SUMO User Conference 2024

TIBconferencesession

TIB-OP will set DOI with \TIBdoi

Submitted: 2024-02-22

Sumonity: Bridging SUMO and Unity for Enhanced Traffic Simulation Experiences

Mathias Pechinger[1], and Johannes Lindner[1]

[1]Technical University of Munich, Chair of Traffic Engineering and Control, Germany

*Correspondence: Mathias Pechinger, mathias.pechinger@tum.de

Abstract: This paper presents ”Sumonity,” an interface that bridges SUMO (Simulation of Urban MObility) and Unity, combining SUMO's robust traffic modeling capabilities with Unity's advanced graphical and physical engine, enhancing realism in traffic simulations. The study explores Sumonity's development and implementation, showcasing its capabilities. The interface offers a significant improvement in simulation fidelity by adopting a pure pursuit control approach within Unity for simulating each traffic agent. This methodological shift allows for more granular control over individual vehicle behaviors, aligning with autonomous and common vehicle dynamics. The paper also discusses the broader implications of Sumonity for future research in this field.

Keywords: Co-Simulation, Unity, Vehicle Control, Traffic Simulation

1 Introduction

The complexity of urban traffic systems necessitates the use of sophisticated simulation tools. Simulation of Urban MObility (SUMO) has established itself as a significant open-source platform for large-scale traffic simulations. Its microscopic simulation capabilities are utilized in various articles, looking at location privacy preservation **Hayat2023**, modeling dynamic vanpooling **Qurashi2020ModelingAD** or platooning of autonomous vehicles **5625277**. In addition to papers facilitating the software, there is also a lot of work on calibrating SUMO simulations **Keler˙Kunz˙Amini˙Bogenberger˙2023**, **9606704**, **Harth˙Langer˙Bogenberger˙2022**, **Yadavilli2020**.

Considering the capabilities of microscopic traffic simulations, the limitations within the vehicle models must be considered. There are several models, such as the Gipps, Newell or IDM **GIPPS1981105**, **NEWELL2002195**, **IDM˙Treiber2000**, which were compared to actual traffic data, resulting in an Root Mean Squared Percent Error (RMSPE) of 15.50% **Punzo2005** or an Mean Absolute Percentage Error (MAPE) between 12% and 17% **Brockfeld2005**, concerning the position of the vehicles. Therefore, one can assume that microscopic simulations are bound to these limitations. Current research aims to also include human factors such as task saturation into human driver models to increase their validity **VANLINT201863**.

Besides the limitations of the vehicle models in Microscopic Traffic Simulators (MTS), the method brings some limitations. MTS abstracts individual vehicles to 2D shapes responding to surrounding traffic via actions, such as accelerations and lane changes **Barcelo2011**. Today's research not only focuses on use cases for which MTS are developed initially but further extends the application fields of MTS as described below. Therefore, the approach of microscopic traffic simulation also reaches its limits.

Some research suggests implementing sub-microscopic co-simulation environments to increase simulation fidelity. In these cases, the microscopic simulation vehicles are shared with a sub-microscopic simulation environment that utilizes more accurate modeling of agents. Pechinger et al. [**Pechinger2023T˙ITS**] used such a co-simulation architecture to evaluate collective perception systems and their impact on visual occlusion. Szalay et al. [**9108745**] used this strategy in combination with a mixed reality environment to evaluate automated vehicle development. In another article, SUMO and the game engine Unity were connected to assess a special driving algorithm by looking at different levels of congestion **9646262**.

Another field of research, in which similar a co-simulation frameworks are used or could benefit of, is studying interaction of various road user groups, such as pedestrians, cyclists and manually driven or automated vehicles (AVs). For instance, **Lindner2022IV**, **Lindner2022ITSC** investigates the interaction of cyclists and an autonomous vehicle using smartphone-based Human-Machine-Interface (HMI) in urban traffic using a coupled bicycle-AV simulator. With a car driving simulator, **Denk2023** investigate the interaction of human drivers and cyclists in the safety-critical right turn situation. All of these studies are conducted with simulators that rely on sophisticated 3D-graphics for high immersion of the person under test. Additionally, realistic road user behavior is essential for valid representation of traffic scenarios in simulator experiments. The examples above highlight the necessity and research interest for tools interfacing MTS and 3D game or render engines.

Unity's prominence as a game development engine offers exceptional graphical and physics simulation capabilities, making it an ideal candidate for enhancing the visualization and interactivity of traffic simulations. Its integration with traffic simulation tools like SUMO can significantly improve the usability and realism of these simulations, offering a more dynamic and engaging user experience. Unity's straightforward integration with Virtual Reality (VR) technology leverages state-of-the-art simulation approaches. As a result, this software combination is used with dynamic driving simulators **sekeran2023investigating**.

This article introduces our interface between SUMO and Unity, called Sumonity. We combine SUMO's robust traffic modeling capabilities with Unity's advanced engine. This integration is poised to improve traffic simulations, offering enhanced realism and an intuitive user interface, making it a valuable open-source tool for researchers, urban planners, and educators. Previous work considering SUMO-Unity co-simulations are discussed in Section 2. In the next Section, the methodology of our architecture is explained, followed by a discussion of simulation results in Section 4. Finally, the article is summarized in the conclusion section.

