Developing an Open-Source Multi-Agent Simulation Environment for Connected Autonomous Vehicles

SAFER SIM

SAFETY RESEARCH USING SIMULATION

UNIVERSITY TRANSPORTATION CENTER

Dan Negrut, PhD
Professor
Department of Mechanical Engineering
University of Wisconsin-Madison

Radu Serban, PhD
Senior Scientist
Department of Mechanical Engineering
University of Wisconsin-Madison

Developing an Open-Source Multi-Agent Simulation Environment for Connected Autonomous Vehicles

Dan Negrut, PhD
Professor
Department of Mechanical Engineering
University of Wisconsin-Madison
https://orcid.org/0000-0003-1565-2784

Radu Serban, PhD
Senior Scientist
Department of Mechanical Engineering
University of Wisconsin-Madison
https://orcid.org/0000-0002-4219-905X

Asher Elmquist
Graduate Research Assistant
Department of Mechanical Engineering
University of Wisconsin-Madison
https://orcid.org/0000-0002-0142-1865

A Report on Research Sponsored by

SAFER-SIM University Transportation Center
Federal Grant No: 69A3551747131

May 2019

Table of Contents

List of Figures

Abstract

This document contains an overview of the software infrastructure developed under the SAFER-SIM project "Developing an Open Source Multi-Agent Simulation Environment for Connected Autonomous Vehicles". We provide a description of the four foundational simulation elements – agent dynamics, sensing, communication, and virtual worlds – that anchor Synchrono, a simulation platform for connected and autonomous agents. Synchrono is built around a "server-paradigm", where two specialized servers are in charge of maintaining time, space, and communication coherence. The first server, called the *Synchrono Dynamics Server*, maintains coherence in space and time for all agents participating in a scenario. This would enable the dynamics of a collection of Synchrono clients to advance their evolution in time in a coherent fashion. Moreover, for sensing purposes, the clients will be in a position to sense each other, owing to the space coherence attribute of a scenario enabled by the dynamics server. The *Synchrono Communication Server* establishes a virtualization layer that provides a "communication space" in which messages are sent and received in a time coherent fashion. Synchrono is demonstrated in conjunction with a 30-vehicle simulation that builds off a busy Madison intersection between University Avenue and Park Street. The simulated scenario touches on several aspects of Synchrono, such as vehicle dynamics simulation, agent communication, sensor simulation, and synthetic virtual worlds.

1. Introduction

The purpose of this SAFER-SIM project was to establish the first version of an open-source software infrastructure used to test the behavior of autonomous vehicles through computer simulation. This software infrastructure is called Synchrono. As a software platform that allows rapid, low-cost, and risk-free testing of novel designs, methods, and software components, Synchrono seeks to accelerate and democratize research and development activities in the field of autonomous navigation. Synchrono is (a) heterogeneous and multi-agent, in that it supports the simulation of heterogeneous traffic scenarios involving conventional, assisted, and autonomous vehicles; (b) open platform, as it allows any client that subscribes to a standard application programming interface (API) to remotely plug into the emulator and engage in multi-participant traffic scenarios that bring together autonomous agents from different solution providers; (c) vehicle-to-vehicle (V2V) communication emulation ready, owing to its ability to simulate the V2V data exchange enabled in real-world scenarios by ad-hoc dedicated short-range communication (DSRC) protocols; and (d) open-source, as the software infrastructure will be available under a BSD3 license in a public repository for unrestricted use and redistribution.

When fully implemented, Synchrono will provide three immediate benefits. First, it will serve as a development platform for algorithms that seek to establish path planning policies for autonomous vehicles operating in heterogeneous traffic scenarios; i.e., it enables the rapid and safe testing of "work in progress" piloting computer programs (PCPs). Second, it will enable auditing of existing path planning policies by exposing connected and/or autonomous vehicles to scenarios that would be costly, time consuming, and/or dangerous to consider in real-world testing. Third, Synchrono will provide a scalable, high-throughput, virtual proving ground that exposes heterogeneous traffic complexity that would not otherwise emerge in actual single-vehicle testing conducted in controlled environments.

In order to facilitate a better understanding of the design principles that anchor Synchrono and the implementation details that glue this infrastructure together, we provide next a glossary of some of the more important terms/concepts used in this project report. In the list below, the terms in *italics* are part of this short glossary.

agent An autonomous entity, e.g., a car, robot, etc., whose evolution is not pre-canned but determined by a *PCP* at run-time as a result of mutual interactions with other agents and the environment.

Its state changes in time according to the laws of physics and its dynamics is predicted through a *Chrono* simulation.

Agent Control Unit/ACU A *Synchrono*-internal applet that controls the flow of information necessary to emulate the evolution of an *agent*.

Chrono An open-source, multi-physics simulation engine used in *Synchrono* to predict the time evolution of the *agent*s and the *Virtual World Dynamic Element*s.

Chrono System An instance of a *Chrono* simulation. It represents one instance of the simulation engine that produces the time evolution of one or multiple *agent*s and other *vwDynamicElement*s operating within a virtual world. One *Synchrono* experiment can draw on one or multiple *Chrono system*s. One *Synchrono client* employs exactly one *Chrono system*.

Communication Server A *Synchrono* server that facilitates the coordination of *X2X Communication* and that enforces time coherence. There is one such server per *Synchrono* emulation.

Dynamics Server The *Synchrono*-internal service that coordinates with *VW*-element control units and *ACU*s to maintain a time- and space-coherent simulation experience across all *Synchrono client*s. There is one such server per *Synchrono* emulation.

Google Protocol Buffers/protobuf A language-neutral library for serializing structured data. Its implementation prioritizes speed over readability.

heartbeat A constant interval of simulation time used to maintain synchronization between *Synchrono Client*s.

Piloting Control Program/PCP The "brain" of an agent. Based, on the one hand, on sensory input and localization and mapping information; and, on the other hand, on control algorithms, it produces a set of inputs that dictate the evolution of an *agent*; i.e., it pilots the agent.

state cache A class for storing external *agent* state updates and synchronizing local simulation time with external simulation times.

Synchrono A framework for the simulation of *agent*s in a *virtual world* that is distributed across a network of participating computers.

Synchrono client A process that runs a *Chrono system* for the purpose of simulating the evolution of one or more *agent*s and elements of the *virtual world*.

Transmission Control Protocol/TCP An Internet protocol that provides an error-corrected and ordered stream of bytes between two processes connected over a network.

User Datagram Protocol/UDP A minimal, packet-based Internet protocol. Uses checksum-based error detection, but does not guarantee packet ordering or delivery.

Virtual World/VW The virtual environment in which the activities of all *agent*s take place.

Virtual World Dynamic Element In the simulated virtual environment, a component whose state changes in time and whose time evolution is simulated by *Chrono*. For instance, the terrain on which an autonomous vehicle moves, smoke drifting around a building, etc. The difference between an *agent* and a *Virtual World Dynamic Element* is that the former has a *PCP*.

Virtual World State Element In the simulated virtual environment, a component that has state associated with it. This state is not controlled by the laws of physics, but rather it's set by a control program or an agent. Examples: a traffic light, a pedestrian avatar that follows a predefined (established pre-run-time) motion, etc.

Virtual World Static Element In the simulated virtual environment, a component that does not change in time. For instance, a building.

X2X Applet A *Synchrono* applet supporting the simulation of *X2X Communication* in the *Synchrono* virtual world. This communication involves both *agent*s and elements of the *VW*.

X2X Communication Any wireless communication that enables the *agent*-to-*agent*, *agent*-to-*VW* element, etc., exchange of information. The protocol that enables the communication is irrelevant. Examples of possible protocols: DSRC, 5G, blinking of lights, etc.

2. Synchrono: Foundational Simulation Elements

Simulation of connected autonomous vehicles requires the simulation of four primary components: dynamics, sensing, communication, and the virtual environment. The simulation of dynamics plays a role in constraining the agent or vehicle to reality, and propagating the desired controls into the vehicle to model the vehicle's time evolution. Simulating the sensing allows the object-detection and object-recognition algorithms to be tested in the loop with path planning and following, resulting in an end-to-end test of the vehicle control stack within the simulation. Simulated communication allows the vehicles to make use of the remaining capabilities that a physical connected vehicle would leverage. The virtual environment is coherent through the other three simulation components and is critical in dictating the test scenarios.

