A Simulation Framework for Real-Time, Pedestrian and Autonomous Vehicle Interaction Using Shared Virtual Reality

Thesis

Presented in Partial Fulfillment of the Requirements for the Degree Bachelor of Science in the College of Engineering at The Ohio State University

By

Ritvik Rao

Undergraduate Program in Mechanical Engineering

The Ohio State University

2026

Thesis Committee

Advisor: Dr. Levent Guvenc

Committee Member: Dr. Bilin Aksun-Guvenc

Abstract

The fast pace of advancements in Autonomous Vehicles (AVs) and Advanced Driver Assistance Systems (ADAS) calls for extensive testing for Vulnerable Road Users (VRU) safety, especially pedestrian safety. The current testing methodologies pose many difficulties: software simulations cannot account for true physical car dynamics and testing on public roads subjects pedestrians to potentially dangerous situations like collisions. To overcome these shortcomings, this thesis uses the Vehicle-in-Virtual-Environment (VVE) concept that allows the operation of a real vehicle(s) and real pedestrian(s) in a safe manner within a shared virtual simulation environment where potentially dangerous interactions can take place without any real danger in the real world where the vehicle(s) and pedestrian(s) move at physically different locations.

The developed system uses onboard Micro-Electromechanical Systems (MEMS) sensors on commercially available smartphones, namely GPS, accelerometer, and gyroscope, to monitor pedestrian motion in the physical world and synchronize that motion with that of the avatar in the virtual world. Due to the poor accuracy of consumer-grade GPS data, which was found to have an average horizontal error of 6.13 meters, three different kinematic models were designed to convert and immerse pedestrian motion into the

simulation: asynchronous Kalman filtering; dead reckoning model with gait detection and step timer; and constant velocity model with yaw rotation.

The pedestrian trajectory telemetry is then mapped mathematically into the ego-vehicle's reference frame and communicated through the UDP network to the Simulink vehicle dynamics simulation model. The vehicle dynamics model simultaneously simulates within Unreal Engine 5. Meta Quest 3 VR headset use ensures that the physical pedestrian is immersed in the 3D environment to react to the virtual vehicle. The end-to-end average latency in the bi-directional network was found to be highly responsive, with an end-to-end average latency of 116.7 ms, whereas the lowest latency was seen in the orientation data with a value of 79.6 ms. Thus, such a VVE setup proves highly accessible and is capable of standalone use for testing algorithms without requiring any extra pedestrian hardware or instrumentation to immerse its avatar to be in the path of the actual vehicle’s avatar in the shared virtual environment.

Acknowledgments

I would like to thank the Automated Driving Lab at The Ohio State University for its support. I would also like to thank the Safety 21 National University Transportation Center (UTC) led by Carnegie Mellon University (UTC funded by the Department of Transportation). I would like to thank Xincheng Cao, Haochong Chen, Prof. Levent Guvenc, and Prof. Bilin Aksun-Guvenc for their invaluable advice and foundational work that has helped me throughout my undergraduate thesis.

Table of Contents

Abstract iii
Acknowledgments v
List of Tables viii
List of Figures ix
Chapter 1. Introduction 1
1.1 Introduction to Advanced Driver Assistance Systems and Autonomous Vehicles ... 1
1.2 The Need for Pedestrian Safety 1
1.3 Current Testing Methodologies and Their Limitations 2
1.4 The Vehicle-in-Virtual-Environment (VVE) Framework 3
1.5 Cooperative Collision Avoidance and V2X Communication 5
Chapter 2. Methodology 8
2.1 Hardware Selection and Sensor Setup 9
2.2 Telemetry Data Acquisition and Transfer 11
2.3 Raw Sensor Data Visualization 12
2.4 Accuracy of GPS sensors 13
2.5 Kalman filter for the GPS data 16
2.5.1 State Space Formulation 16
2.5.2 System Dynamics Model (Prediction Phase) 17
2.5.3 Measurement Model (Update Phase) 19
2.6 Dead-Reckoning and Gait Detection Model 21
2.6.1 Gait Detection via 3D Acceleration Magnitude 21
2.6.2 Velocity Smoothing and The Step Timer 22
2.6.3 Yaw Orientation and Directional Motion Generation 23
2.7 Continuous Constant-Velocity Kinematic Model 24
2.7.1 Yaw Alignment and Coordinate Initialization 25
2.7.2 Velocity Vector Decomposition 25
2.7.3 True-Time Positional Integration 26
2.8 Pedestrian Global Frame Transformation 26
2.9 Frame Transformation: Ego Vehicle 27

2.9.1 Origin Locking and Translation 27
2.9.2 Rotational Alignment 28
2.9.3 Global Offset 29
2.10 Ego-Vehicle Dynamics and Virtual Environment Integration 29
2.11 Bidirectional Network Architecture and Communication Streams 31
2.12 Virtual Reality Headset and Environment Setup 32
Chapter 3. Results 33
3.1 Qualitative Video Demonstration 33
3.2.1 Analysis of Latency Discrepancies 36
Chapter 4. Conclusion and Future Work 38
References 40
Appendix A. Link to all Simulink and MATLAB files 43

List of Tables

Table 1: GPS accuracy data.....15
Table 2: Latency statics.....36

List of Figures

Figure 1: A VVE vehicle and the components in the trunk (source: Ohio State University Automated Driving Lab) 5
Figure 2: The overall system architecture with pedestrian, ego vehicle and the simulation 9
Figure 3: MATLAB phone application 11
Figure 4: MATLAB phone app initialization code 12
Figure 5: Plots of sensor data 13
Figure 6: GPS data with true path walked 14
Figure 7: Drift bounds and accuracy over time 15
Figure 8: Simulink with 2 vehicles, pedestrian and bicyclist 30
Figure 9: Simulink with vehicle and pedestrian 30
Figure 10: Virtual reality setup 32
Figure 11: Latency plots for sensors 35

Chapter 1. Introduction

1.1 Introduction to Advanced Driver Assistance Systems and Autonomous Vehicles

Advanced Driver Assistance Systems (ADAS) encompass electronic technologies that reduce the workload on the driver while also making the vehicle safer, not only for the driver and passengers, but also for other road users like pedestrians, other vehicles, and bicyclists. Autonomous Vehicles (AVs) are the highest level of ADAS which allows for end-to-end autonomous driving without any input from the driver. They use sensors like cameras, lidar, radar, and ultrasound to perceive their environment, understand the situation and make autonomous decisions [1], [4]. Common examples of ADAS features are lane-keeping assist, automatic emergency braking, and adaptive cruise control [6]. To control these vehicles, a highly robust mechatronic frameworks is needed to ensure safety of the passengers and the environment they are interacting with [7], [11]. As the integration of vehicle-to-everything (V2X) connectivity increases, modern vehicles can share telemetry, proactively preventing traffic crashes while simultaneously enhancing mobility and reducing environmental impact [9], [12].

