2020 IEEE Intelligent Vehicles Symposium (IV)
October 20-23, 2020. Las Vegas, USA

TELECARLA: An Open Source Extension of the CARLA Simulator for Teleoperated Driving Research Using Off-the-Shelf Components

Markus Hofbauer[1], Christopher B. Kuhn[1,2], Goran Petrovic[2] and Eckehard Steinbach[1]

***Abstract*— Teledriving is a possible fallback mode to cope with failures of fully autonomous vehicles. One important requirement for teleoperated vehicles is a reliable low delay data transmission solution, which adapts to the current network conditions to provide the operator with the best possible situation awareness. Currently, there is no easily accessible solution for the evaluation of such systems and algorithms in a fully controllable environment available. To this end we propose an open source framework for teleoperated driving research using low-cost off-the-shelf components. The proposed system is an extension of the open source simulator CARLA, which is responsible for rendering the driving environment and providing reproducible scenario evaluation. As a proof of concept, we evaluated our teledriving solution against CARLA in remote and local driving scenarios. The proposed teledriving system leads to almost identical performance measurements for local and remote driving. In contrast, remote driving using CARLA's client server communication results in drastically reduced operator performance. Further, the framework provides an interface for the adaptation of the temporal resolution and target bitrate of the compressed video streams. The proposed framework reduces the required setup effort for teleoperated driving research in academia and industry.**

I. INTRODUCTION

The teleoperator's role in autonomous driving is to take over remote control in case the system fails or encounters unknown traffic situations it cannot handle on its own [1]. The operator has to be fully aware of the remote driving situation to control the system safely. To this end, a description of the current driving situation has to be transmitted from the vehicle to the operator's workspace introducing as little delay as possible. This description includes sensor signals captured by the vehicle, such as visual (camera, range sensors) or positional sensors (GPS). The combination of these sensor signals results in a huge amount of data, which might exceed the current transmission capacity. Therefore, adaptation mechanisms are needed to match the available transmission resources and provide the operator with the best possible situation awareness.

Research and development of such adaptation methods or teleoperated driving in general requires an evaluation environment, an actor which will be controlled remotely, an operator workstation and a low delay streaming system. Previous works have already proposed working solutions, but rely on real vehicles or expensive closed source vehicle simulations [2]–[5]. Additionally, the focus is often on the vehicle setup or solutions to improve the operator view, but few details on the streaming system are provided. For example, [6] proposed a teleoperation vehicle setup using off-the-shelf components, but still relies on real vehicles which might not always be required. Especially for improvements on the operator side or for the transmission part, the expensive vehicle could be replaced with a simulation. Both academia and industry use different simulators and implement their own streaming systems which are not directly available. To the best of our knowledge, there is currently no out-of-the-box solution for teleoperated driving research in a simulated environment available.

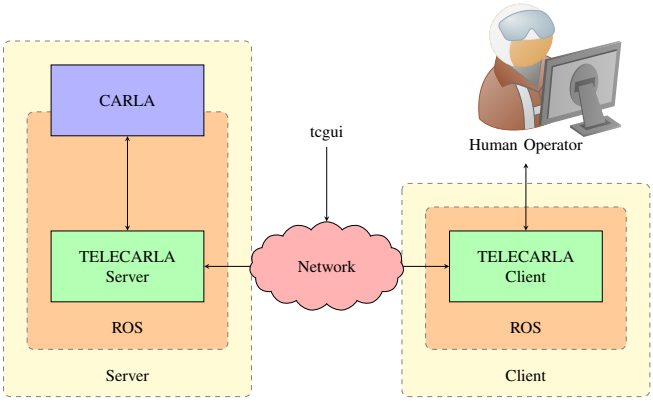


Fig. 1: Overview of the proposed teleoperated driving framework (green) extending the CARLA simulator (blue) using ROS (orange).

Based on the open source vehicle simulation CARLA [7], we propose TELECARLA, an open source framework for teleoperated driving research using low-cost off-the-shelf components. This comprises the hardware setup and a customizable Graphical User Interface (GUI) for the operator workspace, a low delay streaming pipeline with an interface for live stream adaption and a communication interface for exchanging control commands and status information between the server and client. A general overview of the framework is shown in Figure 1. The entire system uses the Robot Operating System (ROS) as middleware to provide common interfaces in a modular and scalable micro-service-like manner. A ROS bridge is used to connect to the CARLA simulator. The source code will be available on GitHub[1].