2.1 Agent Dynamics Simulation

Physics modeling and simulation support for Synchrono infrastructure is provided through the open-source multi-physics package Chrono [1, 2]. The core functionality of Chrono provides support for the modeling, simulation, and visualization of multibody systems, with additional capabilities offered through optional modules. These modules provide support for additional classes of problems (e.g., finite element analysis and fluid-solid interaction), support for modeling and simulation of specialized systems (such as ground vehicles and granular dynamics problems), or provide specialized parallel computing support (multi-core, GPU, and distributed) for large-scale simulations.

Built as a Chrono extension module, Chrono::Vehicle [3] is a C++ middleware library focused on the modeling, simulation, and visualization of ground vehicles. Chrono::Vehicle provides a collection of templates for various topologies of both wheeled and tracked vehicle subsystems, as well as support for modeling of rigid, flexible, and granular terrain, support for closed-loop and interactive driver models, and run-time and off-line visualization of simulation results.

Modeling of vehicle systems is done in a modular fashion, with a vehicle defined as an assembly of instances of various subsystems (suspension, steering, driveline, etc.), as illustrated in Figure 2.1. Flexibility in modeling is provided by adopting a template-based design. In Chrono::Vehicle, templates are parameterized models that define a particular implementation of a vehicle subsystem. As such, a template defines the basic modeling elements (bodies, joints, force elements), imposes the subsystem topology, prescribes the design parameters, and implements the common functionality for a given type of subsystem (e.g., suspension) particularized to a specific template (e.g., double wishbone). Modeling of wheeled

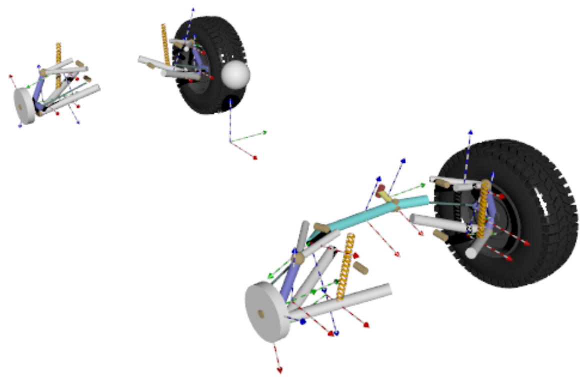

Figure 2.1: Chrono::Vehicle visualization of a wheeled vehicle with double wishbone suspensions and Pitman arm steering.

vehicles can leverage a comprehensive collection of suspension templates (double wishbone, multi-link, solid-axle, McPhearson strut, semi-trailing arm, etc.), steering templates (Pitman arm, rack-pinion), as well as templates for drivelines, anti-roll bars, wheels, and brakes. Chrono::Vehicle offers a variety of tire models and associated templates, ranging from rigid tires (with either a cylindrical or mesh contact shape), to empirical and semi-empirical models (such as Pacjeka, Fiala, and TMeasy), to fully deformable tires modeled with finite elements (which allow for detailed specification of tire geometry and material properties, as well as specification of flexible 3D tire tread patterns).

Various approaches for terrain modeling are supported. The parameterized template for rigid terrain allows specification of flat profiles, arbitrary geometry specified as a Wavefront format mesh object, or profiles constructed from height-field information provided as gray images. We provide support for deformable terrain at various degrees of accuracy and computational efficiency. An expeditious option is given by the Chrono::Vehicle extension of the Soil Contact Model (SCM) technique, which implements simple terramechanics based on the Bekker-Wong formulas. At the other extreme, the granular terrain template in Chrono::Vehicle leverages the granular dynamics support in Chrono to allow simulations of ground vehicles over granular terrain using either a compliant, or a rigid-body approach to the frictional contact problem [1]. An alternative high-fidelity approach to modeling deformable terrain is provided by templates for specifying finite element analysis (FEA) terrain patches, leveraging the support provided in the Chrono::FEA module.

For additional flexibility and to facilitate inclusion in larger simulation frameworks, Chrono::Vehicle allows formally separating various systems (the vehicle itself, powertrain, tires, terrain, driver) and provides the inter-system communication API for a co-simulation framework based on force-displacement couplings. For consistency, these systems are themselves templatized:

vehicle: the vehicle template is a collection of references to instantiations of templates for its constitutive subsystems; specific templates are provided for wheeled and tracked vehicles;

powertrain: shaft-based template using an engine model based on speed-torque curves, torque converter based on capacity factor and torque ratio curves, and transmission parameterized by an arbitrary number of forward gear ratios and a single reverse gear ratio;

driver: interactive driver model (with user inputs from keyboard for real-time simulation), file-based driver model (interpolated driver inputs as functions of time), closed-loop driver models (using PID controllers for path following, speed control, etc.).

2.2 Sensing Simulation

Along with simulating vehicle dynamic behavior, simulating the sensing that is fed into the vehicle's PCP is crucial to understand the overall autonomous behavior of the target vehicle. The development of the Synchrono sensor module is based on the importance of relaying realistic sensor data back to the control algorithm such that decisions made in the Synchrono virtual environment closely reflect those in a physical, real-world setup. To this end, sensor data generated in the virtual environment should be as similar as possible to its physical world equivalent. For example, the data generated from a virtual Light Detection and Ranging (LiDAR) as it sees a wet road with worn lane markings should encompass the appropriate distance, noise, and intensity for the road and material conditions. Highly realistic virtual sensor data allows a higher confidence that the decisions made by the control algorithm in simulation will be transferable to a real-world setup. The framework currently supports global position system (GPS) and inertial measurement unit (IMU) sensors, which include stochastic noise inherent in the data acquisition process. The noise levels are parameters in the simulation, with the virtual GPS additionally accounting for higher noise levels when satellite coverage is partially obstructed. Preliminary support for camera and LiDAR are provided through an open-source rendering engine, with higher-fidelity camera, LiDAR, and radar models the subject of further research and development.

2.3 Communication Simulation

Communication plays a key role in connected autonomous vehicles as it alleviates some of the reliance on sensing and object recognition. For example, a traffic light can broadcast its status to nearby vehicles rather than relying on the vehicles to consistently detect the color of the signal. This form of communication is known as vehicle-to-infrastructure (V2I) communication, but can be extrapolated to V2V or vehicle-to-anything (V2X) whereby a vehicle can listen and broadcast messages ranging from safety messages to traffic signal messages.

One leading protocol for V2X communication is DSRC following the SAE J2735 standard. This standard dictates the format and data for messages such as Basic Safety Message (BSM) for V2V and MAP and Signal Phase and Timing (SPaT) messages used in V2I that specify the relevant information from a traffic-light-controlled intersection. SPaT defines the current signal for all lanes in the intersection as well as the time until the signal will change. The MAP message specifies lane connections in and out of the intersection to control the flow of vehicles. An example MAP message is shown in Figure 2.2.

These messages can be generated from real-world maps using an online MAP and SPaT message creation tool [4]. Because this simulation framework can parse and send DSRC-compliant messages, real intersections can be involved in highly configurable scenarios where communication protocols, data loss, or altered DSRC messages can be changed to understand their effect on the behavior and interaction amongst connected autonomous vehicles.

