# Road-aligned MPC for CARLA

*Release 05.03.2021*

**Michael Seegerer**

**Apr 05, 2021**

# CONTENTS:

CHAPTER

ONE

# INTRODUCTION -

This documentation goes into detail on how to use an MPC control structure for controlling vehicle in CARLA. It starts by showing how to start a CARLA server with scripts. Later, the MPC control design framework will be explained based on the example script 'simpleCar_MPC2.py.' Furthermore, the limitations and known problems will be outlined. In the end, an auto-documentation of the single functions of the MPC is presented.

CHAPTER

# TWO

# METHOD -

This chapter depicts how to operate CARLA with MPC control scripts and how to start a CARLA server. First, a small summary of the necessary installation steps is given to initialize the CARLA system like assumed in the later sections. The second section assumes that CARLA system is already installed.

## 2.1 Quick start guide

If everthing is already installed, the MPC CARLA simulation can be started as followed:

1. Opening up two terminals.
2. Starting the carla server in the first terminal with:

```
carla-serv
```

3. Navigate in the second terminal to the SMPC_Carla directory by:

```
cd SMPC_carla/
```

4. Start the simulation in the second terminal with the following command (but wait till carla server runs smoothly):

```
python3 simpleCar_MPC2.py
```

## 2.2 Installation

This section shows the required installation steps to retrieve the same system as the carla-simulation computer located in the LSR student lab.

### 2.2.1 Installing CARLA

To install CARLA system follow the instructions on CARLA install.

The most important commands are the following:

```
pip install --user pygame numpy
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 1AF1527DE64CB8D9
sudo add-apt-repository "deb [arch=amd64] http://dist.carla.org/carla $(lsb_release -
↪sc) main"
sudo apt-get update # Update the Debian package index
```

(continues on next page)

(continued from previous page)

```
sudo apt-get install carla-simulator # Install the latest CARLA version, or update␣
↪the current installation
cd /opt/carla-simulator # Open the folder where CARLA is installed
```

The following sections are assuming that CARLA is installed as a Debian installation.

To make CARLA accessible as a python package, the CARLA PythonAPI package has to be installed with the carla-*py3.*.egg file located inside CARLA installation location at /opt/carla-simulator/PythonAPI/carla/dist/.

### 2.2.2 Installing casADi

The MPC control method uses casADi to solve the optimal control problem (OCP). Therefore, casADi has to be installed before starting any MPC script for CARLA.

CasADi provides an extra mpctools package for python, which is available on MPCTOOLS CASADI.

Use the following command inside the downloaded mpctools directory to install the python package mpctools for casADi:

```
python3 mpctoolssetup.py install --user
```

### 2.2.3 Installing the MPC CARLA package

The MPC control methods can also be installed as a python package by executing the following inside the *SMPC_Carla* directory:

```
pip3 install --user -e .
```

## 2.3 Usage CARLA

The CARLA simulator consists of a scalable client and server architecture (Fig. 2.1).

- **CARLA server:** Responsible for everything related with the simulation itself: computation of physics, updates on the world-state and its actors and much more. As it aims for realistic results, the best fit would be running the server with a dedicated GPU, especially when dealing with machine learning.
- **Client side:** Consisting of a sum of client modules controlling the logic of actors on scene. A example for a client script is the script 'simpleCar_MPC2.py'.

### 2.3.1 Starting the CARLA Server

To start a CARLA server use the following command:

```
carla-serv
```

In case this commands is not already in your local bash-alias list, you can add the command by:

```
echo "alias carla-serv='bash /opt/carla-simulator/CarlaUE4.sh'" >> ~/.bashrc
```