The rest of the paper is organized as follows. Section II summarizes related work in this area. We present our hard-

[1]Department of Electrical and Computer Engineering, Technical University of Munich, 80333 Munich, Germany {markus.hofbauer, christopher.kuhn, eckehard.steinbach}@tum.de
[2]BMW Group, Knorrstraße 147, 80788 Munich, Germany {christopher.kuhn, goran.petrovic}@bmw.de

[1]https://github.com/hofbi/telecarla

ware setup in Section III and proposed software solutions in Section IV. Section V describes our evaluation setup and the results are shown in Section VI. Section VII concludes the paper.

II. RELATED WORK

Most teleoperation simulation systems are based on vehicle simulators with custom extensions for the streaming part. Hosseini et al. [2], [3] use SILAB and connect via UDP to the operator workstation. SILAB [8] is a closed source vehicle simulation software that can be purchased together with the related hardware components to perform realistic vehicle driving simulation. A major drawback of this system is the lack of support for custom modifications. Camera data, for instance, has to be acquired using a screen grabbing software.

With increasing popularity of autonomous driving, but without the availability of suitable simulations, solutions based on video games were implemented and tested. Modern video games such as Grand Theft Auto V already provide photo-realistic graphics and physics simulation. A publicly available extension called DeepGTAV [9] supports autonomous driving scenarios. Similar to vehicle simulator solutions such as SILAB, video games have the drawback of poor extensibility and customization of the core part.

Solutions such as Deepdrive [10] provide open source simulations for autonomous driving, but with just a small set of features. Microsoft developed its own simulator for autonomous driving and drone simulations, called AirSim [11]. Its main focus is on dataset generation for machine learning. With its detailed physics simulation, it is well suited for aviation simulations [12]. The Advanced Platform Lab at the LG Electronics America R&D Center, formerly the LG Silicon Valley Lab, developed the LGSVL Simulator [13] with full integration to popular open source platforms Apollo [14] and Autoware [15]. The simulation used by Autoware is mainly based on Gazebo [16], which is one of the most popular simulation frameworks for robots working with ROS. Gazebo itself provides the extension CitySim [17], which launches a city road-environment into the empty Gazebo world. Gazebo provides plenty customization possibilities, which increases the simulation effort with many traffic participants.

CARLA is an open source vehicle simulation with special focus on autonomous driving. It provides a wide range of sensor suites, already pre-trained autonomous driving models, a flexible API and supports common map standards such as OpenDrive [18]. The physics simulation and rendering is performed by the Unreal Engine [19]. Due to its flexibility, extensibility and the active CARLA community, this simulator was selected as the basis of our work.

The suggested streaming solution is a customizable open source extension to the CARLA simulator using low cost off-the-shelf hardware components.

III. HARDWARE SETUP

The basic hardware setup for controlling the vehicle consists of a gaming seat and the Logitech G29 gaming wheel and pedals with a price of less than 500€. Additionally, we use three 24″ displays and a standard PC, extended with a NVIDIA GTX 1050 Ti graphics card for running the CARLA simulator. Figure 2 visualizes the complete setup.

Fig. 2: Teleoperation workstation running the CARLA simulator with wheel and pedals control, gaming seat and 3 displays.

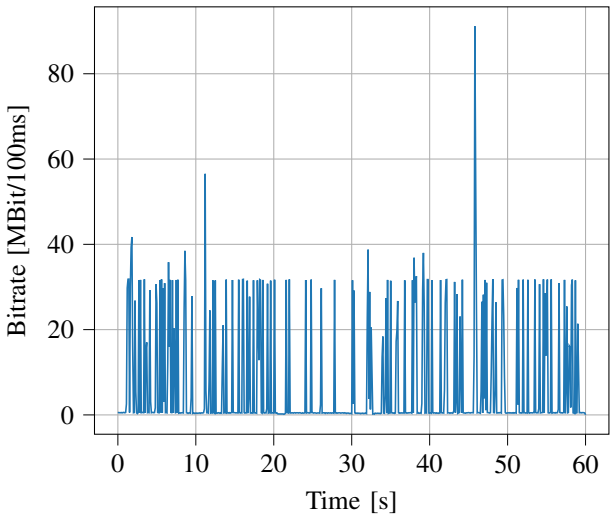


Fig. 3: Network trace of CARLA's raw data transmission.