2.4 Virtual Environments Simulation

The overarching virtual environment in which a simulation takes place has a significant effect on the simulation of both dynamics and sensing, and in some cases even communication. For this reason, it is essential that the virtual environment be realistic and representative of a real environment in which the scenarios of interest would occur. Real environments are littered with strange occurrences, unique

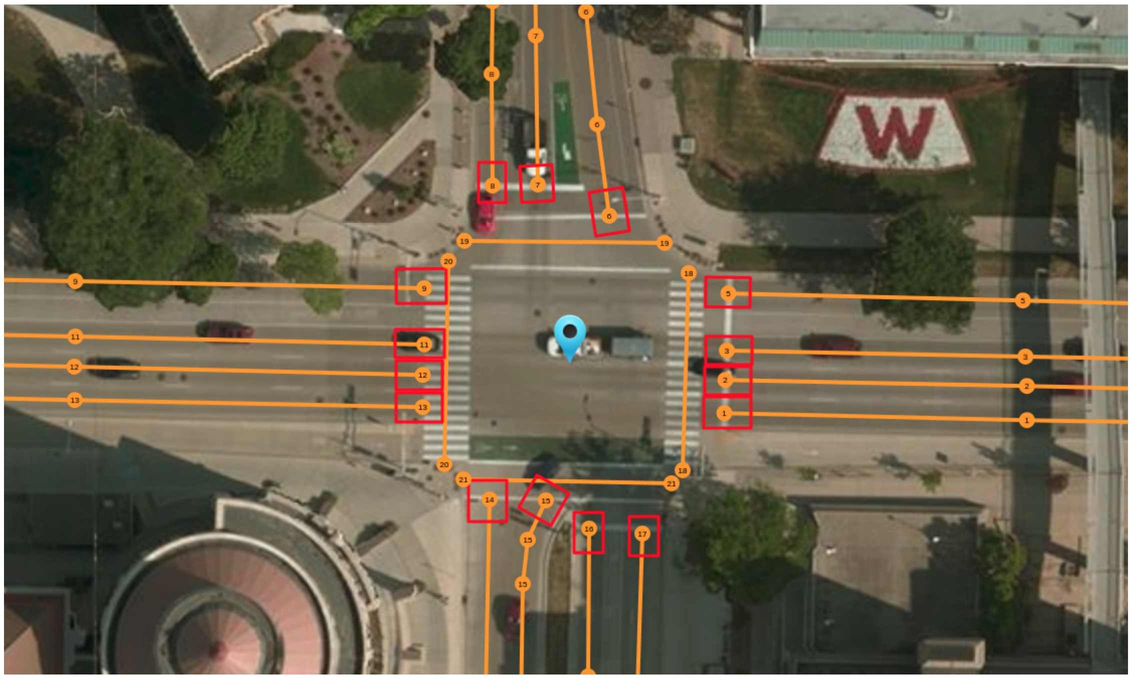

Figure 2.2: DSRC MAP message for intersection of University Avenue and Park Street in Madison, WI.

or inefficient markings, and can be highly unstructured. The virtual environment consists, broadly, of three types of objects: 1) controlled dynamic objects, 2) passive dynamic objects, and 3) static scene. Category (1) covers objects that are controlled and can react and exhibit interesting or unpredictable behavior as a result of decisions made by a vehicle. This would include animals, humans, traffic lights, bicyclists, etc. Type (2) includes those objects that can move through the scene or be pushed through the scene but will not react to any decisions. A plastic bag floating in the wind, construction cones, rain, and snow are all examples of dynamic objects. The last type, (3), is the static scene that does not change through the simulation and includes buildings, roads, and signs. Objects in categories 1 and 2 are a primary basis for understanding the reactionary behavior of an autonomous vehicle in highly dynamic and adverse scenarios and are not yet fully supported in this framework.

Although the static scene is not changing through the simulation, it still plays a critical role in the behavior of a vehicle. Each real-world intersection is unique with its own lane markings, road textures, street signs, and surrounding buildings. In an effort to generate a virtual environment that duplicates this uniqueness, a virtual replica was created of a section of Park St. in Madison, WI. This virtual replica was provided through a collaboration with Continental Mapping [5]. Figure 2.3 shows an aerial view of this virtual environment. A three-dimensional view of an intersection within this environment is shown in Figure 2.4. While still a work in progress, this process would allow the simulation of hundreds of scenarios in a replicated intersection where data could be compared to real-world experiments.

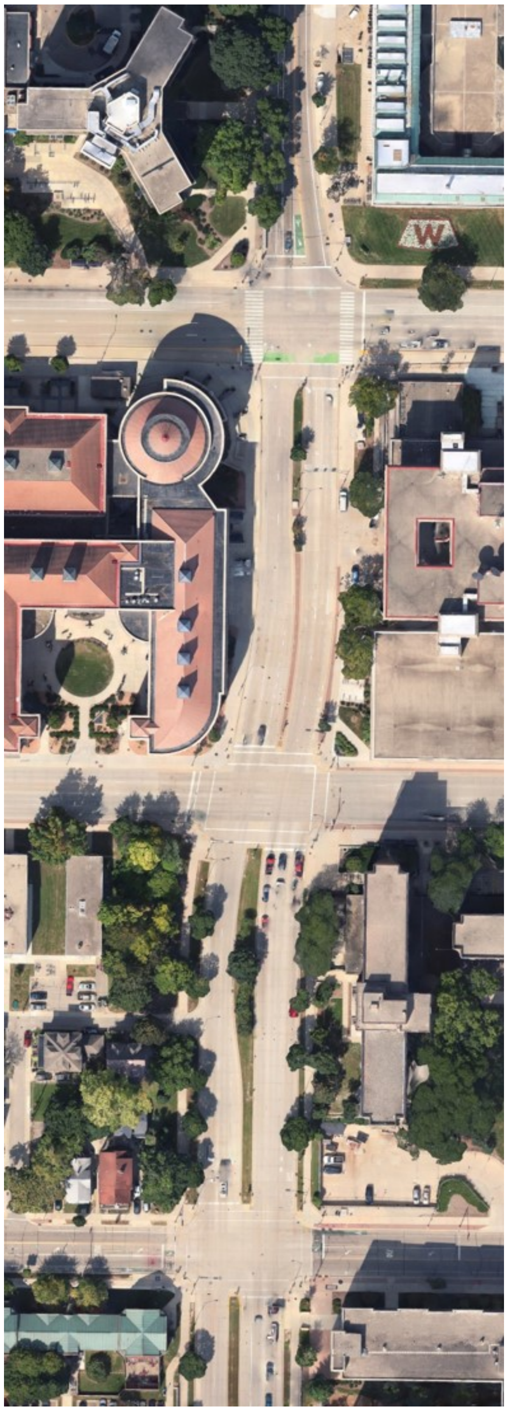

Figure 2.3: Aerial view of virtual environment reconstructed from reality. (Courtesy of Continental Mapping)

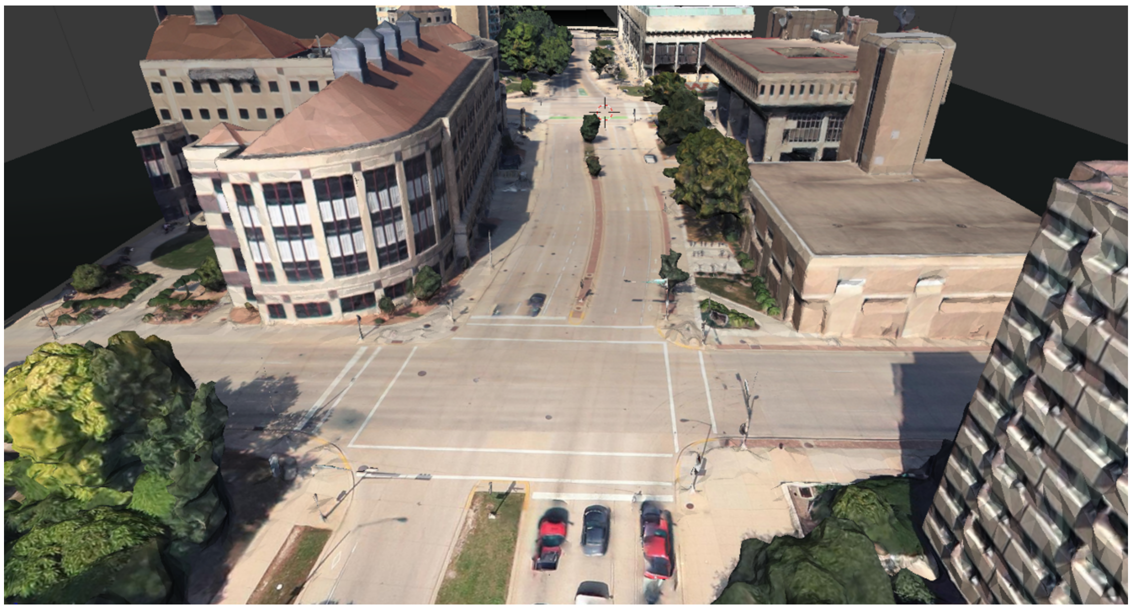

Figure 2.4: Three-dimensional view of virtual environment showing reconstructed buildings and lanes. (Courtesy of Continental Mapping)