1.2 The Need for Pedestrian Safety

Despite rapid advancements in ADAS, Vulnerable Road Users (VRUs) especially pedestrians are the most exposed group of road users who are exposed to lethal collisions

[1]. According to the World Health Organization (WHO), road traffic injuries are projected to rise significantly, becoming the third highest leading cause of disability [18]. In the United States alone, human-driven and, to some extent, partially autonomous vehicles account for thousands of pedestrian fatalities annually [2], [3].

Research indicates that five priority pre-crash scenarios, which are highly addressable by Vehicle-to-Pedestrian (V2P) crash avoidance technologies, account for 79% of all target pedestrian crashes and 91% of deadly pedestrian crashes [2]. Consequently, deployment of connected vehicle technologies, targeting the unpredictable nature of pedestrian kinematics, has become an important goal for transportation authorities and automotive engineers [10], [13].

1.3 Current Testing Methodologies and Their Limitations

The validation of automated vehicle algorithms is in a very critical stage of development, with rapid advancements constantly occurring in the industry. The standard way automotive OEMs develop ADAS involves Model-in-the-Loop (MIL) and Software-in-the-Loop (SIL) simulations, using platforms like CARLA, LG SVL, CarMaker, and Unreal Engine [5], [22]. After testing virtually on those platforms, they perform testing using Hardware-in-the-Loop (HIL) simulations, where real physical components, such as a dSpace MicroAutoBox (MABX) or actual communication modules, are integrated into the simulation environment [19].

While these methods of testing involving SIL, MIL, and HIL provide scalability and efficiency, they lack true vehicle dynamics, which must be emulated [5]. Vehicle-in-the-Loop (VIL) testing introduces the physical vehicle, but early-stage algorithmic development ultimately forces testing onto public roads [20]. This public road development approach is inherently unsafe, as it forces other road users including VRUs to involuntarily participate in the beta-testing of AV software [1], [4]. Furthermore, physical testing is inefficient because repeatedly setting up test scenarios is time-consuming. Worse, relying on millions of driven miles to encounter rare but critical "edge-case" pedestrian scenarios are statistically and logistically unfeasible [4].

1.4 The Vehicle-in-Virtual-Environment (VVE) Framework

To overcome the safety risks associated with public road testing which were mentioned previously and the limitations of pure software simulation, the Vehicle-in-Virtual-Environment (VVE) methodology was introduced [1], [4], [5]. VVE serves as a mixed-reality testing framework that connects the physical and virtual worlds, allowing data to be shared between them. In practice, this involves tracking the live location, state, and heading of a physical ego-vehicle operating in a secure, empty space, such as a proving ground or empty parking lot. That data is then used to synchronize and simulate in real time correspondence the vehicle's avatar presence inside a highly realistic virtual environment [5]. Other vehicles can also take place in this shared virtual environment by synchronizing their motions with that of their avatars in the shared virtual environment.

Additionally, telemetry from a physical pedestrian walking in an isolated location can be synchronized and fed into the shared virtual world in real time. The virtual environment, then, uses this data to generate virtual sensor readings, which are injected directly into the vehicle's onboard computers [1], [4]. This setup provides the ego-vehicle with full perception and situational awareness, allowing it to react to virtual or tracked actors with zero risk of a physical crash. Despite its benefits, VVE is difficult to implement. Running a high-fidelity VVE requires a lot of computational power to render environments in real time. Furthermore, preventing spatial desynchronization between the physical and virtual spaces requires centimeter-level accuracy, which demands highly precise and often expensive telemetry equipment, such as Real-Time Kinematics (RTK) GPS [5], MABx, and a simulation computer as seen in Figure 1.

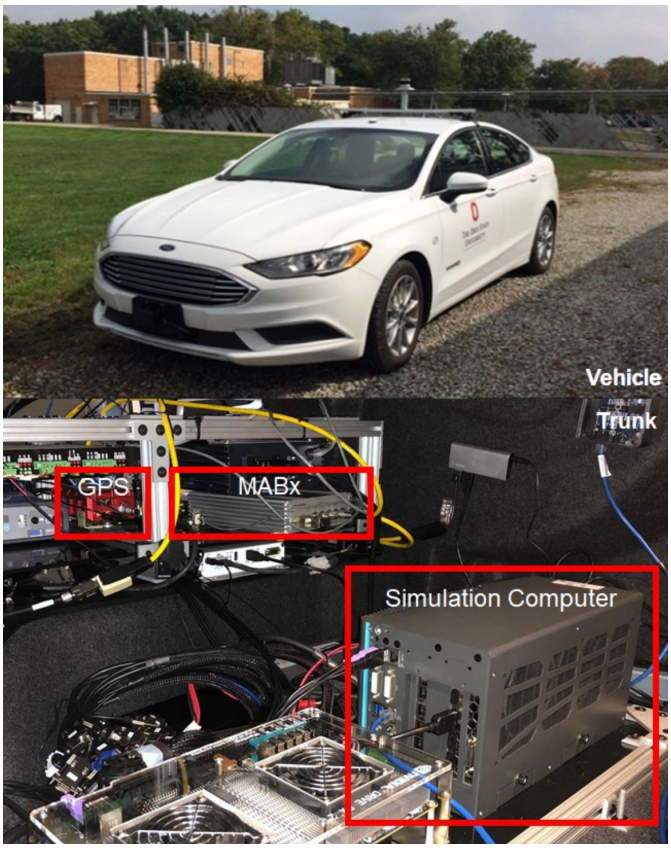

Figure 1: A VVE vehicle and the components in the trunk (source: Ohio State University Automated Driving Lab)

1.5 Cooperative Collision Avoidance and V2X Communication

There is one major disadvantage of perception sensors such as cameras, radars, and LIDARs: they need to have a clear line-of-sight [16]. Urban areas contain numerous objects that could obstruct sensors, such as buildings, parked cars, or blind spots, resulting in harder detection for non-line of sight pedestrians [2], [3].

Cooperative Collision Avoidance (CCA) takes advantage of V2X communication to overcome some shortcomings of those types of sensors that have to work with line-of-sight [17], [21]. During an No-Line-of-Sight (NLOS) situation, V2P communication allows exchanging important telemetry data – location, speed, and direction of the movement, between a car and the mobile device of a pedestrian [3]. Through using DSRC modems, which operate at 5.9 GHz frequency or current technologies for cellular V2X communication, vehicles can detect pedestrians prior to them entering line-of-sight [9], [14].

Once the ego-vehicle gets data from a pedestrian using V2P data pipelines, the ego-vehicle can alter its trajectory. A highly effective approach for this is the Elastic Band method [2], [8]. Originally conceptualized for mobile robotics, this method treats the vehicle's initial planned path as an elastic band that deforms in real time when subjected to the "repulsive forces" of a detected obstacle (e.g., a pedestrian) [2]. The Elastic Band Connected Collision Avoidance (EB-CCA) framework ensures that the autonomous vehicle either stops safely or navigates around the pedestrian, maintaining a mathematically defined "socially acceptable distance" to respect the physical boundaries of the human user [3], [8], [15].