We applied small changes to the simulator's client to make the system controllable via pedals and the wheel, and tuned some parameters for the wheel and pedals sensitivity to optimize the driving experience. The basic driving setup uses a single RGB camera front view with 90° horizontal field of view, which is able to switch to ground-truth semantic segmentation, optical flow and depth views.

IV. TELECARLA

CARLA's internal architecture [20] is based on a client server concept, which performs the rendering part on the server side and allows for the connection of multiple clients. The client interface provides a Python API for fast and simple usage, which is also used on the operator workstation. On the workstation setup, the client-server connection is via localhost, but the client is also able to connect to a remote server.

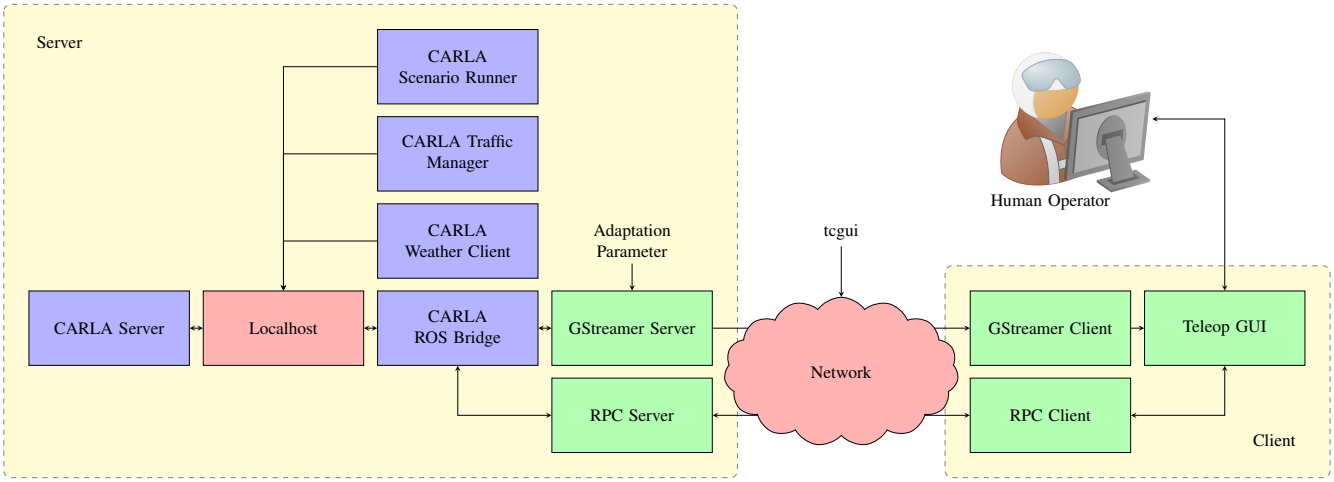


Fig. 4: CARLA simulator based streaming architecture for teleoperated driving.

Connecting to a remote server would already be a teleoperated driving simulation, but with the major drawback of CARLA's uncompressed data transmission. Figure 3 shows the bitrate of CARLA's raw data transmission. This huge amount of data leads to a very low frame rate between 3 and 4 frames per second, which is unusable for manual driving. This is because CARLA's focus is on autonomous driving and most algorithms require raw data input. As CARLA is open source, we potentially could overcome this issue by implementing compressed video transmission in CARLA ourselves. This would, however, require rebuilding the entire simulator for any change in our adaptation algorithms. Further, such modifications on the core part of CARLA would be less flexible than an external solution.

A. Proposed Concept

We use the CARLA ROS bridge [21] instead and connect it to our own streaming architecture. The proposed framework uses the Robot Operating System (ROS) [22] as the underlying middleware, which is a good choice in the area of autonomous driving [23], as this creates a modular and scalable system with common interfaces. Internally, TELE-CARLA uses GStreamer for transmitting sensor information and the Remote Procedure Call (RPC) protocol for sending control commands and status information. Figure 4 shows the proposed architecture with the available CARLA components (blue) and proposed extensions (green). Every green block is a separate ROS application, called node, which can be used multiple times in the same system. The streaming of multiple cameras can be achieved, for instance, by creating a pair of GStreamer server and clients for every available camera that should be transmitted.