3. Synchrono: Multi-Agent Architecture

Synchrono aims to create a framework for the simulation of agents in a virtual world that is distributed across a network of participating computers. This goal is achieved by having client simulations connect to a central server, which facilitates the distribution of state information of each agent to all other clients. The server maintains the state of the virtual world and handles clients' requests for updates. Network communication is done with the Boost.Asio library, making use of a combination of Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). Serialization of agent state information is done using protocol buffers [6]. Figure 3.1 provides a visualization of the client and server networking in the context of Synchrono as a whole. The agent control unit (ACU) interfaces with every component of the client, as well as with the server. It serves as the "connective tissue" that holds two clients together by maintaining the correct control-flow and timing between components. Dynamics and sensing simulation of all local agents is done in a system, with agent input parameters being provided by the ACU, and output eventually being fed back into the PCP as sensor data. The DSRC agent connects the agents through direct wireless communication, allowing them to share information such as BSM with one another. This connection is facilitated by the DSRC server, which routes messages between DSRC agents.

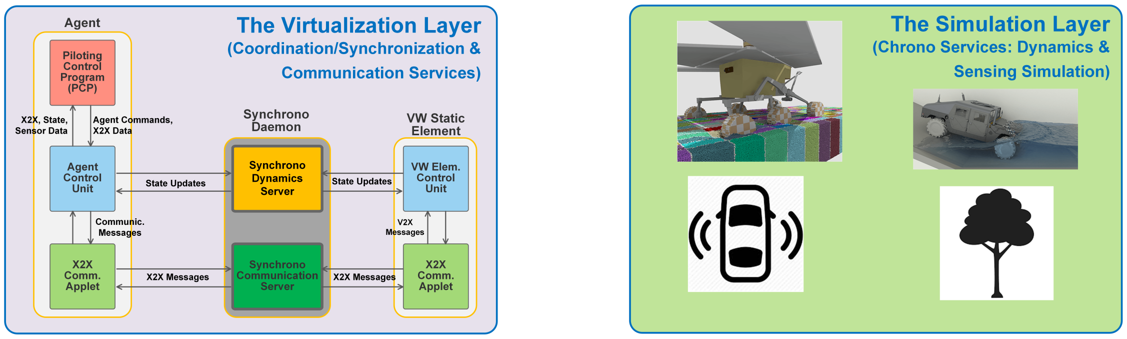

Figure 3.1: Sychrono maintains a coherent environment across a client-server model. The simulation includes a network and simulation layer with the dynamics and sensing taking place within the simulation layer of each client.

The Synchrono multi-agent server consists of three concurrent components:

- Network communication, which is carried out by the Sending and Receiving threads shown in Figure 3.3.

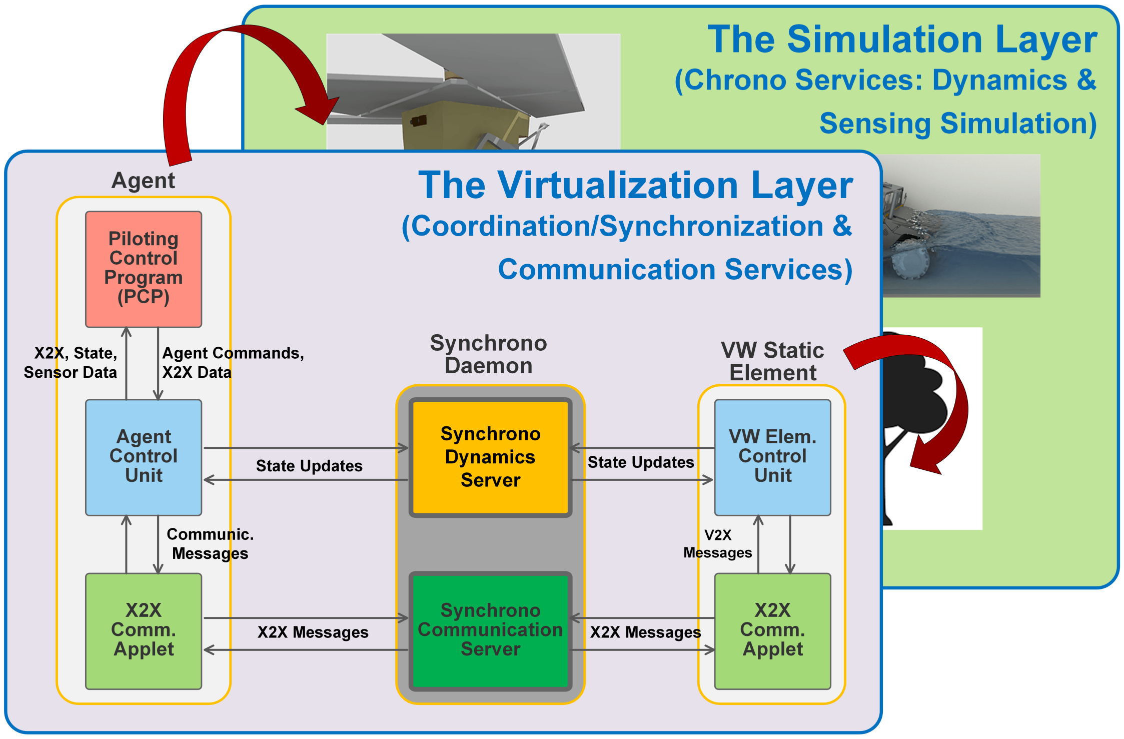

Figure 3.2: Each Synchrono client maintains a simulation layer that is updated, in part, by the dynamics server. The simulation can include controlled agents as well as static and dynamic world elements that define the simulated scenario.

- Message processing, which is done by a collection of threads labeled in Figure 3.3, and makes up the bulk of the multi-agent server's activities.
- Agent-state storage, which is done by the World Update Thread in Figure 3.3.

Network communication is done using the network handler, which will be discussed in greater detail later, and allows for the asynchronous sending and receiving of messages. The multi-agent server is designed for any number of threads to be assigned to the task of processing messages, giving it the ability to scale better as the number of available cores increases. After the messages have been processed, they are committed to the World Map as updated agent states by one thread in order to maintain thread-safety.

The Synchrono client contains at least two concurrent components:

- The main simulation loop, seen in Figure 3.4.
- The network communication threads, which are the Sending and Receiving threads in Figure 3.4.

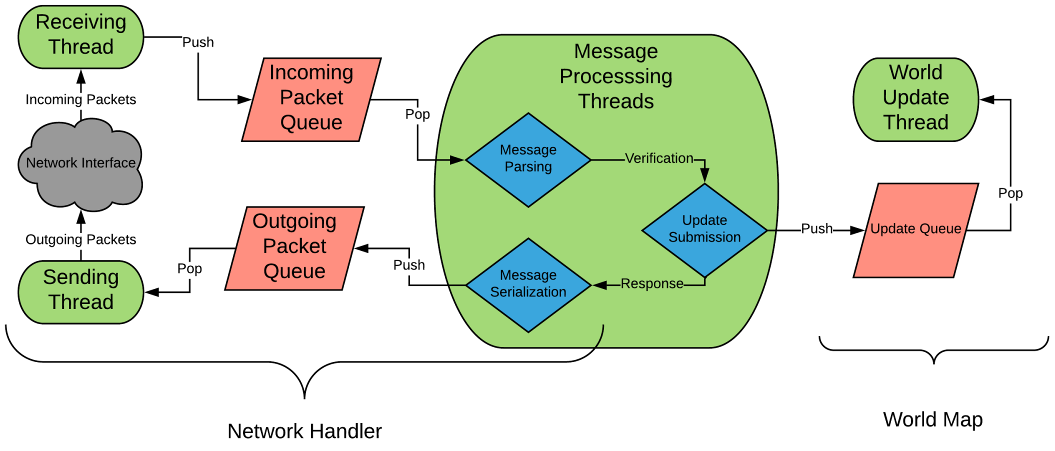

Figure 3.3: Message flow of the Synchrono, server which maintains a coherent world in time and space.