Testing Cooperative Collision Avoidance and NLOS pedestrians or other vehicles will benefit greatly from the use of VVE methodology. This thesis focuses on VVE for pedestrian safety algorithm evaluation and focuses more on the pedestrian motion capture, synchronization and realistic and real time replay in the VVE virtual environment and in immersing the pedestrian in the same environment using a VR headset. The methodology

is presented in Chapter 2. The results are presented and discussed in Chapter 3. The thesis ends with conclusions and recommendations for future work in Chapter 4.

Chapter 2. Methodology

This chapter details the architecture and implementation of the pedestrian actor within the Vehicle-in-Virtual-Environment (VVE) framework. A bi-directional data sharing process between a real pedestrian, a real vehicle, and a virtual simulation environment was established. The goal was to develop a framework that is robust and has low latency. The overall system architecture, showing the flow of sensor data from the pedestrian's hardware (smartphone) to the simulation environment, is presented below in Figure 2. The following sections break down each stage of this framework, starting with collecting the data, processing it into usable form, and then simulating and visualizing the data.

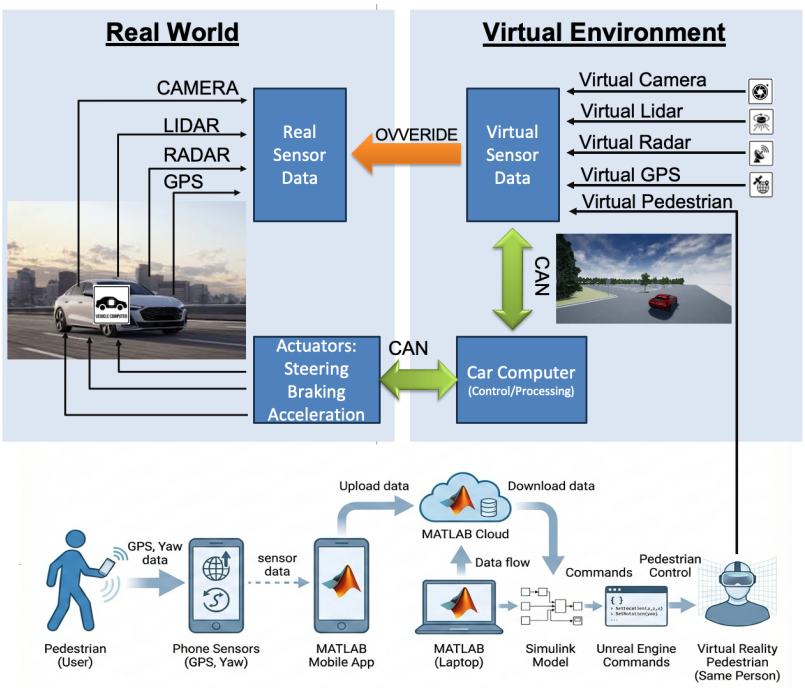

Figure 2: The overall system architecture with pedestrian, ego vehicle and the simulation

2.1 Hardware Selection and Sensor Setup

While building this VVE framework, I wanted to ensure that it is scalable, cost-effective, and highly representative of real-world Vehicle-to-Pedestrian (V2P) situations. The use of sophisticated, highly advanced technologies such as RTK GPS and/or wearable motion capture systems would be inconsistent with the underlying premise of a realistic simulation of pedestrian tracking using a smartphone. As a result, a regular iPhone 16 was used to track the pedestrian's movements.

To get data from the mobile device, I used the MATLAB Mobile app [23]. This platform was selected as it provides direct access to the smartphone's internal Micro-

Electromechanical Systems (MEMS) sensors without requiring the development of a custom application from scratch.

As illustrated in 3, the primary sensors activated within the application for this research included the Position Sensor (GPS) for global coordinates, the Accelerometer for high-frequency linear acceleration, and the Angular Velocity (Gyroscope) to capture the rotational yaw rate of the pedestrian.

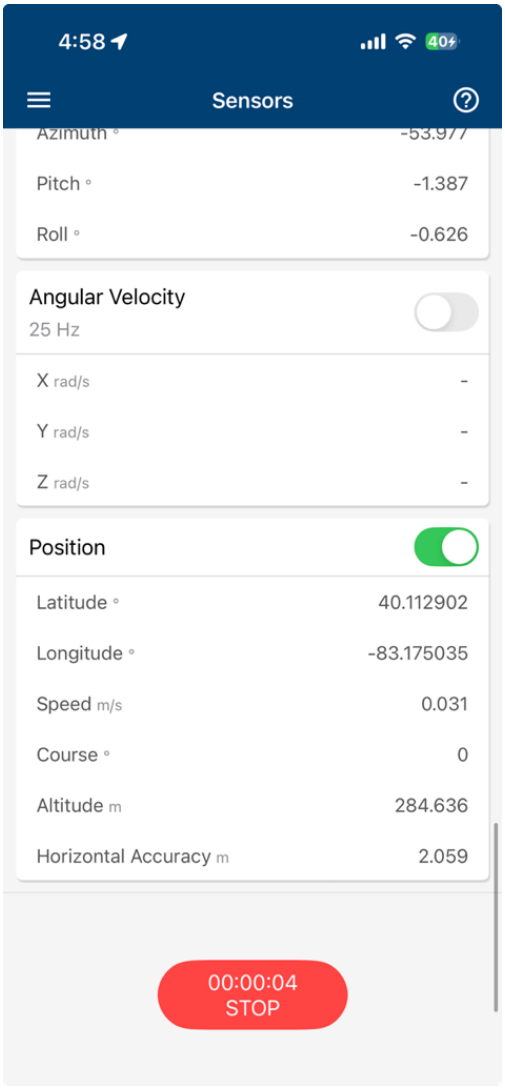

Figure 3: MATLAB phone application

2.2 Telemetry Data Acquisition and Transfer

The sensor data from the phone gets uploaded to the MATLAB cloud that can be accessed by any laptop signed in with the same account. Data is transmitted from the MATLAB Mobile application on the smartphone to a central processing laptop running a MATLAB desktop instance.

This connection was initialized utilizing MATLAB's native mobiledev object [24]. The lines of code given in Figure 4 connect the laptop and the phone over the local Wi-Fi network (or the cellular network), allowing the laptop to remotely enable sensors, turn on and off data logging, and fetch the live data. The initialization script used to pull the data into MATLAB to be processed are shown in Figure 4.

```
% --- INITIALIZATION ---
disp('Connecting to Phone');
m = mobiledev;
m.PositionSensorEnabled = true;
m.AccelerationSensorEnabled = true;
m.AngularVelocitySensorEnabled = true;
m.SampleRate = 100;
```

Figure 4: MATLAB phone app initialization code

2.3 Raw Sensor Data Visualization

Once the device was connected and the laptop successfully could access the data, the raw data from the sensors was streamed and pulled into the MATLAB workspace. The data was structured into arrays containing timestamps, positional coordinates, speeds, and angular velocities.