Multiple CARLA clients connect to the CARLA server via localhost. Further, custom CARLA clients can be implemented depending on the application to be evaluated. The interface between the CARLA environment and our system is based on ROS enabled by the CARLA ROS bridge. To this end, our extension can be used with every other driving simulator or even real vehicles which have a ROS bridge available.

For fully controlled network conditions, we suggest the Linux built-in tool `tc-netem` via `tcgui` [24] to control the network conditions. This allows us to select the available transmission rate, transmission delay, packet loss rate, etc., for the evaluation of different adaptation algorithms.

B. Teleoperation User Interface

We implemented a custom GUI, which provides a simple configuration interface for customizing the UI according to the available sensor data. Figure 5 shows a 6 camera setup with 3 front cameras at the bottom and 3 cameras pointing backwards in the top row. Each camera view contains its own status information located in the top left corner of its segment. The status information consists of the camera name plus the spatial and temporal resolution of the input video stream. The GUI re-scales the input camera streams to fit into the desired UI layout according to the config file. In order to minimize the delay introduced by the visualization, each segment is independently updated as soon as new data arrives. This gives the opportunity to easily implement individual display overlays, such as the predictive display, free corridor etc. [2], [25] to improve the operator performance. The remaining free area on the left side of the top row contains the vehicle status information and an overview about the connected control signals for the steering wheel and keyboard. The block on the right side could contain a map or top level view with the vehicle surroundings. The complete layout of this view can be customized in a config file in a matrix like manner.

The GUI uses the Simple DirectMedia Layer (SDL2) [26] as display library, for reading the operator control commands (keyboard and steering wheel) and to apply haptic feedback to the steering wheel. If the autopilot is in control, the steering wheel is moved according to the vehicles steering angle. This allows for the evaluation of takeover scenarios from autonomous driving mode to manual driving mode.

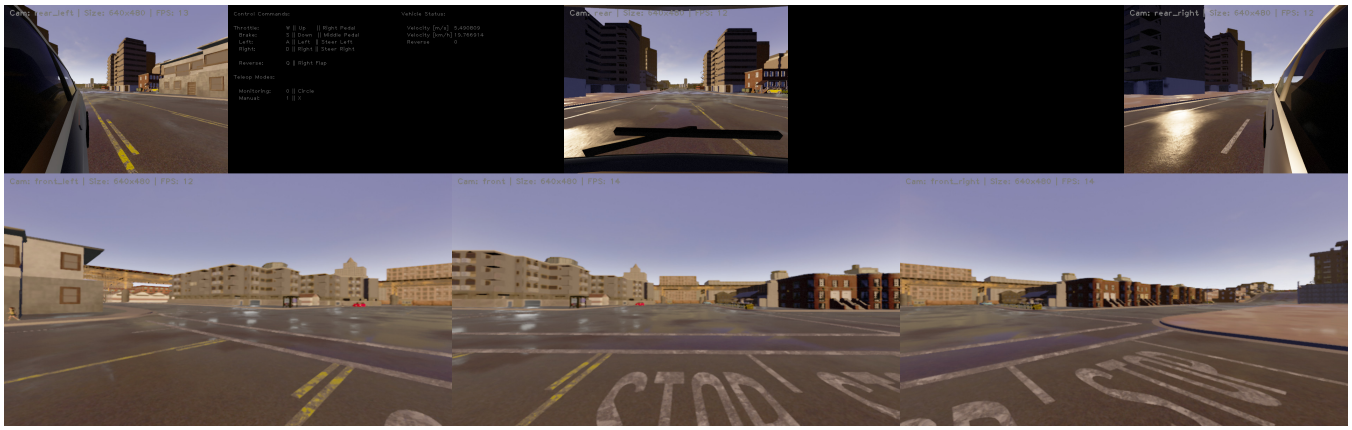
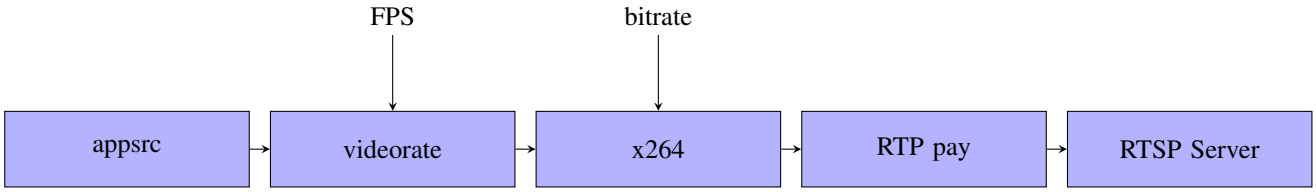
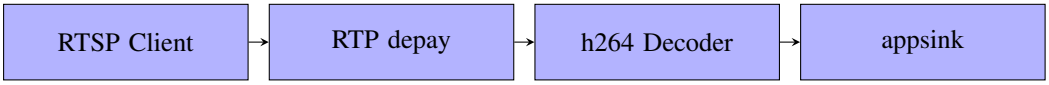