The main simulation loop is handled by Chrono, or any other physics simulation engine, and the network communication is once again handled by an instance of the network handler. In between these two components lies an additional and optional component: the state cache. This class exposes to its user the current states of external agents and handles all network handler calls internally. Before the simulation loop starts, a TCP connection is set up with the server to register a connection number with the server and receive any simulation initialization information, such as weather or agent starting position. Then, in the simulation loop, all agent updates are handed off to the network handler to be sent to the server over UDP. In other words, server messages representing the current states of all agents belonging to the client are created and packed into a buffer to be sent to the server. At this stage, the loop should also check the network handler for any new messages from the server, and update the states of all external agent representations.

3.1 Network Communication

The server uses a combination of TCP and UDP for network communication. The server accepts connections of new clients on a TCP socket. Since TCP guarantees no packet loss, it can be used to reliably exchange essential information between the client and the server. Once connected, the client is expected to send a connection request message. If the server cannot accept the connection, then a

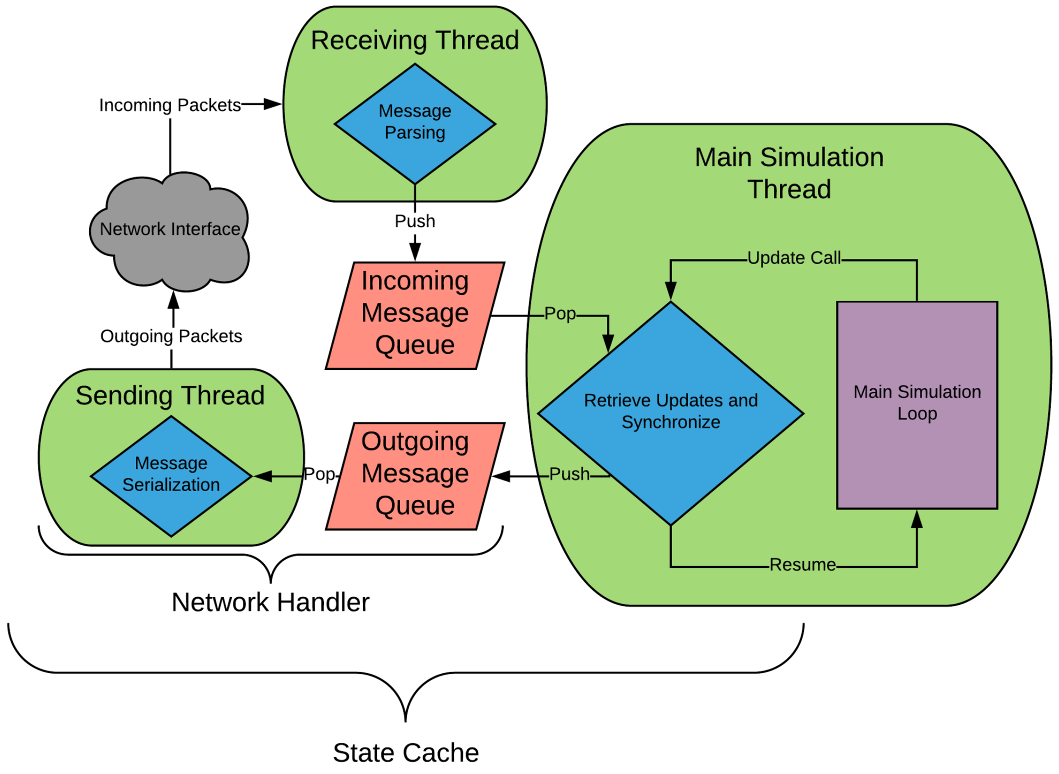

Figure 3.4: Synchrono client design shows the connection between the network and simulation layers.

decline message is sent and the socket is closed. If the server is capable of accepting the connection, it will respond with a message indicating that it has accepted the connection, and then send the client its connection number. After the client's connection number has been successfully established, the client and the server are free to continue to communicate over TCP. Information sent can include virtual world data, agent specifications, starting location, etc. Currently, the only information sent to the client is the current simulation time and the heartbeat size, which will be discussed in greater detail later.

After all TCP communication has been concluded, the connection is closed and all further communication is done over UDP. UDP is less reliable than TCP, as it does not implement any internal packet-loss countermeasures, such as acknowledgment messages. The benefit of this is that the sending and receiving of messages is more efficient. Moreover, since a more up-to-date packet would be more optimal than a re-sent stale packet, the use of UDP is able to further expedite communication. The client is expected to use UDP for all updates of agent states. The server is capable of sending three different kinds of updates to any client:

- Individual agent state updates. This can be used when agent states are updated very rarely, and can be most efficiently sent to clients individually.
- State updates of the entire virtual world. This is most useful when all agents are physically packed closely together in the virtual world, necessitating for each agent to efficiently receive updates about all of its peers.
- State updates of any subset of the virtual world. When agents are more physically spread out, it is useful to minimize the amount of information sent by only sending updates of other agents within the immediate vicinity of a receiving agent.

3.2 Message Handling

The message handling threads are used to parse incoming messages, evaluate their sanity, and submit lambda expressions that update the states of agents to a queue to be processed in serial. While parsing is done within the handler class, it is still done by the handler threads in order to keep the network communication threads from being overused.

When parsing on the server side, the “any” message provided by protocol buffers is used to prevent unnecessary parsing of agent states. Since the server is concerned only with identifying the sending client, the precise agent corresponding to a message, the system time-stamp at the time of sending, and the simulation time of the message, there is no need to parse any information regarding the agent’s internal state. In order to prevent this, an outer “AgentMessage” format is used to store the meta-data relevant to the server, along with the still-serialized state of the agent in an “any” message. This also allows the internal state to be of any format, which can be written and specified by users.

Once a message is parsed, the handler threads check that the message is of the same type as the preexisting agent state, and that the update has a newer time-stamp. These checks are done using protocol buffers’ introspective functions. These functions essentially allow the handler threads to handle any protocol buffer message without needing to know its type, so long as the message has certain fields, like time-stamp and identification number. Once these checks are done, the message is handed off to the world update thread.

3.3 World Coherence

The World Map stores all information about the server's connected clients and their associated agents. Upon the initial client connection, the connection number of the new client is added to a set of numbers in the virtual world. After the switch to UDP occurs, the connection number of incoming messages is compared with agents of this set. If an incoming message has a connection number in this set, the connection number is removed from the set, and an insertion is made to a mapping of connection numbers to client UDP endpoint information. This process is what completes the "handshake" from TCP to UDP. At this point, the client is considered to be registered in the virtual world. If further agent state updates or new agents are sent, this registration is verified by checking for its entry in the endpoint map. If a message is received that does not have a corresponding entry in either the connection number set or the endpoint map, then it is thrown out, thereby enforcing the TCP-to-UDP handshake process.

The current states of the actual agents are stored in the form of a mapping from connection number and identification number to protocol buffer message. For this reason, any agent is uniquely identified by the combination of its connection number and identification number. After a message has been verified to correspond to a valid connection number and endpoint, the user of the World Map (a message processing thread) is given a handle to an endpoint-profile structure, which stores the receiving endpoint of a client, as well as the number of agents associated with that client, and iterators to the first and last agents of the client in the agent map. The endpoint-profile enables the user to modify the agents within the World Map associated with that profile's corresponding client. In order for the agent to be updated in a thread-safe manner, the message handler threads push a lambda expression to a queue, which upon being popped is executed by the world-updating thread.

Additionally, the World Map has another thread that scans all connected agent states for sanity. If a client has not sent updates to an agent recently enough (since a few seconds prior to the check), then the server will "shoot down" that agent, and it will not be seen by other agents in the virtual world. If that client begins sending updates again, then it may be reintroduced to the World Map.