2 Previous Work

The integration of Unity and Simulation of Urban MObility (SUMO) is discussed in recent traffic simulation research, primarily aimed at enhancing the realism and applicability of traffic models in urban planning and vehicular communication systems. A notable study by **GameTheoryBasedRampMerging** introduced a game theory-based strategy for ramp merging in mixed traffic environments. This research utilized a co-simulation environment combining Unity for visual representation and SUMO for traffic simulation, demonstrating the effectiveness of their merged strategy in handling complex traffic scenarios. While they did use a SUMO-Unity architecture they only pointed out that it enables human in the loop simulation architectures. Furthermore, **ConnectionSUMOUnity3D** explored the connection between SUMO and the Unity 3D game engine to evaluate Vehicle-to-Everything (V2X) communication systems. This study highlighted the practical application of integrating these two platforms, providing a robust simulation environment to test and develop traffic light signal assistant solutions. The work of **CouplingSUMOMotionPlanning** extended the application of this integration to the domain of automated vehicles. By coupling SUMO with the motion planning framework Common Road **Althoff2017**, the study enhanced the simulation accuracy for autonomous vehicle behaviors in traffic, offering significant insights into their operational dynamics. Additionally, the extension of the 3DCoAutoSim platform, as discussed by **3DCoAutoSimSUMOUnity**, included the simulation of pedestrian and vehicle interactions. This development is crucial for creating more comprehensive urban simulations, considering vehicular and pedestrian traffic. Lastly, **MicroscopicDriverCentricSimulator** presented a microscopic driver-centric simulator that links Unity and SUMO. This approach focuses on individual driver behaviors, contributing to a more detailed and nuanced understanding of traffic flows and individual decision-making processes.

In addition to presented work using SUMO and Unity, there are solutions utilizing Aimsun Next and Simcenter Prescan to enable co-simulation. This setup was used in previous work to analyze cyclist safety with respect to an Connected Automated Vehicle (CAV) in an urban environment **Pechinger2021bike**. Furthermore the automated vehicle simulator CARLA **DosovitskiyRCLK17** is used in other works, which are mainly focused on individual automated vehicles, whereas we want to consider larger road networks. We want to point out, that both Unity and Unreal Engine, utilized by CARLA, could be used for a sub-microsopic co-simulation. Nevertheless, looking at a faster and more convenient implementation of C# used by Unity offers advantages compared to C++, used in Unreal Engine. In addition, the asset store of Unity is much larger, speeding up the development process by using existing models for, e.g., traffic participants.

In conclusion, while the reviewed literature demonstrates significant advancements in traffic simulation through the combination of Unity and SUMO, the architecture presented in this work, introduces a enhancement to this paradigm. Unlike the conventional methodologies where SUMO predominantly drives the simulation of traffic flow, and Unity serves as a visualization conduit, this project adopts the pure pursuit control approach within Unity for simulating each traffic agent; it transcends the traditional usage of Unity as merely a graphical interface. This methodological shift towards implementing autonomous navigation imparts a granular control over individual vehicle behaviors, aligning more closely with the intricacies of autonomous and common vehicle dynamics. This is particularly advantageous for applications demanding high precision in vehicle behavior modeling, such as autonomous driving research and advanced traffic management systems. Lastly, we provide our implementation on GitHub[1] to the scientific community, encouraging joint development and utilization by research and industry.

[1]https://github.com/TUM-VT/Sumonity

3 Methodology

For our architecture, we connect SUMO and Unity using our Sumonity interface. Figure 1 shows the basic setup, including images from the simulation environment. On the left side, the microscopic SUMO simulation; on the right side, the sub-microscopic Unity simulation, and in the middle, the Sumonity Interface is shown.

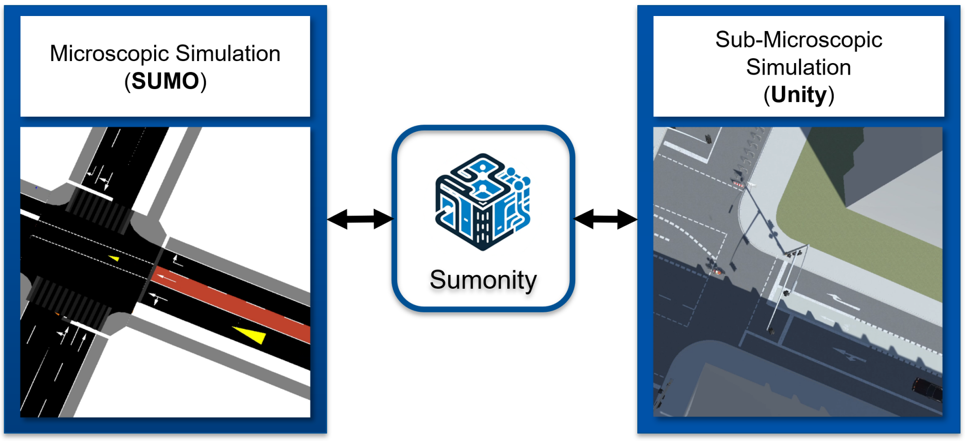

Microscopic Simulation (**SUMO**)
Sumonity
Sub-Microscopic Simulation (**Unity**)

Figure 1. *Illustration of the basic architecture of the Sumonity, SUMO/Unity interface.*

Figure 2 provides closer insights into the architecture. The microscopic simulation is connected using SUMO's Traffic Control Interface (TraCI) functionality to retrieve information in Table 1, discussed later. A significant advance compared to previous work is the integration of proper vehicle control algorithms into the co-simulation architecture. We use a Proportional–Integral–Derivative (PID) controller type for longitudinal control and a Pure Puresuit Control (PPC) system for lateral controls for each agent. Within the Python environment, we evaluate the lookahead point needed for the PPC. Details on the implementation are given in Section 3.3. On the Unity C# part of the Sumonity interface given on the right side of Figure 2, we implement the controls that utilize information from the TraCI connection. Finally, we connect the control outputs, such as target steering angle, driving torque, and velocity, to physics-based/rigid body assets.