To verify the integrity of the data transmission, the incoming signals were plotted and visualized. Figure 5 demonstrates the live visualization of the raw linear acceleration data plotted against time during a pedestrian walking straight between 2 points twice.

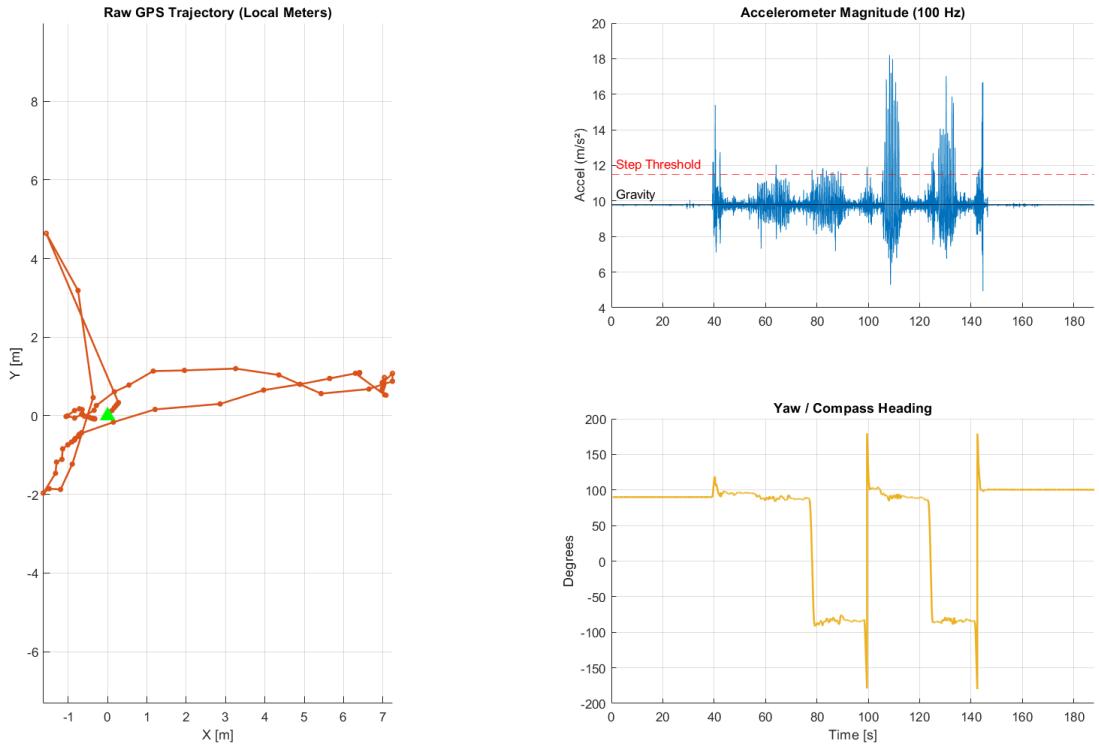

Figure 5: Plots of sensor data

2.4 Accuracy of GPS sensors

After plotting the data in Figure 5 against the real path walked in Figure 6, initial testing indicated significant limitations of the raw data obtained from the sensors, mainly the GPS data. A GPS embedded in a smartphone works at a low refresh rate (approximately 1 Hz) and is prone to multiple errors.

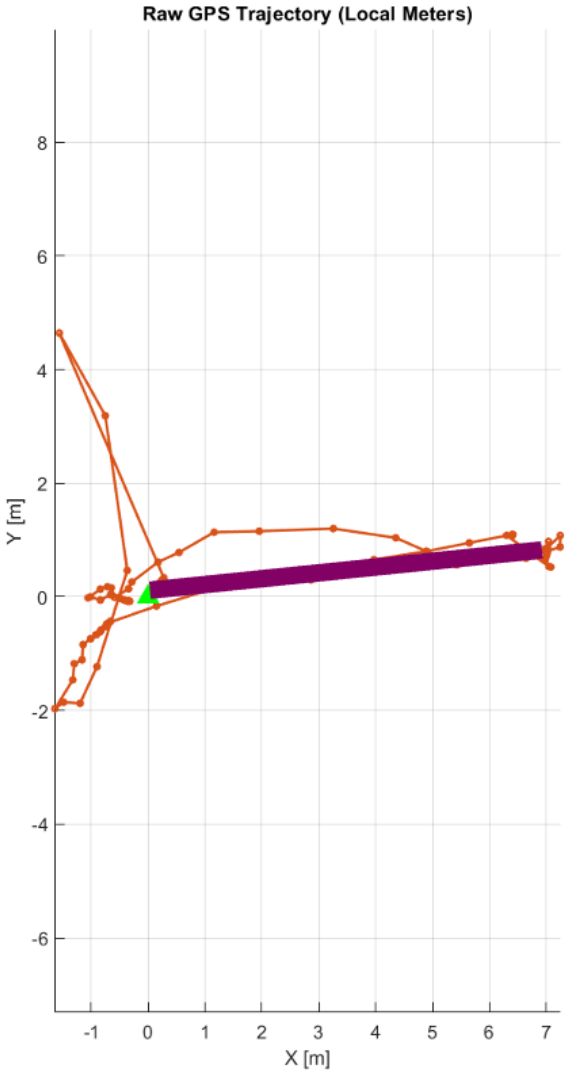

Figure 6: GPS data with true path walked

To analyze the low accuracy, an experiment was performed to determine the amount of drift and error that occurred in the GPS of the smartphone when in motion. This can be observed in Table 1, where statistics on the horizontal inaccuracy radius were gathered from this test run.

The poor accuracy of the GPS system is graphically illustrated in Figure 7, which shows how much drift occurs by plotting out the 2D drift bounds at the localized origin.

Table 1: GPS accuracy data

Metric	Value
Total Data Points Logged	121
Average (Mean) Inaccuracy:	6.13 meters
Worst (Max) Inaccuracy	9.95 meters
Best (Min) Inaccuracy	2.00 meters

