev/tumgeka_bridge *

-- autoware-humble-patched

Again to run manually (**not needed** when running the complete project) you can start autoware like this:

cd /work/Valentin_dev/tumgeka_bridge/autoware_latest/autoware/

source install/setup.bash

ros2 launch autoware_launch e2e_simulator.launch.xml vehicle_model:=carla_t2_vehicle sensor_model:=carla_t2_sensor_kit map_path:=/work/Valentin_dev/tumgeka_bridge/Town10

----------Run project----------

If you have started all containers you can then specify the container names of the running containers in the run_xosc_scenario.py file as well as choose the desired strategy to create critical scenarios:

*parser.add_argument('--strategy', type=str, default="**cat** ")*

and execute it like this:

./scripts/run-docker-xosc_scenario.sh

----------Side Note Running activeDoE----------

TO be able to use activeDoE for creating critical scenarios you need to run the activeDoE server docker image such that the project can connect to it.

docker run -it --rm -p 8011:8011 docker.avl.com/avl/cameo/active_doe:latest

You might need a new license for this which can be granted by the Cameo guys.

How the project works

When running the project from *./scripts/run-docker-xosc_scenario.sh* it runs the run_xosc_scenario.py file. In this file you can specify which procedure (CAT, activeDoE, RCAT, random) should be run and which base xosc scenario you want to play.

Depending on this the script will correctly initialize the environment, spawn the cars in the correct positions and provide the correct goal position to AW.

Depending on the strategy used to create critical scenarios the base scenario will be played first or a critical scenario will be directly executed.

Since the simulation environment sometimes hangs due to segfaults or other reasons there is a small helper script *controller.py* that can be used to periodically restart all containers to best prevent such problems.

For further information on the critical scenario generation methods please have a look at the paper published for CAT: https://arxiv.org/abs/2310.12432 , take a look at the Masters thesis document for understanding reactive CAT, or talk to the responsible AVL people for activeDoE.

To restrict the generated critical scenarios to some ODD some rules were established and enforced within the code. In short, the trajectory of the attacking vehicle was not allowed to drive of the driveable area, it was not allowed to exceed a certain speed and it was not allowed to change into opposite lanes. For further details please also have a look at the MT document.