Not presented in this paper, but also included in the interface, is the direct representation of SUMO vehicles in Unity. The position and rotation information is directly transferred to Unity objects. This approach might have benefits in performance and usability, but lacks realistic vehicle physics.

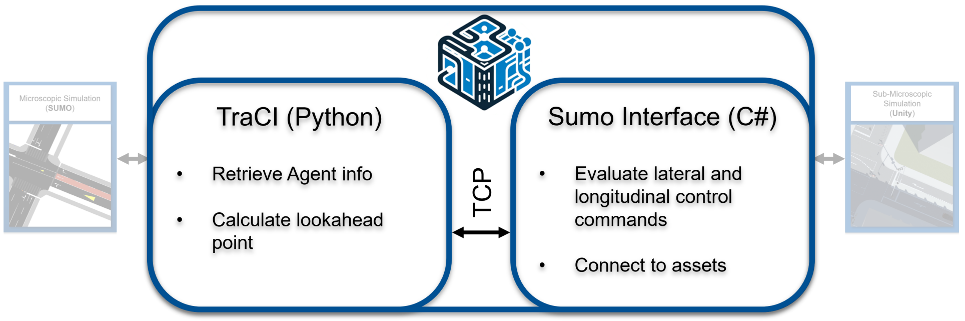

Microscopic Simulation (SUMO)
TraCI (Python)
Retrieve Agent info
Calculate lookahead point
TCP
Sumo Interface (C#)
Evaluate lateral and longitudinal control commands
Connect to assets
Sub-Microscopic Simulation (Unity)

Figure 2. *Illustration of the detailed architecture of the Sumonity, SUMO/Unity interface.*

3.1 Map Setup

Sumonity depends on aligning the SUMO environment and Unity. Therefore, we have established a strategy given in Figure 3. The process begins with georeferenced CityGML objects and satellite images to create the digital twin for our simulations. This data is used within Mathworks Roadrunner[2], which can generate an OpenDRIVE map and a 3D map representation as Filmbox. Finally, we load the corresponding files in Unity and SUMO

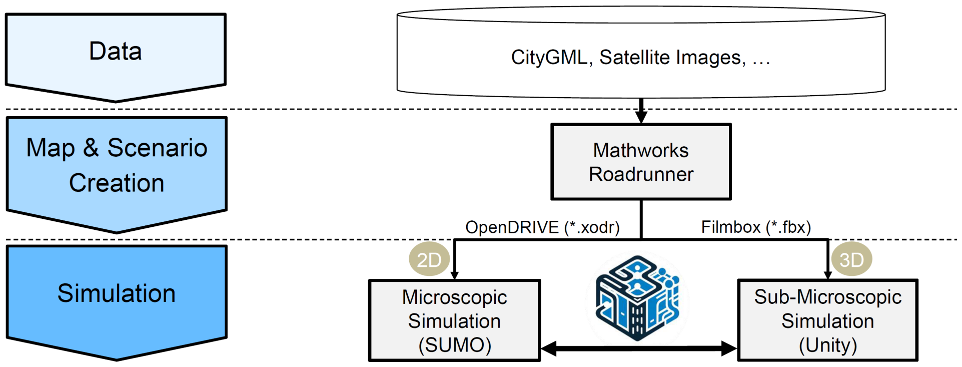


Figure 3. *Details on the creation of the aligned SUMO and Unity map.*

3.2 Agent Info

Sumonity currently supports the following SUMO vehicle types, but is not limited to them as everyone can implement further types using our open-source repository:

- passenger vehicle
- bicycle
- bus
- taxi
- pedestrian

Each traffic participant inherits the properties in Table 1. The unique identifier *id*, is used to handle a dictionary for the agents. The *vehicle type* is used to classify the agent based on the above-mentioned list, e.g., a *passenger vehicle*. The *position*, *orientation*, and *speed* contain the pose information, and the signal is used for the indicator lights. In addition to previous work, we include the *look-ahead point* used by the PPC.

Table 1. *SerializableVehicle Class Properties*

Property Name	**Description**
id	Unique identifier of the vehicle
vehicle type	Type of the vehicle
position	X and Y coordinate of the vehicle's position
orientation	Rotation angle of the vehicle
speed	Speed of the vehicle
signals	Signal flags of the vehicle
look-ahead point	X and Y coordinate of the look-ahead point

[2]https://de.mathworks.com/products/roadrunner.html

3.3 Vehicle Control

Vehicle control is executed for each agent and implemented for vehicles, while the approach works for pedestrians as well. The architecture is divided into longitudinal and lateral control, as explained in the following.