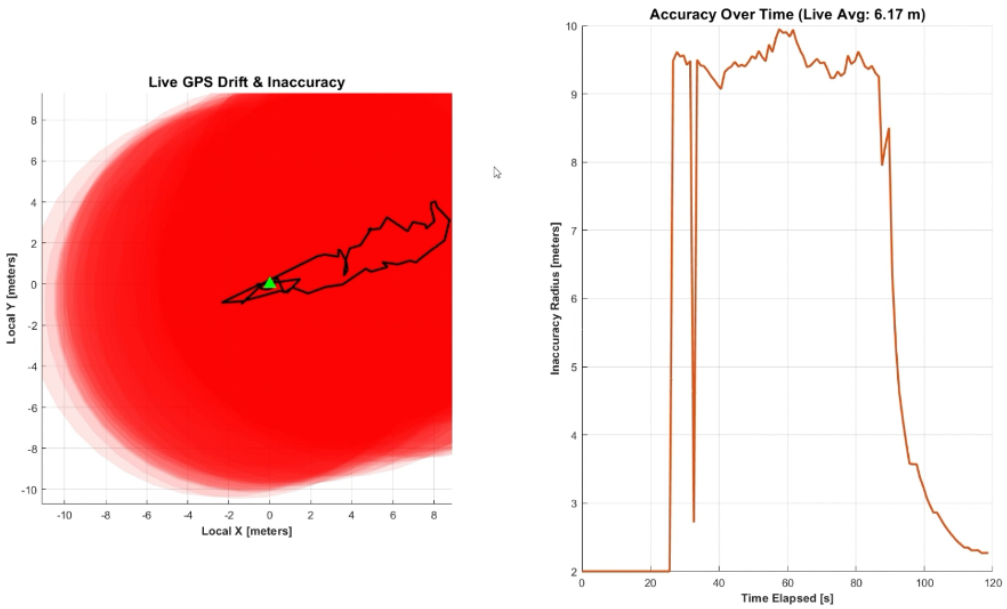

Figure 7: Drift bounds and accuracy over time

With these error margins, the data is not very useful for autonomous vehicle simulation. If an autonomous ego-vehicle operating in Unreal Engine 5 or CARLA relied solely on this

raw data, a pedestrian standing completely still in the real world would appear to randomly teleport within a 6-to-10-meter radius in the virtual world. The erratic, jumping nature of this raw positional data compared to an actual path walked by the user was explicitly demonstrated in Figure 6.

Due to these limitations, a complex sensor fusion and dead-reckoning methodology was developed, which will be explained in the next sections.

2.5 Kalman filter for the GPS data

The first method to process the GPS data to be more useful is the Kalman filter. The Kalman filter is the optimal estimator for processing the GPS data, which consists of noise with low frequency updates, while simultaneously considering the accelerometer and gyroscope data of the phone, which have high-frequency updates.

2.5.1 State Space Formulation

Position of the pedestrian in the 2D cartesian coordinate frame is tracked. The system state vector $\widehat{x_k}$ is represented by a 4-dimensional column vector that consists of localized positions x and y and their corresponding velocities

$$\widehat{x_k} = \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix} \tag{1}$$

where $\widehat{x_k}$ is the system state vector, x and y are the localized coordinates, and v_x and v_y are corresponding direction spatial velocities.

The control input vector u_k is obtained based on readings from the accelerometer and magnetometer which are sensors within the mobile phone. The effect of sensor noise is removed by scaling the longitudinal acceleration and splitting it into global longitudinal and lateral forces based on the relative yaw angle (ψ) from the compass

$$u_k = \begin{bmatrix} a_x \\ a_y \end{bmatrix} = \begin{bmatrix} a_{forward} \cos(\psi) \\ a_{forward} \sin(\psi) \end{bmatrix} \quad (2)$$

where u_k is the control input vector, a_x and a_y are the longitudinal and lateral accelerations, respectively, $a_{forward}$ is the scaled longitudinal acceleration, and ψ is the relative yaw angle.

2.5.2 System Dynamics Model (Prediction Phase)

As the 100 Hz loop is much faster than the polling cycle of GPS, the filter depends mostly on its prediction function, which assumes constant movement of the person. In the State Transition Matrix (A), we define the dynamics of the movement between states during one discrete time step Δt.

It is important to note that there is a virtual coefficient of friction equal to 0.85 for the velocity states in the state transition matrix. This serves as a physical friction in the physics engine to avoid gliding caused by an accelerometer spike. The state matrix in the process dynamics equation or the motion model is given by

$$A = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0.85 & 0 \\ 0 & 0 & 0 & 0.85 \end{bmatrix} \tag{3}$$

where A is the state transition matrix, and Δt is the discrete time step. The first two lines correspond to a uniform speed-based position updates in the x and y directions. The last two lines correspond to reduction in speed in the x and y directions by a factor of 0.85 during each iteration.

The Control Input Matrix (B) defines how the physical forces measured by the IMU influence the spatial position and velocity, derived from standard Newtonian kinematics and is given by

$$B = \begin{bmatrix} 0.5\Delta t^2 & 0 \\ 0 & 0.5\Delta t^2 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \tag{4}$$

Since the two inputs are acceleration components, the first two lines when combined with the state matrix correspond to constant acceleration motion in the x and y directions. The last two lines increase speed in the x and y directions due to the constant acceleration inputs.

During every iteration of the 100Hz loop, the a priori state estimate $\hat{x}_{k|k-1}$ and the predicted error covariance $P_{k|k-1}$ are calculated. The fixed Process Noise Covariance matrix (Q) accounts for inherent hardware drift in the IMU

$$\hat{x}_{k|k-1} = A\hat{x}_{k-1|k-1} + B\hat{u}_k \tag{5}$$

$$P_{k|k-1} = \mathrm{A}P_{k-1|k-1}A^T + Q \tag{6}$$

where $\hat{x}_{k|k-1}$ is the predicted state estimate (based on the motion model and the IMU readings), A is the state transition matrix, $\hat{x}_{k-1|k-1}$ is the previous state estimate, B is the control input matrix, u_k is the control vector, $P_{k|k-1}$ is the predicted error covariance matrix, $P_{k-1|k-1}$ is the previous error covariance matrix, and Q is the process noise covariance matrix [25].

2.5.3 Measurement Model (Update Phase)

When the MATLAB controller detects a newly timestamped GPS packet, then the filter runs an update process to correct for the drift caused by the dead reckoning algorithm given in the previous section. The Observation Matrix (C) filters the position variables from the state vector because the GPS sensor cannot determine the instantaneous speed of the pedestrian and is given by

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{7}$$

To deal with the multipath effects in cities, the Measurement Noise Covariance (R_k) matrix is updated. The covariance value is adjusted based on the ratio of the radius of the instantaneous horizontal accuracy radius (acc_{gps}), this penalizes the measurement if the satellite signal is degraded

$$R_k = R_{base}\left(\frac{acc_{gps}}{5}\right) \tag{8}$$

where R_k is the dynamic noise measurement covariance, R_{base} is the baseline uncertainty parameter, and acc_{gps} is the instantaneous horizontal accuracy radius reported by the GPS hardware.

The Kalman Gain (K_k) evaluates the ratio of certainty between the IMU's prediction (P) compared to the dynamically scaled GPS measurement noise (R_k) using

$$K_k = P_{k|k-1}C^T\left(CP_{k|k-1}C^T + R_k\right)^{-1} \tag{9}$$

where K_k is the Kalman Gain, $P_{k|k-1}$ is the predicted error covariance matrix, C is the observation matrix, and R_k is the dynamic measurement noise covariance matrix [25].