Fig. 5: Teleop GUI with a 6 camera setup and vehicle status information.



(a) Server pipeline



(b) Client pipeline

Fig. 6: GStreamer server and client pipelines.

C. Streaming System and Pipeline

The streaming system itself is based on GStreamer [27], a multi-platform, plugin-based multimedia framework. GStreamer is very powerful and flexible due to is modular structure, but finding the right configuration for a specific task might be time consuming [28].

The core streaming workflow of our system is defined by two GStreamer pipelines for server and client side. Figure 6 shows both GStreamer pipelines for the server (6a) and client (6b) conceptually. We wrapped this core pipeline into ROS to provide a flexible interface and process scalability with respect to the multi-view streaming system.

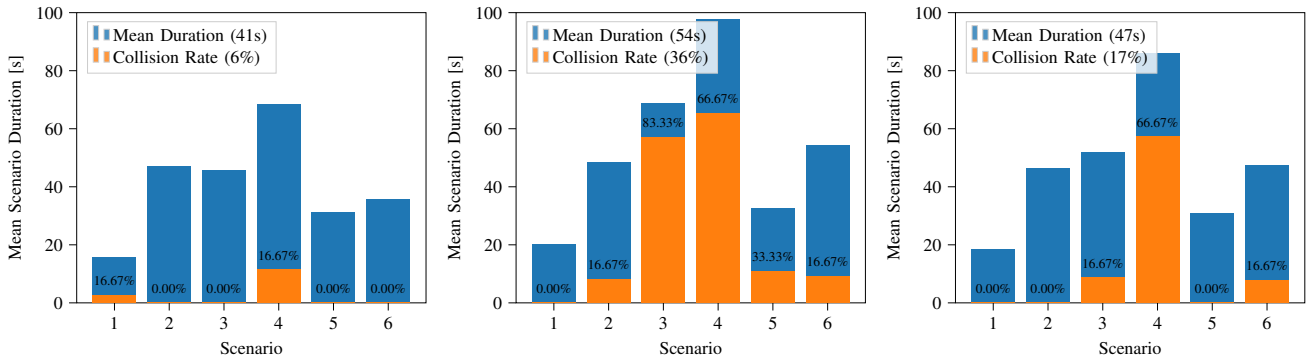
From the ROS input image topics we parse the image size and color format and provide it to the GStreamer pipeline. This enables the streaming system to take various input formats. Further, we use the ROS package *dynamic_reconfigure* [29] to provide an interface for injecting adaptation parameters into the streaming pipeline. This can be done by an automatic adaptation algorithm or by a human operator.

The server node reads the input data from ROS and feeds them into the GStreamer pipeline using the appsrc element. Appsrc is configured for *live-mode* to avoid artificial delays. The videorate plugin [30] is responsible for changing the temporal resolution (frame rate) of the video stream according to the target frame rate specified by the aforementioned input interface. The encoder plugin [31] uses the x264 codec, configured for low delay video streaming [32] and running the x264 constant bitrate control algorithm for reaching the target bitrate. We decided for x264 [33] rather than x265 [34] due to its speed advantages for low delay transmission, which is of higher priority than the best possible compression rate [35]. Thanks to GStreamer's modular pipeline concept, this can be replaced with x265 or any other encoder of preference. The target bitrate for the video encoder is again read from the dynamic reconfiguration interface. The encoded video stream is forwarded into the RTSP server element [36]. The network communication is either based on UDP or TCP on the transport layer. The default option is set to UDP.