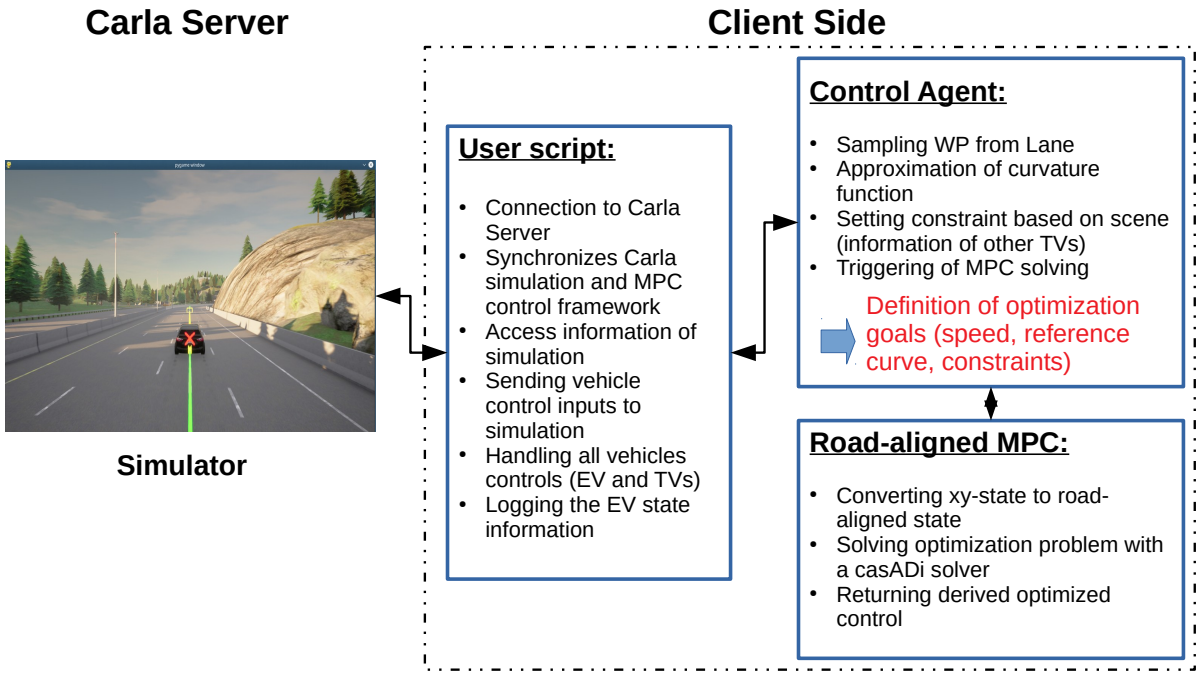


Fig. 2.1: Client-server architecture overview.

### 2.3.2 Starting the MPC client script

The necessary layout of a CARLA script using the MPC control approach is presented in 'simpleCar_MPC2.py'. To start a script inside CARLA, a CARLA server must be running, and then in a second terminal, a script can be executed.

The scripts must be executed with python3; python2 causes malfunction. Example:

```
python3 simpleCar_MPC2.py -f './logs/ff_states_log.h5'
```

With this script, an additional window should open, which shows a car on the CARLA TOWN04 (Fig. 2.3) controlled by MPC control logic. The passed arguments with -f refers to the desired filename of the log-data and the saving location. Other possible arguments to pass along with the script are the size of the up-poping window, with -x and -y for the corresponding sizes. It also possible to change the port of the CARLA server to listen to by passing -p and the respective port number.

In the visualization window (shown in Fig. 2.2) , the green line shows the reference curve used in the MPC OCP problem. The yellow lines show the prediction from implemented MPC vehicle model by using the derived controls. Crosses mark sample waypoints, which are used in the reference curve approximation and the conversion to a road-aligned EV-state.


Fig. 2.2: Visualization of CARLA with MPC control script.

## 2.4 Structure of the MPC approach

The design of the MPC controller can be split into the three major parts, as also shown in Fig. 2.1:

1. **Script to interact with *CARLA* server:** Synchronization of the optimization process with the CARLA server. Triggering a new Carla frame after finishing the other two scripts Sending the derived optimal control to the CARLA server.
2. **Script to generate reference curve and handle further constraints:** Mathematical approximation of the reference curve from the given road layout information from the CARLA server. Providing the OCP with an optimization goal and with scenario constraint based on the current scene.Performes the logging process of the Frenet Frame states.
3. **Road-aligned MPC class:** Solving the OCP by having the position of the CARLA vehicle, the reference curve, and WP_C and WP_N. The solving process is solved every 6th simulation timestep, in the other simulation timesteps the previous derived optimal control is used. (simulation timestep = 1/30 seconds ==> solving OCP every 0.2 seconds) The road-aligned MPC class is implemented in the model_predicitve_control.py.

All the basic functions to convert the ego vehicle (EV)'s xy-state to a road-aligned state are collected in the file waypoint_utilities.py.