The final step is calculating the measurement residual, which is done by comparing the raw GPS coordinates (z_k) against the predicted coordinates. Next, the state vector and error covariance are updated (a posteriori) using the gain value, ensuring that the simulated trajectory is smoothly pulled toward the actual coordinates while avoiding sudden teleportation. The following equations

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k\left(z_k - C\hat{x}_{k|k-1}\right) \tag{10}$$

$$P_{k|k} = (I - K_kC)P_{k|k-1} \tag{11}$$

where $\hat{x}_{k|k}$ is the updated a posteriori state estimate, $\hat{x}_{k|k-1}$ is the predicted state vector, K_k is the Kalman filter gain, z_k is the raw GPS measurement vector, C is the observation matrix, $P_{k|k}$ is the updated error covariance matrix, and I is the identity matrix [25] is used.

The output position coordinates are shifted to match the map origin in the shared virtual Unreal Engine environment and sent via UDP to the host simulation.

2.6 Dead-Reckoning and Gait Detection Model

The second method for pedestrian simulation is a localized dead reckoning which continuously updates the pedestrian position based on yaw and the detection of footsteps. In this case, the GPS will not be used since the localization is achieved by using a high-speed (100 Hz) measurement using micro electromechanical sensors incorporated into the mobile device to detect steps and generate the forward velocity of movement in the direction the phone is facing.

2.6.1 Gait Detection via 3D Acceleration Magnitude

To simulate the experience of a forward walk, it is necessary for the software to have the capability to distinguish whether the user is static, performs general operations with the device, or makes steps deliberately. As the physical orientation of the mobile device is unrestricted (held in hand, put into the pocket, etc.), it cannot rely solely on one axis of acceleration measurement. Instead, the system computes the 3D Euclidean magnitude of acceleration vector for every 100 Hz sample

$$|| a || = \sqrt{a_x^2 + a_y^2 + a_z^2} \quad (12)$$

where ||a|| is the total acceleration magnitude, a_x is the acceleration along the X-axis, a_y is the acceleration along the Y-axis, and a_z is the acceleration along the Z-axis.

The absolute value of acceleration is around 9.81 m/s^2 while a person is not moving due to gravitational acceleration. When the person starts moving, there is a spike in this value which is detected using the pre-established heuristic peak-detection threshold equal to 11.5 m/s^2. When exceeding this threshold, a step is detected.

2.6.2 Velocity Smoothing and The Step Timer

A problem that comes from incorporating raw impulses from step detection into the simulator is temporal discretization. If only the velocity vector is activated at the exact microsecond when the heel-strike passes the set threshold, then the movement of the virtual person is characterized by a discontinuous movement pattern.

To counteract that, a continuous walk temporal mechanism called the step timer was implemented. Once the acceleration spikes beyond the threshold, the internal timer of the person (t_{step}) is reset to zero. As long as t_{step}<1.50 seconds, it can be considered that the person is in-motion. During the in-motion stage, the velocity (v) of the person stays constant corresponding to the average walking speed of a fast walking person:

$$v = \begin{cases} 1.4\ m/s, if\ t_{step} < 1.5 \\ 0\ m/s, if\ t_{step} \geq 1.5 \end{cases} \quad (13)$$

where v is the applied forward velocity of the pedestrian and t_{step} is the elapsed time in seconds since the last detected acceleration spike. The 1.50-second time window functions as a low-pass filter and provides a perfect solution for handling different human striding cadences.

2.6.3 Yaw Orientation and Directional Motion Generation

After getting the constant scalar value for the speed through the step timer, the system, then, gets the orientation of the phone so that it can convert that speed into displacement values in the 2D plane. This is done using the geographic orientation of the mobile device as derived from the magnetometer readings on the phone.

To align the physical device with the virtual simulation environment, the live raw yaw is normalized against an initialized origin heading

$$\psi_{relative} = \psi_{raw} - \psi_{origin} \tag{14}$$

where $\psi_{relative}$ is the normalized relative heading used for the simulation, ψ_{raw} is the live magnetometer reading, and ψ_{origin} is the locked initial heading recorded at system startup.

With the relative heading calculated, the scalar forward velocity (v) created from the step timer is, then, considered the hypotenuse of the right triangle. The system then uses trigonometric decomposition to create orthogonal velocity vectors for the longitudinal and lateral axes:

$$v_x = v \cdot cos(\psi_{relative}) \tag{15}$$

$$v_y = v \cdot sin(\psi_{relative}) \tag{16}$$

where v_x is the velocity vector along the X-axis, v_y is the velocity vector along the Y-axis, v is the smoothed forward velocity computed based on the step timer, and $\psi_{relative}$ is the normalized relative heading.

Lastly, these velocities are numerically integrated over the 100 Hz time frame to calculate the continuous location coordinates of the pedestrian in his local Cartesian coordinate system, thus computing the smooth motion vector in the precise direction toward which the mobile device is pointing

$$x_{local} = x_{local} + v_x \Delta t \tag{17}$$

$$y_{local} = y_{local} + v_y \Delta t \tag{18}$$

where x_{local} and y_{local} are the computed spatial coordinates, v_x and v_y are the decomposed directional velocities, and Δt is the exact fractional seconds elapsed since the previous execution frame.

2.7 Continuous Constant-Velocity Kinematic Model

Even though the algorithm proposed in the previous section proved quite successful in smoothing out impulses coming from hardware sensors, the stop-and-go nature of the step timer might still have caused small temporal discrepancies within the physics engine. To fix this problem, and make sure a smooth trajectory is achieved, another kinematic model was developed and used. The idea behind this model was to totally exclude the step

counting algorithm based on acceleration data from the equation of motion. Instead, the forward velocity would remain constant, while spatial displacement would be controlled by processing the user's physical Yaw (heading).

2.7.1 Yaw Alignment and Coordinate Initialization

To make sure that the heading angle from the hardware corresponds to the forward direction in the simulation, the angle needs to be normalized to fit the simulation requirements. The first thing done during runtime is flipping the sign of the raw angle to fit the Unreal Engine's left-hand coordinate system, then converting it from degrees into radians and subtracting the locked initial heading angle:

$$\psi_{relative} = \left(-\psi_{raw} \cdot \frac{\pi}{180}\right) - \psi_{origin} \tag{19}$$

where $\psi_{relative}$ is the normalized relative heading used for the simulation, ψ_{raw} is the live magnetometer reading in degrees, and ψ_{origin} is the locked initial heading in radians.

2.7.2 Velocity Vector Decomposition

Since the test for the magnitude of acceleration is skipped, the forward walking speed remains permanently activated. The controller treats this scalar as the hypotenuse of a right triangle and uses trigonometry to decompose it into two orthogonal velocity vectors as

$$v_x = v_{walk} \cdot \cos(\psi_{relative}) \tag{20}$$

$$v_y = v_{walk} \cdot \sin(\psi_{relative}) \tag{21}$$

where v_x is the longitudinal velocity vector along to the X-axis, v_y is the lateral velocity vector along the Y-axis, v_{walk} is the permanent constant forward walking speed of 1.4m/s and $\psi_{relative}$ is the normalized relative heading. This allows us to determine what share of the speed is going to be applied in each direction. If the user aims the mobile device at an angle of 0°, all 100% of the constant speed will be dedicated to movement along the X-axis. If the user changes the angle to 45°, then speed will evenly be split between the two axes without the slightest stutter.