3.3.1 Longitudinal Control

The longitudinal control is based on a PID system, where the control only utilizes P-Controllers given in Figure 4. We are using the reference position p_{ref} and reference velocity v_{ref} from SUMO. The *System* is evaluated using the *Realistic Car Controller Pro* from the Unity asset store, including a model for the engine, clutch, gearbox, differential, axles, and wheels to provide the actual position p_{act} and actual velocity v_{act}. Two proportional controllers P_1 and P_2 are used to generate control inputs u_1 and u_2. Adding u_1 and u_2 and dividing the result by two, u_{in} is derived, which is the torque input to our vehicle. The division by two could be removed by adjusting the gains of P_1 and P_2; we present it like this, as it is our tested version of the architecture. We chose the gains of the controllers to be $K_1 = 15$ and $K_2 = 1$. The comparatively high gain of the positional controller to the velocity controller pushes for positional error reduction while the velocity controller smoothes the generated torque input u_{in}. Although a pure position-based control approach using a full PID controller utilizing integral and derivative parts should be possible, our approach is far easier to parameterize.

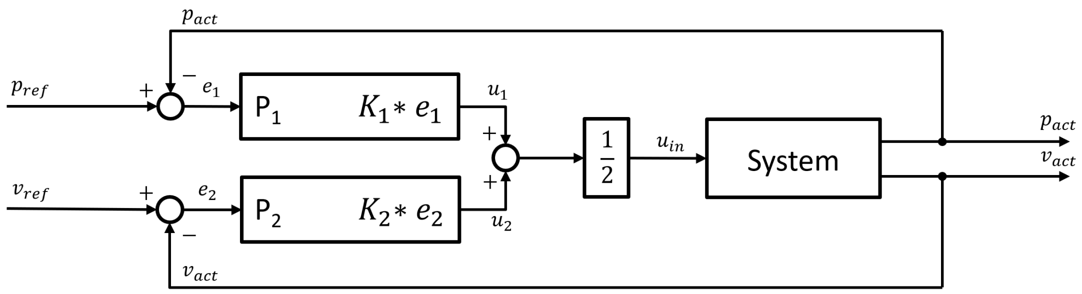


Figure 4. *Block diagram of the control loop used for longitudinal control.*

3.3.2 Lateral Control

The lateral control is based on the PPC approach, a common control strategy introduced in 1992 by **Coulter-1992-13338**. It stands out because of its computationally effective path-tracking logic. The *look-ahead point*, evaluated on the Python part, is a significant input for the control of this system. Figure 5 highlights the basics of how the PPC evaluates the reference steering angle δ_{ref}. In this case, we are referring to the steering angle as a reference due to the challenge that, depending on your vehicle model, the steering actuation may include a time constant to model real steering actuator behavior. On the left side of Figure 5, we can see a qualitative trajectory representation of the vehicle, extracted from SUMO, using TraCI.

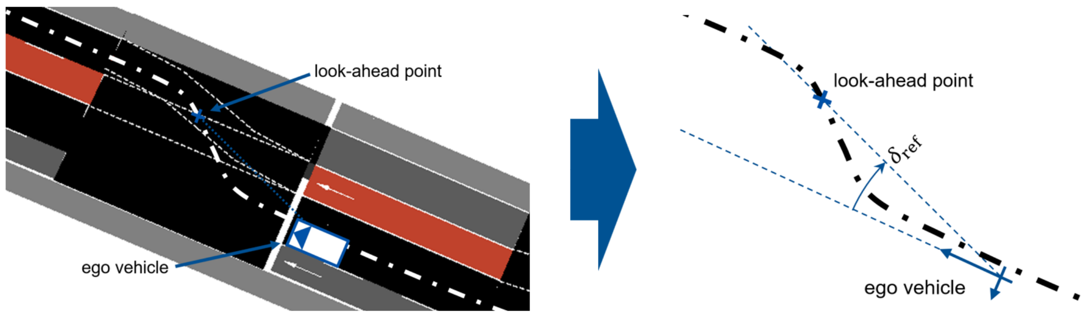


Figure 5. *Image of the evaluation of the target position, used to derive the reference steering angle δ_{ref}.*

Based on the position of the ego vehicle, the look-ahead point is derived by evaluating the Look-Ahead Distance (LAD) along the track. Based on the current velocity, this distance can be adaptive to increase control stability at higher speeds. As a result we increase the LAD for higher velocities. The LAD is derived as follows,

$$\text{LAD} = \begin{cases} \text{LAD}_{\text{min}} & \text{if } \text{v}_{\text{ref}} < \text{LAD}_{\text{min}} \\ \min(\text{v}_{\text{ref}} \cdot \text{k}_{\text{LAD}}, \text{LAD}_{\text{max}}) & \text{else} \end{cases}$$

where LAD_{min} is the minimum and LAD_{max} the maximum LAD and k_{LAD} a multiplier to account for distance adjustments depending on the vehicle's velocity. In our case, we have used $k_{LAD} = 1$, which must be greater or equal to one if it would be changed. The minimum and maximum values are discussed in the Results section. Another noteworthy feature that can be observed in Figure 5 is the offset of the ego vehicle from the reference track, as we are using the actual position of the agent and not the one from sumo, because we are controlling the actual vehicle and are using SUMO to derive the look-ahead point. The basic principle of the pure pursuit approach is given on the right side of Figure 5. We draw two lines originating from the base of the ego vehicle. One is in line with the forward-facing axis and one intersects with the center and the look-ahead point. The angle between both can be used as reference steering angle δ_{ref}.