On the receiver side, the RTSP client feeds the data into the client pipeline where they get decoded and end in the appsink element. The appsink element publishes the decoded image data as a ROS image topic into the client system.

V. EVALUATION

In order to evaluate TELECARLA, we compare our streaming pipeline against the original CARLA client server connection. The evaluation is performed using two Dell Optiplex 9020 with an Intel Core i7-4790 8x3.6 GHz processor, a NVIDIA 1050Ti GPU and 16GB DDR3 RAM for the



(a) Driving Mode *Local* (b) Driving Mode *Carla* (c) Driving Mode *TeleCarla*

Fig. 7: Average duration (blue) and the collision rate (orange) for each driving scenario of the evaluated driving modes: *Local, Carla and TeleCarla*. We evaluated for the scenario classes *DynamicObjectCrossing* (1), *FollowLeadingVehicleWithObstacle* (2), *HeroActorTurningRightAtSignalizedJunction* (3), *ManeuverOppositeDirection* (4), *Stationaryobjectcrossing* (5), and *TurnLeftAtSignalizedJunction* (6).

Local	Local driving (no delay and no compression)
Carla	Remote driving with CARLA's remote connection (delay and no compression)
TeleCarla	Remote driving with TELECARLA streaming pipeline (delay and compression with a target bitrate of 2000 kbit/s)

TABLE I: Driving modes used for the evaluation.

Mode	**Scenario Duration**	**Collision Rate**	**Bitrate**	**Frame Rate**
Local	41 s	6 %	-	20 Hz
Carla	54 s	36 %	69 Mbit/s	3.6 Hz
TeleCarla	47 s	17 %	2000 kbit/s	20 Hz

TABLE II: Results for the driving mode evaluation.

server and 24GB DDR3 RAM for the client PC. Both were running with a 64-bit Ubuntu 18.04.3 LTS, ROS melodic and the latest GStreamer version 1.16.1. The scenarios were evaluated with CARLA version 0.9.6 using a single front camera with a resolution of 1270x720 pixel at 20 Hz.

For the evaluation of the proposed framework we use the CARLA Scenario Runner [37]. This module contains predefined traffic scenarios and allows for their execution in the CARLA simulator. We selected a set of 6 scenario classes which have to be fulfilled by the human operator. They cover different turning and avoidance tasks while interacting with other traffic participants. Each scenario class contains multiple scenarios to avoid repetitions of the same scenario which could be learned by the human operators. All participants have a regular driving license and started with 3 min of local (no delay, no compression) free driving to become familiar with the simulator and the steering wheel. Afterwards, we evaluated 3 different driving modes as listed in Table I.

Every participant had to complete a set of 6 scenarios for each driving mode (*Local, Carla, TeleCarla*), covering the same scenario classes, but different scenarios.

VI. RESULTS

In this section, we show the results of our experiments. Table II summarizes the results for the average scenario duration, collision rate, transmission rate and frame rate of the 3 evaluated driving modes. A network trace of the CARLA remote connection (*Carla* mode) transmission rate was already shown in Figure 3. Due to the raw data transmission of CARLA, this results in an average bitrate of 69 Mbit/s with an update rate of 3.6 frames per second. The compressed data transmission (*TeleCarla*) is set to a target bitrate of 2000 kbit/s which still runs at the input data update rate of 20 frames per second. The *Local* driving mode, which is used as a baseline, runs at 20 frames per second, which is defined by the CARLA ROS bridge.

The results of the scenario evaluation for the 3 driving modes are shown in Figure 7. Each bar represents the average duration to finish this scenario. The orange part shows the collision ratio of all participants for this scenario.

The *Local* mode (7a) should act as a baseline without compression and transmission delay, such as driving the car as a real driver. In comparison, the remaining two modes evaluate the remote driving case. The *TeleCarla* mode (7c) shows slightly increased scenario duration of 15 % compared to the *Local* mode. This is mainly caused by the increased duration for scenario 4. The scenario duration while running the *Carla* mode remote connection (7b) increases the average duration by 32 %. This can be explained by the reduced temporal resolution of 3.6 Hz which leads to a stop-and-go driving behavior. Even at reduced speed the remote driving is very challenging at low frame rates which is highlighted by an increased collision rate of factor 6. Nevertheless, the results of *TeleCarla* mode show a remarkable increased collision rate for scenario 4 *ManeuverOppositeDirection*. In this scenario, the ego vehicle lane is occupied and the obstacle has to be passed on the opposite lane with continuous oncoming traffic. This requires fast and accurate steering actions which was difficult for the mainly untrained participants experiencing transmission delay.