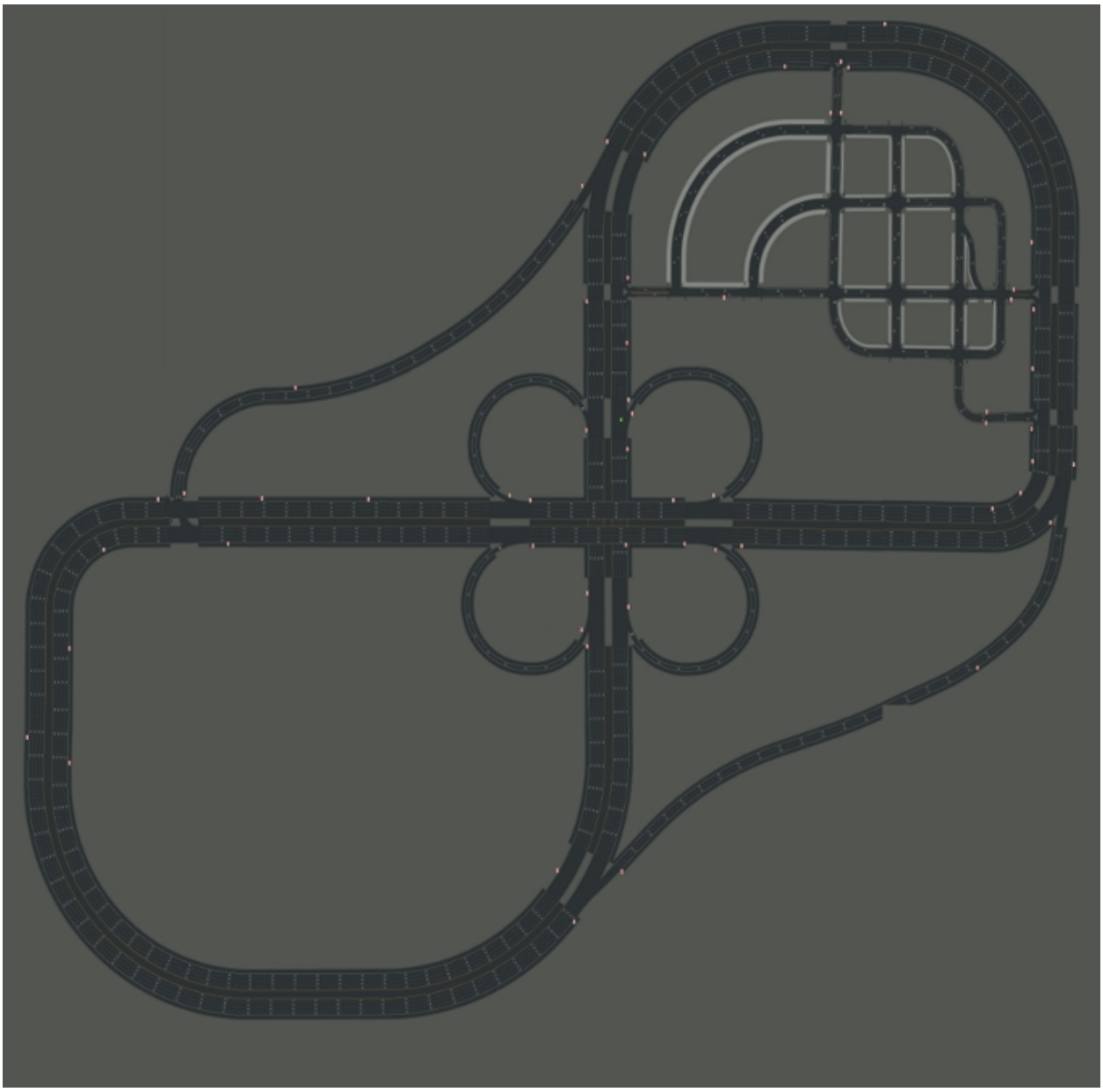

Fig. 2.3: Town4 of the CARLA map package.

### 2.4.1 MPC Constraints

So far only three simple cases of traffic scenes are realized in simpleCar_MPC_TVs.py

1. Case B: one target vehicle (TV) in front (same lane) is consider for constraints –> vertical xi constraint.
2. Case D: one TV in front (same lane) is considered, constraint set to overtake the TV.
3. Case B + F: one TV vehicle in front and one TV on right lane of EV consider in constraints.

The control agent class is responsible for analyzing the scene and sending the fitted constraint to OCP problem of the MPC.

The constraint are split into hard and soft constraints. Hard constraints limiting the legible state space by considering the time-varying position of the other vehicles. Soft constraints ensure a desired behaviour depending on the scene. For example in case B, a vertical constraint approximately 30 meters beyond the rear of the TV front of the EV establish a safe distance to the TV in front.

### 2.4.2 MPC - FT - Architecture

The script simpleCar_MPC_FT.py, control_agent_mpc_ft.py, ft_mpc.py contain a draft version of a SMPC + FT control layout. The general framework is similar to the one shown in Fig. 2.1, however here the control agent contains additionally a second MPC controller. This second MPC controller derives a failsafe trajectory based on the predicted state after applying the derived u_0 control of the first agent. The framework is described in [Bruedigam2020] in great detail.

The constraints for the FT MPC method works similar like to the normal MPC, the control agent sends them based on the current scene (other TVs). A difference is that the FT MPC class method holds to every time a FT safe trajectory for the EV (accessible with agent_ft.controller.failsafe_trajectory). Then if the FT controller does not find a legible solution, the saved ft trajectory gets reinitialized with a emergency braking one.