Figure 6 shows a vehicle and a bicycle in an active Sumonity co-simulation. The picture also depicts the pure pursuit following points by red spheres on the road surface, used for debugging and validation purposes.

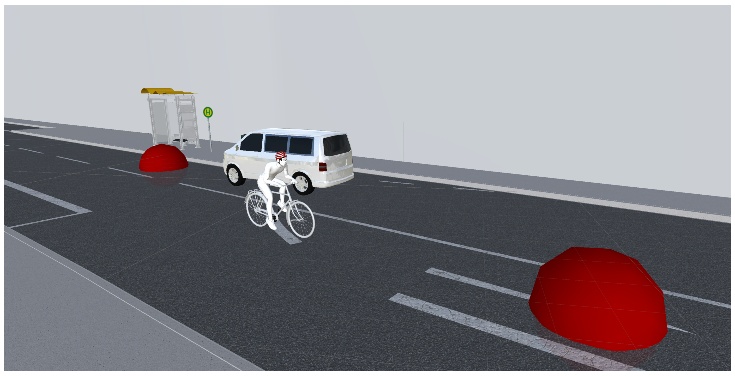


Figure 6. *3D representation showing the simulation running in unity.*

Our approach has one minor difference with reference to the standard PPC: We derive the look-ahead point using the position of the vehicle in the SUMO simulation p_{ref}

instead of the position in the Unity simulation p_{act}. Therefore, the lateral control will become unstable if the longitudinal controls fail to track the position correctly. This could be fixed by sending the actual position to the SUMO simulation or sending the extracted trajectory for the vehicle to the Unity C# component. Nevertheless, we achieve reasonable longitudinal control accuracy and neglect this concern, which is proven in Section 4.

4 Results

This Section discusses the results, showing our architecture's validity and explaining different design decisions.

Figure 7 shows the reference trajectory gathered from the agent in SUMO, whose trajectory we intend to follow. The blue solid line shows the actual trajectory driven by the vehicle in Unity, and the red dashed line shows the reference trajectory we intend to follow. We can see that we closely match the reference except for turning movements. This results from the PPC, discussed in the following.

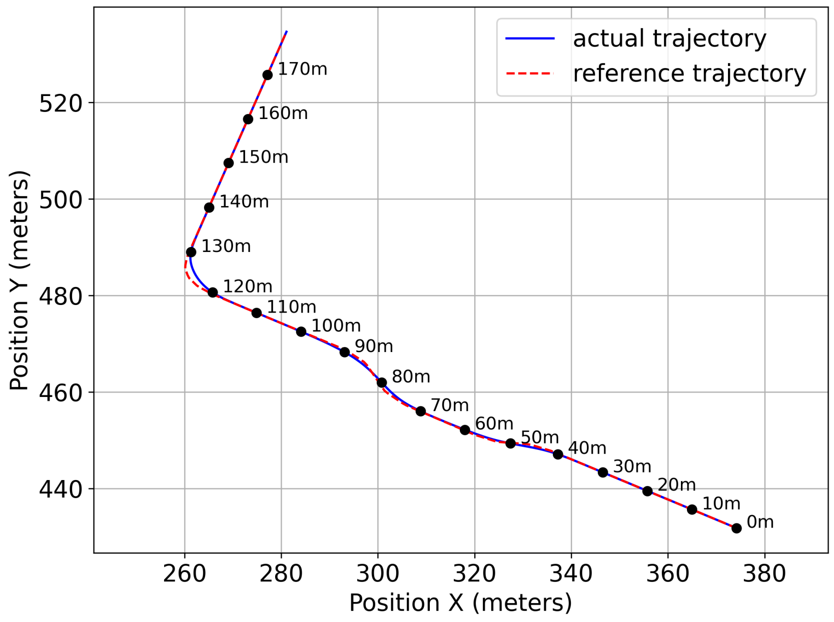


Figure 7. *Overlayed trajectory plot indicating actual and reference trajectory .*

Figure 7 includes position markers that are aligned with the x-axis of Figure 8 in order to easily understand the positional error. In the beginning, at about 10 meters, we can see the first error peak, resulting from the different acceleration behaviors of the SUMO and Unity vehicle. At 20 meters our vehicle in Unity caught up to the SUMO one. At 45 and 78 meters we can see two additional peaks that show a deviation from the reference trajectory, which are a result of two lane change maneuvers. The same effect can be observed at 128 meters when the vehicle performs a right turn. This error is related to the PPC system and, as mentioned before, is discussed later on. In addition, we can see a kind of oscillation of the error between 0 and 0.5 meters. This results from the difference in simulation time frequencies, as our SUMO simulation runs at approximately 10 Hz and Unity at 50 Hz in soft real-time conditions. The sumo update rate limitation of 10 Hz ensures real-time execution performance. Therefore, it is impossible for our controller to improve the error, as the positional resolution is about 0.5 meters. We, therefore, conclude proper position tracking using our control architecture.