VII. CONCLUSION

The proposed framework in this work extends the open source simulator CARLA with a teleoperation mode by contributing an adaptive low-delay streaming pipeline and customizable user interface in combination with low cost off-the-shelf hardware components. The system make use of the CARLA scenario runner and the transmission control network emulator for reproducible scenario evaluation and controllable network conditions. All components are implemented as separate ROS nodes to maximize modularity and scalability.

The results show that the default client server connection of the CARLA simulator is not directly applicable for teleoperated driving research. This is mainly caused by the raw data transmission between the CARLA server and client as its focus is on autonomous driving research where most algorithms require raw data input. The proposed framework enables the operator to control the vehicle remotely at the same video frame rate as it is generated by the simulator. Further, the adaptation interface allows for the evaluation of stream adaptation algorithms while running the same scenario evaluation as reproducible performance analysis.

Currently this framework is optimized to work as an extension of the CARLA simulator, but due to the modular ROS structure, it can be used with any other simulator that provides a ROS bridge. We plan to use this as a baseline development and evaluation framework for our future research on adaptive streaming of sensor information for teleoperated driving applications. The proposed solution will enable other research groups to evaluate their algorithms in an end-to-end system and supports the active CARLA community.

REFERENCES

[1] L. Kang, W. Zhao, B. Qi, and S. Banerjee, "Augmenting self-driving with remote control: Challenges and directions," in *Proceedings of the 19th International Workshop on Mobile Computing Systems & Applications*, ser. HotMobile '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 19–24. [Online]. Available: https://doi.org/10.1145/3177102.3177104

[2] A. Hosseini and M. Lienkamp, "Predictive safety based on track-before-detect for teleoperated driving through communication time delay," in *2016 IEEE Intelligent Vehicles Symposium (IV)*. Gotenburg, Sweden: IEEE, June 2016, pp. 165–172. [Online]. Available: http://ieeexplore.ieee.org/document/7535381/

[3] ——, "Enhancing telepresence during the teleoperation of road vehicles using HMD-based mixed reality," in *2016 IEEE Intelligent Vehicles Symposium (IV)*. Gotenburg, Sweden: IEEE, June 2016, pp. 1366–1373. [Online]. Available: http://ieeexplore.ieee.org/document/7535568/

[4] S. Gnatzig, F. Chucholowski, T. Tang, and M. Lienkamp, "A System Design for Teleoperated Road Vehicles," vol. 2, July 2013.

[5] S. Neumeier, N. Gay, C. Dannheim, and C. Facchi, "On the Way to Autonomous Vehicles Teleoperated Driving," p. 6, 2018.

[6] X. Shen, Z. J. Chong, S. Pendleton, G. M. James Fu, B. Qin, E. Frazzoli, and M. H. Ang, "Teleoperation of on-road vehicles via immersive telepresence using off-the-shelf components," in *Intelligent Autonomous Systems 13*, E. Menegatti, N. Michael, K. Berns, and H. Yamaguchi, Eds. Cham: Springer International Publishing, 2016, pp. 1419–1433.

[7] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An Open Urban Driving Simulator," Nov. 2017. [Online]. Available: http://arxiv.org/abs/1711.03938

[8] W. GmbH, "Driving Simulation and SILAB," accessed on: 2019-10-29. [Online]. Available: https://wivw.de/en/silab

[9] A. Ruano, "Deepgtav," accessed on: 2019-11-27. [Online]. Available: https://github.com/aitorzip/DeepGTAV

[10] C. Quiter, "Deepdrive," 2017, accessed on: 2019-12-04. [Online]. Available: https://deepdrive.io/

[11] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles," May 2017. [Online]. Available: http://arxiv.org/abs/1705.05065

[12] J. Mertens, "Generating Data to Train a Deep Neural Network End-To-End within a Simulated Environment," p. 48, 2018.