3.4 Client Synchronization

While its interface is simply a set of agent external states and a function to update an internal agent state, the primary function of the state cache is the synchronization of simulations between participating clients for the purpose of maintaining time coherence within a Synchrono emulation scenario. Time coherence is important since it allows all agents to perceive the same universal time. Unless special precautions are taken, an agent whose simulation is trivial would potentially advance quickly into the future, which leads to a "time-gap" between the current time that the simulation-light agent perceives, and the time reached by another agent whose simulation requires longer wall time.

Figure 3.5 shows an illustration of the synchronization mechanism that the state cache implements. When the handler is initialized, the heartbeat size is used by the state cache to synchronize clients. At intervals of time equal to the size of the heartbeat, the state cache periodically checks the simulation times of external agents. If there is a simulation time that is earlier than two heartbeat-lengths before the local simulation time, then the cache blocks the simulation until a sufficiently up-to-date message for that agent arrives. Alternatively, the server may perform a "shut-down," or removal, of the agent from the virtual world. In this case, the client simulation will receive a "shut-down" message in place of a normal update, and will remove the representation of that agent from the local simulation.

A schematic of how the UDP-based Synchrono dynamics server maintains time coherence is provided in Figure 3.5(C). The simplified depictions in (A) and (B) come in handy to explain the philosophy of the mult-agent server (M-AS). In (A), we show two agents ($\#i$ and $\#j$), which advance their state in time at their own pace. For instance, Agent $\#i$ can be a very simple model of a vehicle, while Agent $\#j$ is a highly accurate representation of a vehicle with complex tire models. There is a Chrono solver advancing the state of Agent $\#i$ forward in time; a second Chrono solver advances the state of Agent $\#j$. This is shown in (A). Given that Agent $\#i$ is quick in advancing its state from time c^i_{n-1} to c^i_n, this agent could move forward in simulation time, leaving Agent $\#j$ behind. Because sensing and communication take place between these two agents, they need to move forward in time in an approximately coordinated fashion. In (B), we illustrate the concepts of heartbeat, which in this example has a size of 0.005 seconds (200 Hz), and synchronization post, which is used to keep the agents approximately in sync. To explain why rough synchronization is needed, imagine that Agent $\#i$ is LiDAR-sensing Agent $\#j$. Then, the former needs to place the latter in Agent $\#i$'s environment to sense its presence. To this end, Agent $\#j$ information needs to be UDP-conveyed to Agent $\#i$. If Agent $\#i$ reaches a synchronization point and no fresh Agent $\#j$-information is available, Agent $\#i$ uses extrapolation to estimate the state based on the most recent Agent $\#j$-states it cached. With Agent $\#j$ information in hand, which might be stale, Agent $\#i$ advances to c^i_{n+1}, but will stop before crossing this synchronization post if it used cached Agent $\#j$ information. That is, referring to (C), if moving beyond synchronization post c^i_n required Agent $\#j$ cached information, Agent $\#i$ will not continue beyond c^i_{n+1} before getting updated information from Agent $\#j$. Hence, the size of the heartbeat is a measure of how many "guesses" (via caches) one agent can make about the state of a different agent that it interacts with. Small values indicate that there is tight coupling since the cached values are very recent. The trade-off for small heartbeats is increased network traffic, which adversely impacts the speed of simulation.

The communication involved in advancing the emulation along the c^i_{n-1} to c^i_n to c^i_{n+1} simulation timeline would take place as follows. The client-side manager for Agent $\#j$, which overlooks the progress of Agent $\#j$ along its simulation timeline, collects and packages state information from Agent $\#j$ right before reaching the synchronization post c^j_n (part P.1). Next, in part P.2, this state information is passed to M-AS ($\mathcal{S}^{j\to S}_n$; the superscript $j \to S$ stands for "j to Server"). The M-AS receives this information, ($\mathcal{R}^{S\leftarrow j}_n$), bundles it with the most up-to-date list of agents $\mathbb{P}(i)$ found in the proximity of Agent $\#i$, and sends it to Agent $\#i$ ($\mathcal{S}^{S\to i}_n$, P.3). In part P.4, while moving beyond the synchronization point c^i_n Agent $\#i$ uses the most recent information for Agent $\#j$. Cached information might be used if no fresh information has arrived and Agent $\#j$ is still on the list of proximity agents. Note that, currently, this component of the Synchrono is still in flux and is under active development. This inter-agent communication strategy described in Figure 3.5 is not ideal and not without shortcomings. For instance, one might have the M-AS server stay out of the way and only provide Agent $\#i$ with its most up-to-date list of proximity agents, letting the Agent $\#i$ client-side manager engage in one-to-one UDP communication with any Agent $\#j \in \mathbb{P}(i)$. One way or the other, we accomplish two important things: we prevent one agent moving fast and drifting too far into the future; and we handle, via agent-caching, the lack of fresh information.

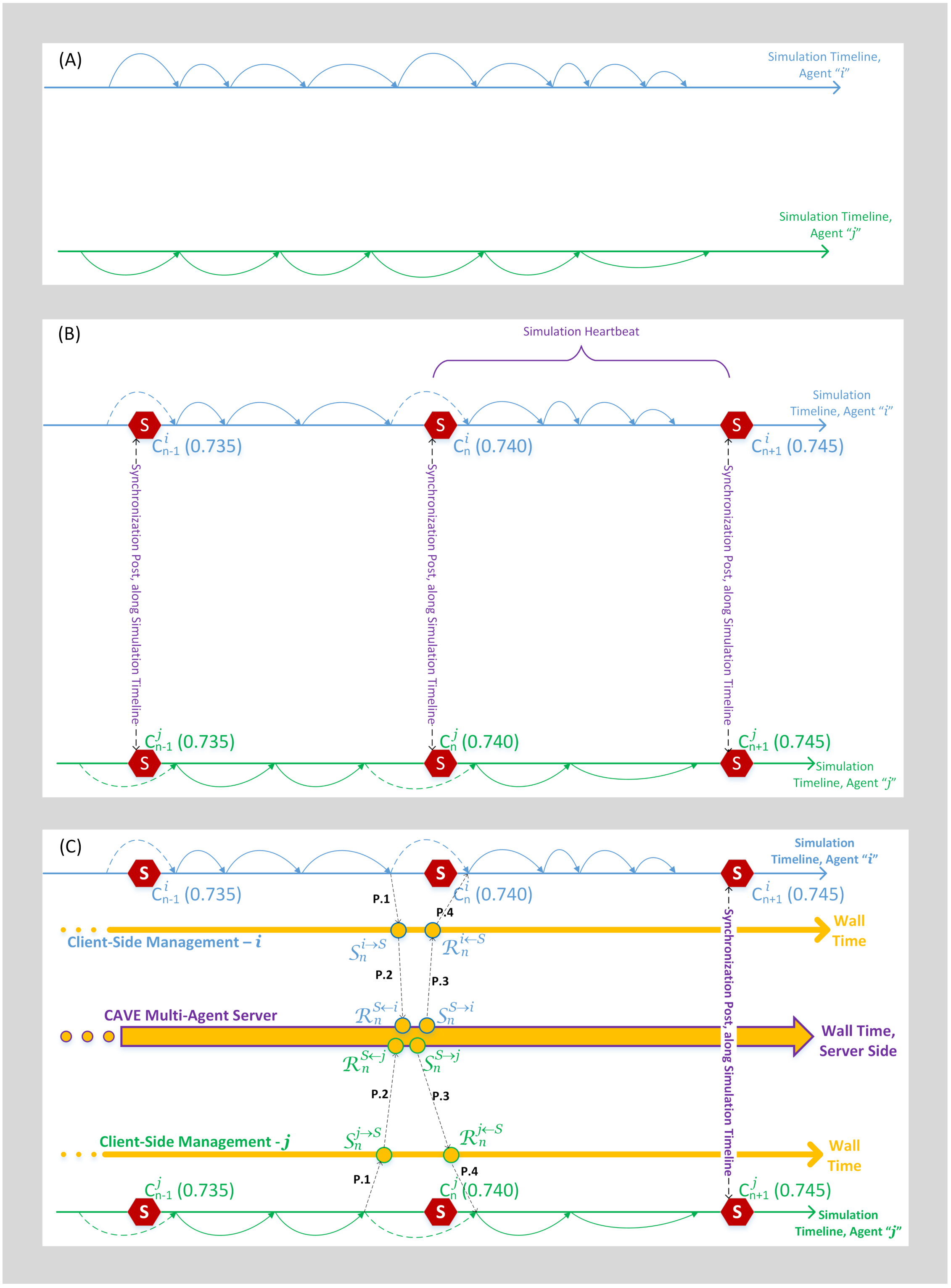

Figure 3.5: The Synchrono framework maintains a consistent simulation time whereby each client connected to the server proceeds in a synchronized manner.