2.7.3 True-Time Positional Integration

These obtained velocities are then translated into space using fast Euler integration coupled with an accurate real-time stopwatch. The small displacements are computed and directly summed into the pedestrian's local position

$$X_{local} = X_{local} + (v_x \cdot \Delta t) \tag{22}$$

$$Y_{local} = Y_{local} + (v_y \cdot \Delta t) \tag{23}$$

where X_{local} is the current position on the X-axis, $Y_{local}$is the current position on the Y-axis, v_x and v_y are the decomposed directional velocities, and Δt is the elapsed fractional seconds since the previous execution frame.

2.8 Pedestrian Global Frame Transformation

Regardless of whether we use one of the three methods to estimate pedestrian's trajectory (Kalman Filter method described in section 2.5, Gait Detection approach discussed in section 2.6, or constant-velocity dead-reckoning from section 2.7) the position vectors will be relative to a mathematical zero (0,0) location. But the high-fidelity simulation platforms

such as Unreal Engine and CARLA implement absolute global coordinate systems that represent the physical testing field or a virtual city. To accurately position the pedestrian in the simulation environment, frame transformation on the coordinates is required. A static geographic translation is applied to convert the local track into the target global map coordinates

$$X_{ped} = X_{local} + X_{offset} \tag{24}$$

$$Y_{ped} = Y_{local} + Y_{offset} \tag{25}$$

where X_{ped} and Y_{ped} are coordinates of a pedestrian within the absolute coordinate system of an Unreal Engine virtual world, X_{local} and Y_{local} are coordinates estimated using one of the pedestrian tracking techniques above, while X_{offset} and Y_{offset} are spatial translation components needed to match the local zero point with the starting crosswalk/sidewalk on the map.

2.9 Frame Transformation: Ego Vehicle

The raw data from the ego vehicle must also undergo frame transformations before being rendered in Unreal Engine.

2.9.1 Origin Locking and Translation

The position of the ego vehicle at the start is saved and fixed as a spatial reference. Any further positional information collected from the ego vehicle via GPS will then be measured in relation to this fixed origin by obtaining the difference between the two:

$$dx = X_{raw} - X_{origin} \tag{26}$$

$$dy = Y_{raw} - Y_{origin} \tag{27}$$

where *dx* and *dy* are the lateral and longitudinal distance, X_{raw} and Y_{raw} are the current raw GPS measurement along the axis, and X_{origin} and Y_{origin} are the locked coordinate recorded at system startup. By doing this translation, a new coordinate system centered around a local mathematical zero is created for measuring ego vehicle position.

2.9.2 Rotational Alignment

After the translation process, the displacement vector needs to be rotated to ensure that the orientation of the vehicle at the start of the simulation is set as the principal axes for travel. Otherwise, the simulation coordinate frame would be randomly oriented with respect to the initial orientation of the test vehicle. To achieve this, a 2D rotation matrix based on the vehicle yaw angle is used to transform the displacement vector:

$$\begin{bmatrix} X_{rotated} \\ Y_{rotated} \end{bmatrix} = \begin{bmatrix} cos(\psi_{origin}) & sin(\psi_{origin}) \\ -sin(\psi_{origin}) & cos(\psi_{origin}) \end{bmatrix} \begin{bmatrix} dx \\ dy \end{bmatrix} \tag{28}$$

where $X_{rotated}$ and $Y_{rotated}$ are coordinates in the heading frame, ψ_{origin} is the vehicle's yaw that is fixed during the initialization of the system, and *dx* and *dy* are the translation vector components obtained during the previous section. This guarantees that a vehicle traveling forward along its heading at the beginning of a trial will follow the X-axis in the simulation space, for example.

2.9.3 Global Offset

The coordinates obtained by rotating are relative to the mathematical point origin but not to the real location within the world as defined by the Unreal Engine world map, which operates on an absolute global coordinate system based on the actual world coordinates on the map. For correctly placing the vehicle within the world, a global offset is needed, and that is why we add a global offset vector to the rotated coordinates as follows:

$$X_{car} = X_{rotated} + X_{offset} \tag{29}$$

$$Y_{car} = Y_{rotated} + Y_{offset} \tag{30}$$

where X_{car} and Y_{car} are the final lateral world coordinate of the ego vehicle, $X_{rotated}$ and $Y_{rotated}$ are the heading-aligned local position, and X_{offset} and Y_{offset} the pre-calibrated constant displacements along the longitudinal and lateral axes, respectively. These new values of X_{car} and Y_{car} are sent using the UDP stream to the virtual simulation world to update the vehicle location.

2.10 Ego-Vehicle Dynamics and Virtual Environment Integration

The fundamental building blocks for modeling the physical dynamics of the autonomous ego-vehicle were simulated in Simulink. High-fidelity simulation requires realistic representation of longitudinal, lateral, and vertical force acting upon the vehicle. This was accomplished using the MATLAB Vehicle Dynamics Blockset [26].

This software package delivers complete reference models which compute the precise kinematic properties of the ego-vehicle. The most significant aspect of this simulation framework, highlighted by the high-level diagram in Figure 8 and Figure 9 below, lies in its inherent capability for co-simulating with a 3D rendering engine.