[13] A. P. Lab, "LGSVL Simulator," 2019, accessed on: 2019-12-04. [Online]. Available: https://www.lgsvlsimulator.com/

[14] Baidu, "Apollo auto," accessed on: 2019-11-29. [Online]. Available: http://apollo.auto/

[15] A. Foundation, "Autoware," accessed on: 2019-11-29. [Online]. Available: https://www.autoware.org/

[16] O. S. R. Foundation, "Gazebo - robot simulation made easy," accessed on: 2019-11-27. [Online]. Available: http://gazebosim.org/

[17] ——, "Citysim," accessed on: 2019-11-27. [Online]. Available: https://bitbucket.org/osrf/citysim/src/default/

[18] V. S. GmbH, "Opendrive," accessed on: 2019-11-27. [Online]. Available: http://www.opendrive.org/

[19] E. Games, "Unreal engine 4," accessed on: 2019-10-29. [Online]. Available: https://www.unrealengine.com

[20] CARLA-Team, "Architecture," accessed on: 2019-11-27. [Online]. Available: https://carla.readthedocs.io/en/latest/dev/build_system/

[21] ——, "Carla ros bridge," accessed on: 2019-11-27. [Online]. Available: https://github.com/carla-simulator/ros-bridge

[22] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," p. 6, 2009.

[23] A.-M. Hellmund, S. Wirges, O. S. Tas, C. Bandera, and N. O. Salscheider, "Robot operating system: A modular software framework for automated driving," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. Rio de Janeiro, Brazil: IEEE, Nov. 2016, pp. 1564–1570. [Online]. Available: http://ieeexplore.ieee.org/document/7795766/

[24] C. of Communication Networks, "tcgui," accessed on: 2019-11-27. [Online]. Available: https://github.com/tum-lkn/tcgui

[25] T. L. T. Chen, "Methods for Improving the Control of Teleoperated Vehicles," Ph.D. dissertation, 2014.

[26] S. Lantinga, "Simple directmedia layer (sdl)," accessed on: 2019-11-27. [Online]. Available: https://www.libsdl.org/

[27] GStreamer-Team, "Gstreamer," accessed on: 2019-11-27. [Online]. Available: https://gstreamer.freedesktop.org/

[28] S. Govindarajan, T. Bernatin, and P. Somani, "H. 264 encoder using Gstreamer," Mar. 2015, pp. 1–4.

[29] ROS-Team, "Dynamic reconfigure," accessed on: 2019-11-27. [Online]. Available: http://wiki.ros.org/dynamic_reconfigure

[30] GStreamer-Team, "videorate," accessed on: 2019-11-27. [Online]. Available: https://gstreamer.freedesktop.org/documentation/videorate/index.html?gi-language=c

[31] ——, "x264enc," accessed on: 2019-11-27. [Online]. Available: https://gstreamer.freedesktop.org/documentation/x264/index.html?gi-language=c

[32] C. Bachhuber, E. Steinbach, M. Freundl, and M. Reisslein, "On the Minimization of Glass-to-Glass and Glass-to-Algorithm Delay in Video Communication," *IEEE Transactions on Multimedia*, vol. 20, no. 1, pp. 238–252, Jan. 2018. [Online]. Available: http://ieeexplore.ieee.org/document/7976319/

[33] V. Organization, "x264 - a free open-source h.264 encoder," June 2007, accessed on: 2019-12-04. [Online]. Available: http://www.videolan.org/developers/x264.html

[34] ——, "x265 - a free open-source h.265 encoder," April 2014, accessed on: 2019-12-04. [Online]. Available: http://www.videolan.org/developers/x265.html

[35] N. Barman and M. G. Martini, "H.264/MPEG-AVC, H.265/MPEG-HEVC and VP9 codec comparison for live gaming video streaming," in *2017 Ninth International Conference on Quality of Multimedia Experience (QoMEX)*. Erfurt, Germany: IEEE, May 2017, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/7965686/

[36] GStreamer-Team, "rtsp server," accessed on: 2019-11-27. [Online]. Available: https://gstreamer.freedesktop.org/documentation/gst-rtsp-server/rtsp-server.html?gi-language=c

[37] CARLA-Team, "Carla scenario runner," accessed on: 2019-11-29. [Online]. Available: https://github.com/carla-simulator/scenario_runner