4. Demonstration of Technology

To demonstrate the capabilities of the simulation framework discussed, 30 vehicles modeled as fully dynamic vehicles in Chrono are setup to navigate a computer-generated intersection that loosely represents the intersection of University Ave. and Park St. in Madison, WI. Each of the 30 vehicles is connected and autonomous, using simulated GPS and IMU to follow their desired paths and simulated LiDAR to prevent collisions with other vehicles in the environment. A virtual traffic light broadcasts signal status messages that each vehicle uses to decide on a safe time to enter the intersection. This demonstration is shown in Figures 4.1 and 4.2 with the latter giving a closer view of the vehicles in the simulation.

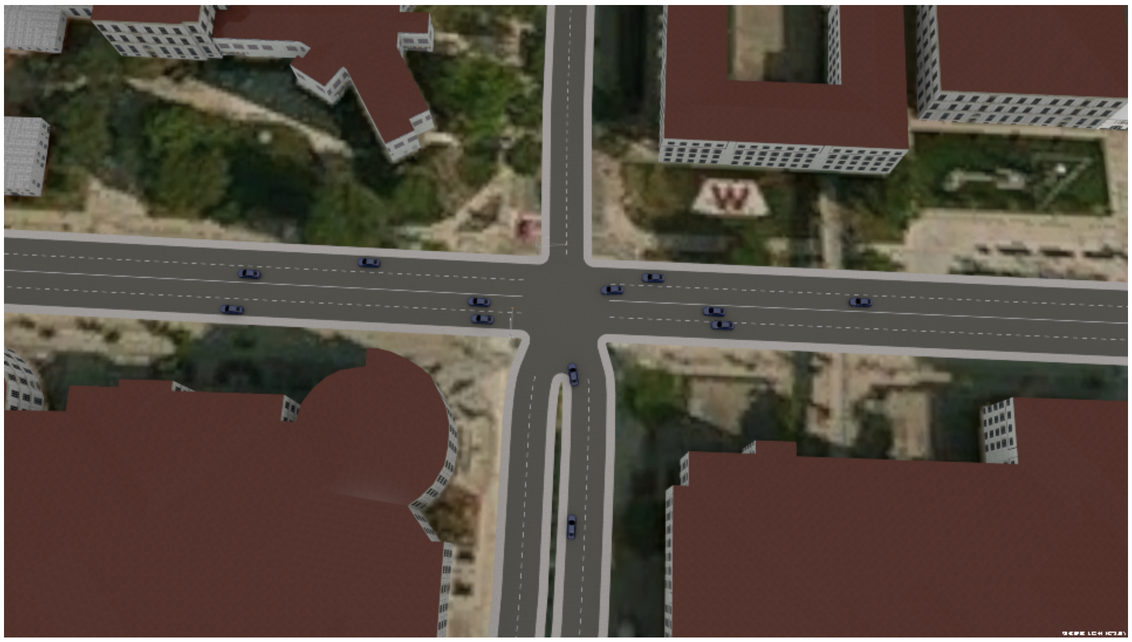

Figure 4.1: Top view of simulated intersection containing 30 connected autonomous vehicles. These vehicles use communication and simulated sensor data to safely navigate the congested roadway.

The algorithm used to control the vehicles is a simple stand-in that uses GPS and IMU data to maintain a predetermined path, as if the vehicle already knows which path should be followed on a map and which lanes should be maintained. The controller is also responsible for preventing collisions by maintaining a safer stopping distance to the vehicle ahead based on a continuous stream of LiDAR data. It is also up to the controller to listen for relevant signal status messages from the traffic light and determine if it is safe to progress along its own trajectory.

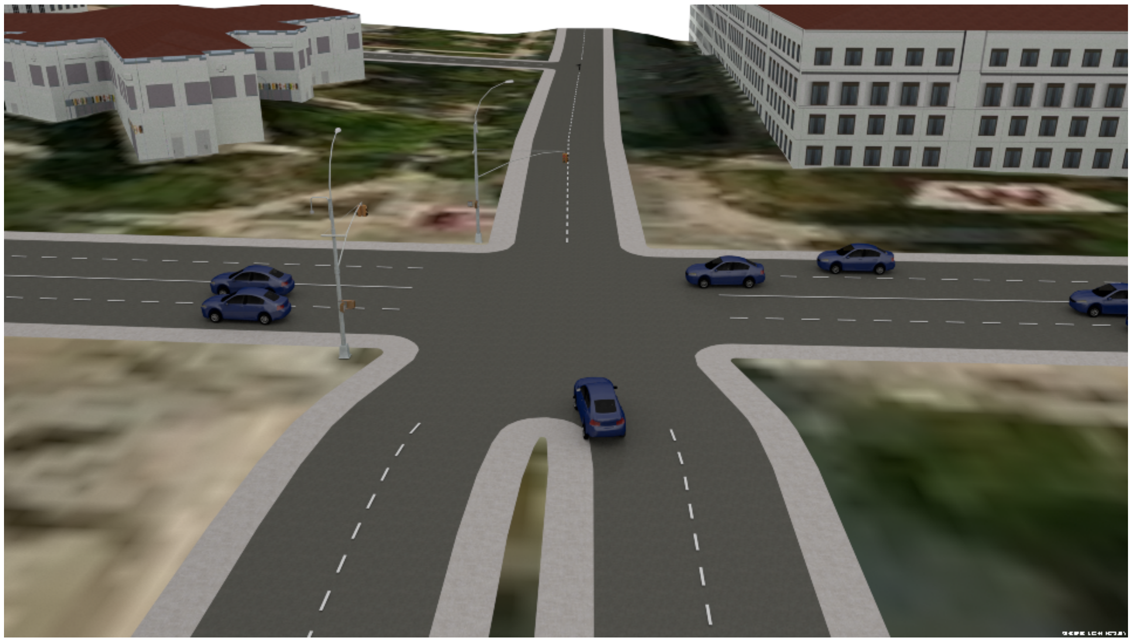

Figure 4.2: Side view of simulated intersection containing 30 connected autonomous vehicles. Each vehicle is simulated separately as a high-fidelity vehicle in Chrono but share the same virtual environment and scenario through the Synchrono multi-agent simulation framework.

In addition to an intersection showing the capability to allow multiple vehicles to interact in the same virtual environment and participate in a shared scenario, a single vehicle simulation is shown to demonstrate the virtual environment. Figure 4.3 shows a single vehicle that uses a virtual environment, courtesy of collaboration with Continental Mapping, that is a replica of reality, specifically Park St. in Madison, WI. This simulation demonstrates use of a higher-fidelity virtual environment where future scenarios can be run and compared with a real-world counterpart. This virtual environment will allow more advanced scenarios that rely on realistic roadways, more advanced sensor simulation, and increased interaction between vehicles, where inter-vehicle communication and collaboration can be studied.