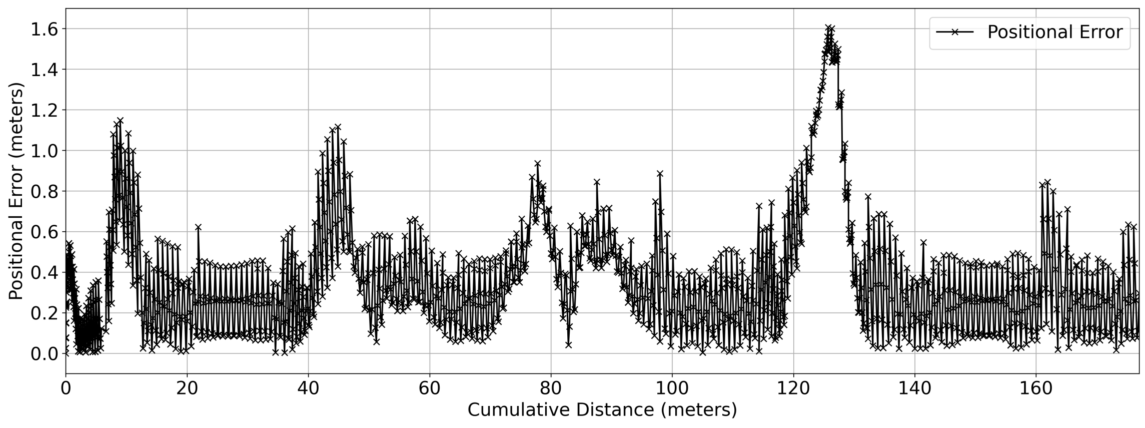


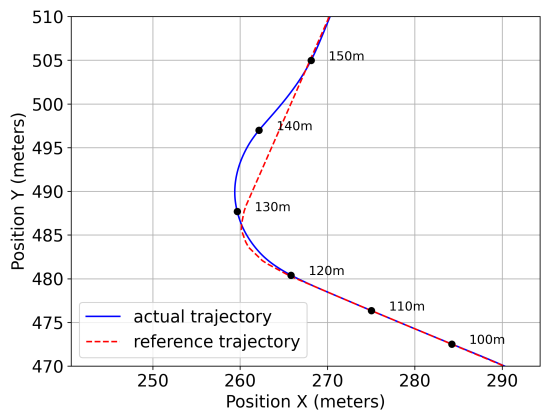
Figure 8. *Positional error comparing the actual and the reference trajectory.*

The PPC effect on lateral deviation to the reference trajectory is mentioned before and discussed here based on Figure 9. Four sub-figures can be seen, each one showing simulations varying LAD_{min} from 5 to 8 meters. Starting with Figure 9a, we can see that a low LAD results in the overshoot looking at the actual trajectory. This is not ideal; because, we would obstruct traffic on the opposing left lane, which could result in a potential accident. With a higher LAD given in Figure 9c, we can observe the simulated vehicle to slightly cut the corner. With reference to Figure 8, we can see that we deviate from the reference by 1.5 meters. Another point worth mentioning is the qualitative turning movement. The SUMO simulation does not consider actual turning movements or a drivable path for our vehicle. Nevertheless, in our case, the vehicle must comply with its physical constraints. The PPC approach resembles exactly what we need. We have to cut the corner given by the SUMO path to achieve natural driving behavior. This position tracking error is, therefore, rather beneficial than a downside of our implementation. We can improve the overall path tracking further by adjusting a gain value multiplied by the reference steering angle δ_{ref}. In our case, δ_{ref} is multiplied by steering gain k_{steer} to achieve a steering angle percentage P_{steer} between $\pm$100% for left and right steering, according to Equation 1.

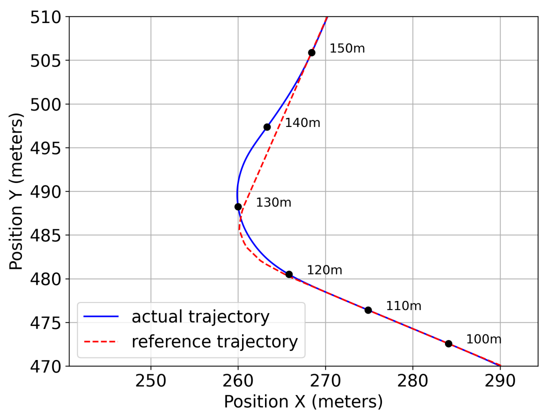
$$P_{steer} = \delta_{\text{ref}} \times k_{\text{steer}} \tag{1}$$

In our simulation, we chose $k_{steer} = 0.05$. Finally, in Figure 9d, we can see that $LAD_{min} = 8$, resulting in even more deviation from the reference track, which leads us to choose $LAD_{min} = 7$.

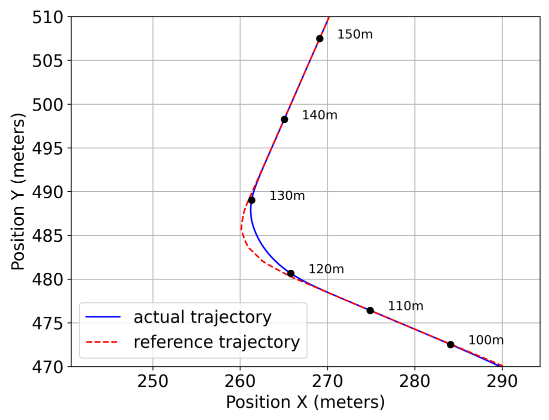
Figure 10 shows the reference steering angle with a valid range from $\pm$100%, as this is the input to the vehicle model in Unity, setting up respective maximum steering angles. Therefore, the requested steering angle at 17.5 seconds is clipped in the simulation, and the vehicle will not achieve the desired steering angle. The plot shows several ripples in a spike-shaped form. This is a result of the different simulation frequencies. As we can see from the positional error, this is an acceptable behavior. One could solve the ripples by adding a low pass filter. We do not want to use one here, as a responsive control system may not contain filters, because they would degrade the path tracking for a smoother steering behavior.