## 2.5 Visualization of the MPC performance

To evaluate the performance, the EV's road-aligned state will be logged and later be plotted. A variety of helping plotting functions can be found in 'visualization_tools.py'.

The jupyter-script 'Visualization_frenet.ipynb' gives an example of helping evaluation plots.

### 2.5.1 Quick visualization guide

1. Open new terminal and navigate to SMPC folder

```
cd SMPC_carla/
```

2. Opening a jupyter server with:

```
jupyter-lab
```

3. Select the Visualization_MPC.ipynb file and change the variable file to the desired recording to visualize
4. To Execute a row of code in jupyter, use the enter key in the desired row (every has to be reexecuted by restarting the notebook)

CHAPTER

# THREE

# DISCUSSION -

This chapter discusses some known problems of the CARLA simulation framework.

## 3.1 Vehicle Models Difference

The MPC controller currently uses a kinematic vehicle model, a simplified approximation compared to CARLA's realistic model, especially in curves. Because kinematic models are a function of the vehicle geometry, they can represent the vehicle motion in a range of conditions that do not involve highly dynamic maneuvers and /or tire force saturation. Therefore, the MPC prediction has problems in the curve situation with a higher EV's velocity and the start (== zero EV's velocity). This model mismatch leads to a small lateral offset in curves.

MPC assumes that the first steering input is enough to bring the vehicle back to the curve centerlane, then only smaller inputs are planned to keep the vehicle on the centerlane. However, the first MPC input is not sufficient in the CARLA simulation, as the kinematic model overestimates the influence of the steering angle on the lateral motion.

## 3.2 casADi calculation errors

By manually calculating the prediction steps under the suggested control from MPC, slightly different values result compared to casADi can be observed. The reason is still unclear.

## 3.3 Reference curve problem

The reference curve generation process has some problems getting the right sampling waypoints WP during intersections. Here, the MPC may assume that the passed waypoints of EV comes from a different direction. Also in some cases also the future direction is wrongly predicted.

The reason for this lies in the difficulty of finding the lane following waypoint if two or more possible WP successors points are available. The distinction is currently made based on the angle to the last sampled WP. Small angles are assumed to be lane following, angles above a threshold are labeled as right or left turns, depending on the sign of the angle. If there are only WP marked as turns, the one with the smaller angle is picked.

## 3.4 Visualization

The calculation of the simulation is made with 30 fps. However, the simulation of the simulation runs with 10-15 fps due to the current *GPU* limitation. CARLA server alone needs already ~ 0.05-0.06 seconds to perform all the necessary simulation steps for the next frame. With the additional computation of the OCP solving of the MPC, the passed time between two visualization frames t_vis is actual ~0.065-0.075 seconds. This leads to visible fps output of 1/t_vis = 10-15 frames per seconds.

So the visualization is actually running in slow motion.

CHAPTER

FOUR

# MPCCARLA

## 4.1 mpcCARLA package

### 4.1.1 Subpackages

#### mpcCARLA.examples package

#### Submodules

#### mpcCARLA.examples.simpleCar_MPC2 module

#### Module contents

### 4.1.2 Submodules

### 4.1.3 mpcCARLA.curvature_mpc_controller module

### 4.1.4 mpcCARLA.model_predictive_controller module

### 4.1.5 mpcCARLA.visualization_tools module

This module contains toosl to visualize the performance of MPC controllers.

mpcCARLA.visualization_tools.**compare_errors**(*file1*, *file2*, *filename: str = None*, *txt: str = None*, *timestamp=0*)

Plotting function to compare the control performance of two dataframes. :param file1: h5-file of first measurement :param file2: h5-file of second measurement :param txt: Text displayed in legend for 2. dataframe (e.g. 'PID') :param timestamp: :return:

mpcCARLA.visualization_tools.**compare_velocity_errors**(*data1*, *data2*, *txt: str*, *filename: str = None*, *timestamp=0*)

mpcCARLA.visualization_tools.**plot_frenet_states**(*filename: str*, *time=None*)

Plotting the frenet states. :param filename: h5-filename :param time: timestamp for zoom-in. :return:

mpcCARLA.visualization_tools.**plot_prediction_at_time**(*filename: str*, *time: float*)

mpcCARLA.visualization_tools.**read_log_informations**(*filename: str*)

Reading the h5-logfile. :param filename: h5-filename :return: panda.Dataframe

### 4.1.6 mpcCARLA.waypoint_utilities module

### 4.1.7 Module contents

CHAPTER

FIVE

# INDICES AND TABLES

- genindex
- modindex
- search

# BIBLIOGRAPHY

[Bruedigam2020] Brüdigam, T., Olbrich, M., Wollherr, D., and Leibold, M., "Stochastic Model Predictive Control with a Safety Guarantee for Automated Driving", link, 2020.

# PYTHON MODULE INDEX

## m

# INDEX