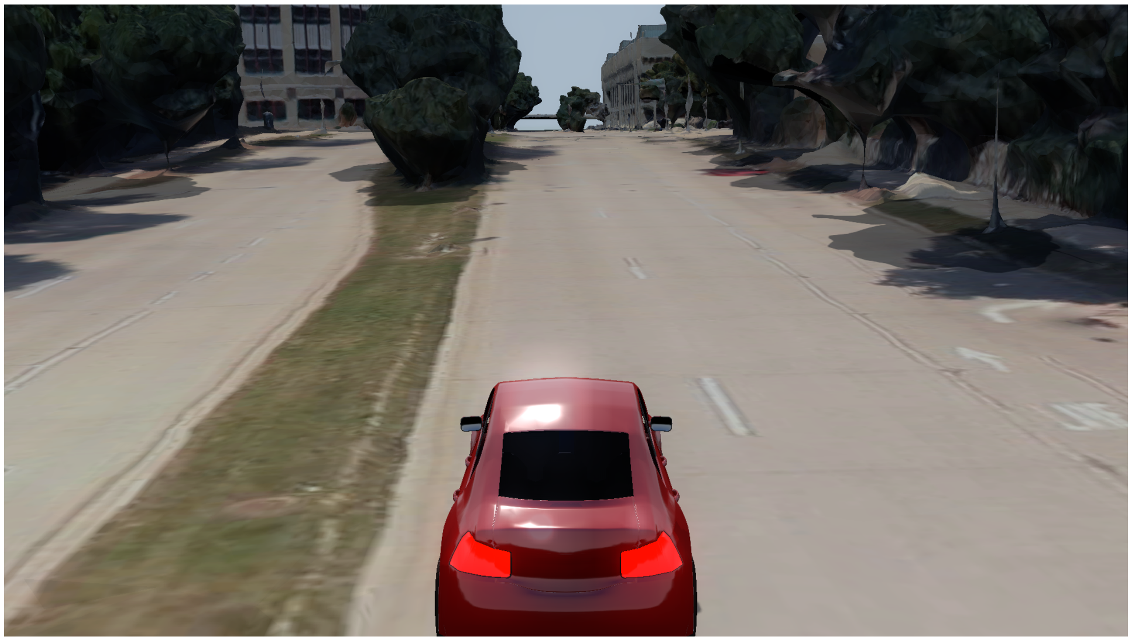

Figure 4.3: A single simulated vehicle driving along a virtual replica of Park St. in Madison, WI. The virtual environment is courtesy of collaboration with Continental Mapping.

5. Conclusions

Synchrono is an evolving simulation platform that, in a physics-based framework, characterizes the dynamics of the interaction between multiple agents. It relies on two servers, the dynamics server and the communication server, to provide a time-, space-, and communication-coherent solution to the task of having multiple dynamic agents operate in a joint scenario. With an eye towards easy scalability, Synchrono has the ability to run in a distributed fashion; i.e., using multiple computing nodes in a supercomputer. The end goal of this infrastructure is to provide a testing environment for software- and hardware-in-the-loop. For the former, one can test the controls stack, or what is called here the PCP, to gauge how good the "brain" is when it comes to handling of the behavior of the agent/autonomous vehicle. For the latter, one can use Synchrono to test whether an actual processing system such as NVIDIA Jetson AGX Xavier is capable of handling the computational load that autonomous driving would likely throw at it when used in a real-world scenario.

5.1 Outcomes

Outcome performance measures:

- We produced an open-source software infrastructure called Synchrono, which is used in conjunction with another open-source software infrastructure called Chrono. Both are developed/augmented by the Simulation-Based Engineering Lab at the University of Wisconsin-Madison (UW-Madison).
- Synchrono will be used in conjunction with a City of Madison pilot project that seeks to understand how V2X communication comes into play in controlling the traffic of the future. We are constructing a virtual replica of a busy Madison intersection that will be equipped with communication hardware for V2X-enabled intersection navigation.
- The UW-Madison Formula Team is using Synchrono in preparation for their first participation in the 2020 edition of the Formula Student Germany Driverless. The team will use Synchrono to determine a piloting control program for their formula vehicle. Details are provided below.
- Several presentations have been given that highlighted this SAFER-SIM project. Specifically, the PIs/students working on Synchrono reported on this project at Northwestern University, University

of Wisconsin-Stout, Clemson University, Mississippi State University, Georgia Tech, MIT, University of California Berkeley, Jet Propulsion Lab, Disney Research, and University of California San Diego.

- We are in the process of using this project to generate a two-hour module that will be taught in eight weeks in conjunction with a residential summer camp for underrepresented high-school students interested in STEM activities.

5.2 Impacts

It is too early to judge the long-term impact of this project. However, Synchrono or a simulation platform like it can make a difference in the deployment of autonomous vehicles and their adoption by the public at large. Specifically, the technology pursued under this SAFER-SIM project does the following:

- Plays a role in reducing the number of crashes from implemented policy, practice, regulation, rulemaking, or legislation. These policies, regulations, etc. can be informed by statistical data that can be generated using a tool such as Synchrono.
- Can help researchers improve traffic flow owing to the scalable attribute of the infrastructure developed, which in theory can simulate thousands of vehicles if deployed on a supercomputer infrastructure. As a byproduct, traffic congestion alleviation and improved flow are poised to have a positive environmental impact as well.

5.3 Future Work

There are four directions of future work. First, Synchrono has not yet been used in any actual project to demonstrate the benefits of using an emulation environment to improve the performance of autonomous vehicles. This issue is being addressed now, with the UW-Madison Formula Team using Synchrono in preparation for their first participation in the 2020 edition of the Formula Student Germany Driverless. No US university has participated in this event as of now. The team will use Synchrono to develop a PCP for their formula vehicle.

Second, the software infrastructure needs further development. One of its current weaknesses is tied to the communication simulation support. Likewise, there is more software infrastructure work needed, as well as more polishing of rough edges when it comes to the user-experience side of the product. As it stands, Synchrono is a product that today can only be used by its developers. It needs further work to improve the robustness of the infrastructure, to validate it, and to facilitate user adoption. The one-year duration of this project, the number of development hours that were budgeted, and the significant amount of work required by this vision have translated into a product that has plenty of opportunities for improvement.

Thirdly, the multi-agent support is nascent. It draws on a TCP/UDP implementation of the physics and communication layers, which adds unjustified overhead to the virtualization layers. This comes into play particularly when one uses Synchrono to simulate multi-agent scenarios. In this report we presented a 30-autonomous-vehicle scenario, yet in principle this number can be scaled up significantly when one has a cluster configuration available to run Synchrono in a distributed memory setup. As is, the software supports execution on a supercomputer, but more work needs to be done to convincingly demonstrate the scalability attribute of the simulation platform.

Lastly, a critical component of this autonomous vehicle simulation framework is the ability to synthetically generate sensor data for software-in-the-loop testing. This sensor data should be physically realistic such that true limitations can be explored. For example, a scenario involving rain, snow, or fog may result in limited sight from a camera. To understand what happens if it unexpectedly begins to rain, the virtual camera sensor should mimic the distortion from the rain on the camera, or the limited range of sight through the raindrops. We have SAFER-SIM funding to develop and implement physics-based camera and LiDAR sensor models, validate them against data captured by physical sensors, and demonstrate their use within Synchrono on realistic mixed autonomous/conventional traffic scenarios. This ongoing work will leverage the project with the UW-Madison driverless formula vehicle as a means for demonstrating our sensor simulation progress.

Bibliography

[1] A. Tasora, R. Serban, H. Mazhar, A. Pazouki, D. Melanz, J. Fleischmann, M. Taylor, H. Sugiyama, and D. Negrut, "Chrono: An open source multi-physics dynamics engine," in *High Performance Computing in Science and Engineering – Lecture Notes in Computer Science* (T. Kozubek, ed.), pp. 19–49, Springer, 2016.

[2] Project Chrono, "Chrono: An Open Source Framework for the Physics-Based Simulation of Dynamic Systems." `http://projectchrono.org`. Accessed: 2016-03-07.

[3] R. Serban, M. Taylor, D. Negrut, and A. Tasora, "Chrono::Vehicle Template-Based Ground Vehicle Modeling and Simulation," *Intl. J. Veh. Performance*, vol. 5, no. 1, pp. 18–39, 2019.

[4] U. DOT, "Isd message creator." `https://webapp.connectedvcs.com/isd/`. Accessed: 2019-05-30.

[5] "Continental Mapping." `https://www.continentalmapping.com/`. Accessed: 2019-04-19.

[6] Google, "Protocol buffers." `http://code.google.com/apis/protocolbuffers/`.