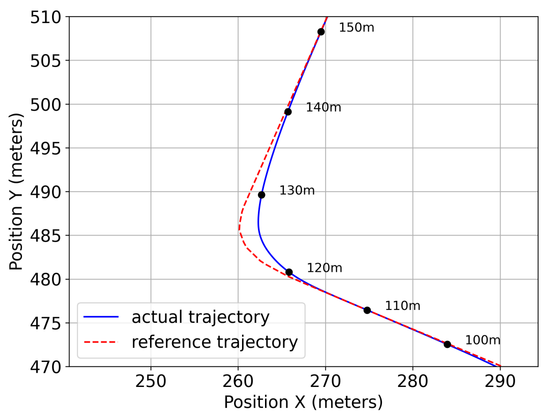
(a) $LAD_{min} = 5$



(b) $LAD_{min} = 6$



(c) $LAD_{min} = 7$



(d) $LAD_{min} = 8$

Figure 9. *Influence of the LAD on the lateral deviation for a right turn at an intersection.*

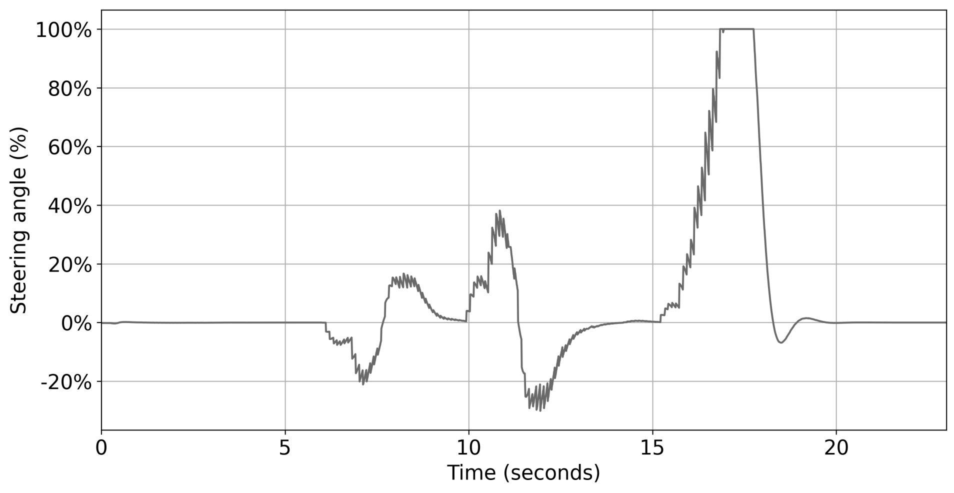


Figure 10. *Reference steering angle δ_{ref} for the vehicle running in Unity plotted over the simulation time.*

Figure 11 shows the torque input. Here, we can see a typical oscillating behavior, which is the expected output by a control algorithm. We can achieve precise position tracking using this torque input; nevertheless, this approach differs from human-driven vehicles, with their typical lift and coast driving behavior. The goal of our control is to track the position of the SUMO vehicles closely in order to match vehicle positions between SUMO and Unity; applying lift and coast behavior would result in position difference, which we cannot accept, as vehicles from SUMO would collide in Unity.

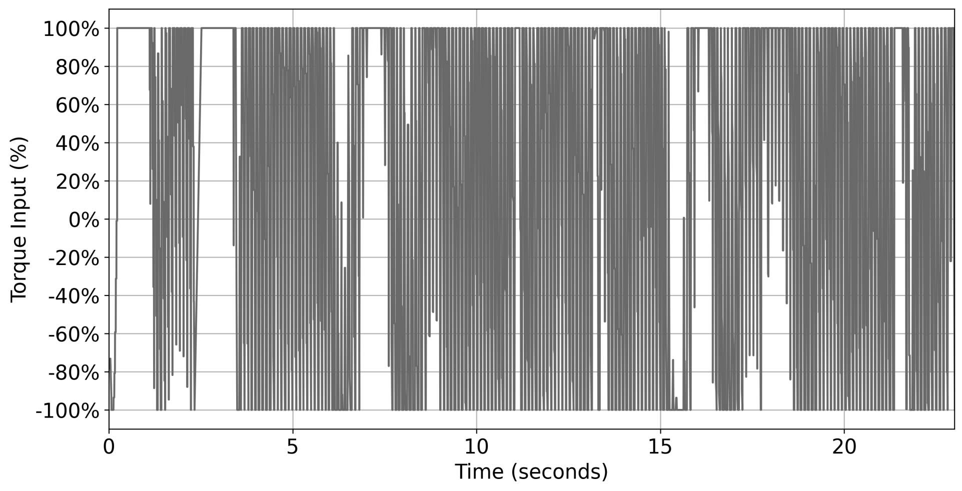


Figure 11. *Reference torque input u_{in} for the vehicle running in Unity plotted over the simulation time.*

Figure 12 shows the desired velocity from the SUMO vehicle, compared to the current velocity driven by the vehicle in Unity at the given simulation time. We can observe that the vehicle's velocity in Unity oscillates around the desired velocity. Here, we are dealing with a trade-off between position and velocity tracking. On the one hand, we want to accurately track the vehicle's velocity in SUMO. On the other hand, we want to track the position precisely. Our control is focused on the position rather than the velocity, as we do not want colliding vehicles in the Unity environment. Therefore, the deviation from the desired velocity is in an acceptable range from a qualitative standpoint looking at the given data.

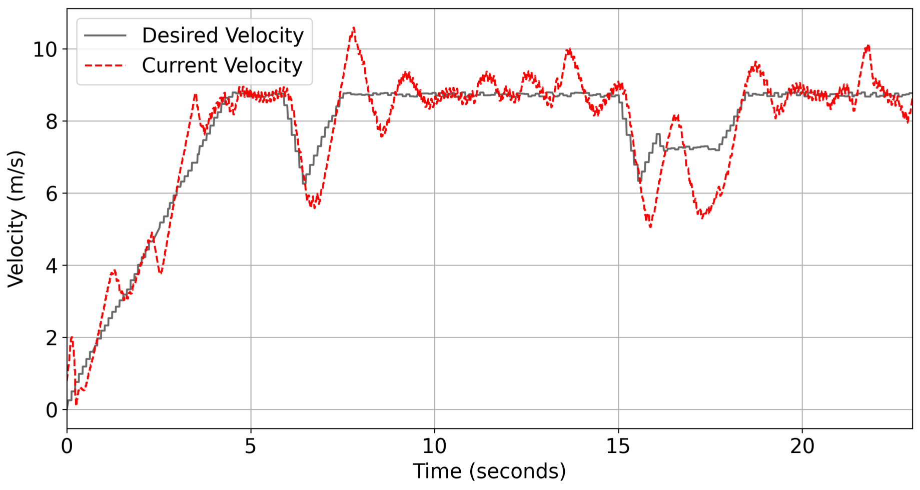


Figure 12. *The velocity tracking performance of the simulated vehicle in Unity compare to the reference from SUMO.*

5 Conclusion

In this article, we have presented our SUMO, Unity interface: Sumonity. We ran simulations and evaluated the performance of a physics-based car in Unity. We can accurately track the position of vehicles from SUMO. Furthermore, we improve the lateral driving behavior by introducing the PPC architecture. The framework is disclosed alongside this publication, and we strongly encourage fellow researchers or industry to utilize the system, as it offers open-source advantages, such as complete insight into all parts of the system, compared to similar commercial co-simulation solutions.

Sumonity interfaces different kinds of vehicles. Nevertheless, it is limited regarding pedestrian simulations. We are testing an improved position-tracking approach for pedestrians, which still needs to be implemented in the architecture's stable branch. Furthermore, traffic light synchronization is still being implemented and will be added to a future release. From our literature review, we can see that there are several interfaces for SUMO co-simulation architecture, which need to be maintained and appropriately documented. We offer a novel state-of-the-art platform to perform SUMO co-simulations incorporating Unity, enabling 3D-simulation studies.

Underlying and related material

The implementation of the given framework is published on the GitHub page of the Chair of Traffic Engineering and Control of the Technical University of Munich:
https://github.com/TUM-VT/Sumonity

Author contributions

Mathias Pechinger and Johannes Lindner both played substantial roles in conceptualizing the work, contributing to developing the research's core ideas and objectives. Mathias Pechinger was responsible for collecting and managing data (Data Curation) and conducting the formal analysis, including the evaluation and interpretation of data. Johannes Lindner was instrumental in acquiring the necessary funding for the research project (Funding Acquisition). The investigation, primarily conducted by Pechinger, involved undertaking the empirical or experimental work required for the research. Both Pechinger and Lindner contributed to the methodology, with Lindner providing the initial idea and implementation and Pechinger making significant enhancements and further developments. Pechinger also managed and coordinated the research project's administrative aspects (Project Administration) and created visual representations of the data, including figures, tables, and diagrams (Visualization).

In software development, both Lindner and Pechinger contributed significantly; Lindner with the initial system implementation and Pechinger with substantial enhancements and additional developments. The validation of the research results, ensuring the findings were robust and reproducible, involved efforts from both authors. The original draft of the manuscript was primarily written by Mathias Pechinger, who also led most of the manuscript's writing (Writing – Original Draft). Both authors were involved in the review and editing process, refining the manuscript and ensuring its intellectual content was critically evaluated (Writing – Review & Editing).

Both authors have approved the submitted version (and any substantially modified version that involves the author's contribution to the study) and have agreed to be personally accountable for their own contributions. They commit to investigating and resolving any questions related to the accuracy or integrity of any part of the work, ensuring the resolution is documented in the literature.

Competing interests

The authors declare that they have no competing interests.

Funding

This work was funded by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) and by the European Union in the frame of NextGenerationEU within the project STADT:up (FKZ 19A22006T). In addition this work was funded by the Federal Ministry of Education and Research (BMBF), Germany, under the project MCube (Clusters4Future) DatSim.

Acknowledgements

We have used the Generative Pre-training Transformer (GPT) from SciSpace for an initial Literature review with a review done by ourselves. Overall, we did double-check all literature content created by the model.