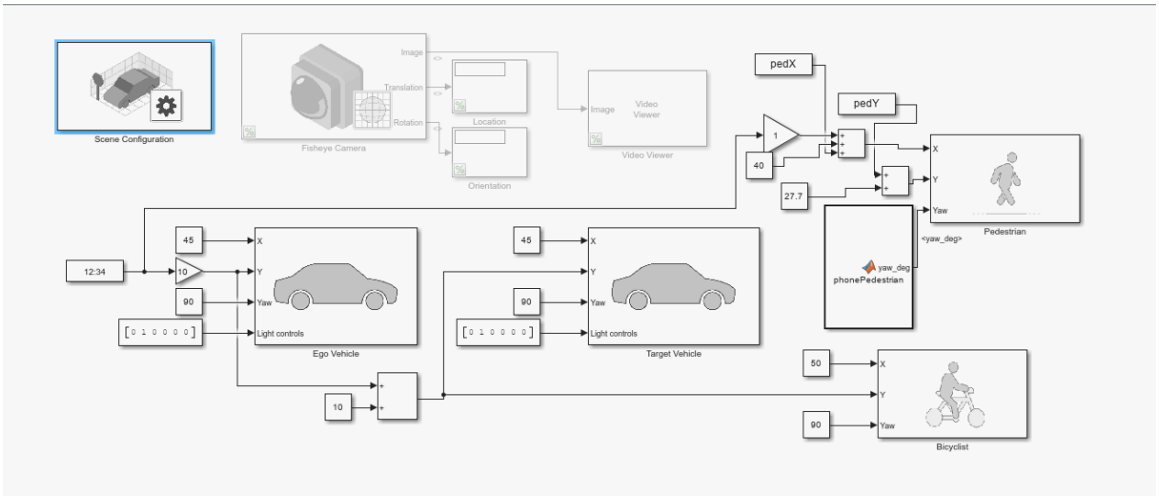

Figure 8: Simulink with 2 vehicles, pedestrian and bicyclist

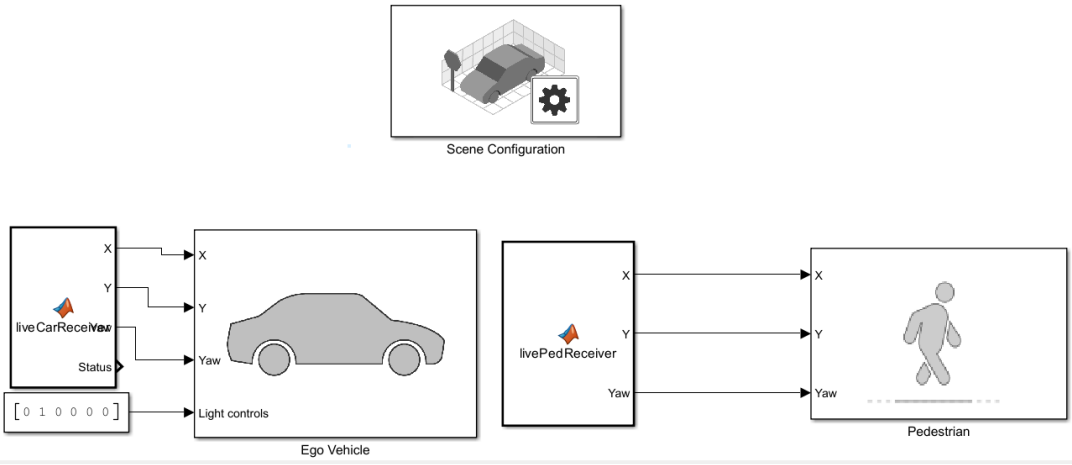

Figure 9: Simulink with vehicle and pedestrian

The ego-vehicle's simulation within Simulink is then streamed live into the Unreal Engine (UE5) environment to realistically render the ego-vehicle's physical dynamics in the 3D

environment. At the same time, the interface with UE5 enables feedback from the virtual environment in terms of spatial information, including pedestrian telemetry synchronized using UDP packets [26].

2.11 Bidirectional Network Architecture and Communication Streams

To establish an effective connection between the physical world and the virtual environment, the VVE design necessitates a reliable and low-latency network architecture. To create a connection between the pedestrian and the autonomous ego-vehicle, both of whom are in different physical spaces but need to be able to communicate within a virtual environment, a bidirectional LAN bridge is employed. For faster transmissions and reduced latency, the User Datagram Protocol (UDP) is used to send data through two continuous and synchronized streams.

The first data stream mirrors the physical autonomous vehicle into the pedestrian's virtual environment. In real-time, the computing module in the ego-vehicle captures the live dynamic state of the ego-vehicle, such as global GPS coordinate data, speed, and yaw angle, and sends this information via a UDP payload to the laptop that has the pedestrian data. The laptop analyzes the incoming data stream and uses the Unreal Engine 5 to render the ego-vehicle and track its exact location and path within the virtual environment.

Concurrently, the second data stream closes the loop by sending the live stream back to the ego-vehicle's onboard computer. The central laptop calculates the live pedestrian kinematics derived from the mobile phone sensors and transmits the resulting coordinate

and velocity matrices back to the ego-vehicle's onboard computer over a dedicated UDP port. This data is then used to update the local CARLA environment to ensure that the AV (or ADAS) systems can detect, follow, and interact with the pedestrian.

2.12 Virtual Reality Headset and Environment Setup

The final step for the creation of the VVE system lies in immersing the human pedestrian in the simulated world using the VR interface. The VR experience can be set up by installing and configuring the Meta Horizon Link software on the central processing laptop, which provides a physical tether between the VR headset and the machine's computational power as seen in Figure 10.

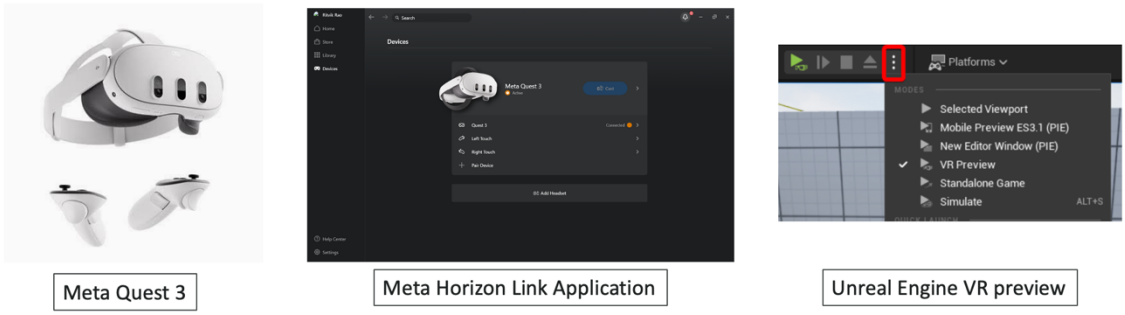

Figure 10: Virtual reality setup

Upon establishing the necessary UDP communication streams and simulating the vehicle dynamics with Simulink, the visual experience can then be initiated from within the Unreal Engine 5 editor itself. The experience will begin once the "VR Preview" mode is selected from the Play menu of the engine, allowing the user to physically interact with the 3D simulation and track the movement of the approaching ego-vehicle.

Chapter 3. Results

This chapter reviews the performance, stability, and latency of the VVE framework. In addition, this chapter conducts a quantitative analysis on network performance and a qualitative review on the visual performance of the pedestrian tracking algorithms within the UE environment.

3.1 Qualitative Video Demonstration

To support the validity of our mathematical models developed in Chapter 2, real world tests have been conducted. The following video presentation shows the visual performance and accuracy, as well as the differences among the three methods of pedestrian detection (Kalman Filter Method, Gait Detection Method, and Constant Velocity Method).

https://youtu.be/4QpeetoKREQ

https://youtu.be/cReppCZcBbc

https://youtu.be/kVRgbitTf2Y

https://youtu.be/9AaHwknT034

https://youtu.be/k4MTBI6zFBc

https://youtu.be/qfJ4TAfEWEs

3.2 Sensor Network Latency Analysis

In all HIL or VVE simulation systems related to AV or ADAS technology, network latency is an important measurement for the efficiency of the system. If the delay of communication between physical pedestrians and virtual ego-vehicles is too large, then collision avoidance algorithms will not be able to work in time. Time delay compensation methods will then need to be used [27]. For analyzing the performance of LAN UDP bridge, we measured the latency of communication between physical pedestrians and virtual ego-vehicles.

The real-time latency variations experienced by the four major hardware sensors are represented below in Figure 11 over a period of continuous testing lasting 250 seconds.

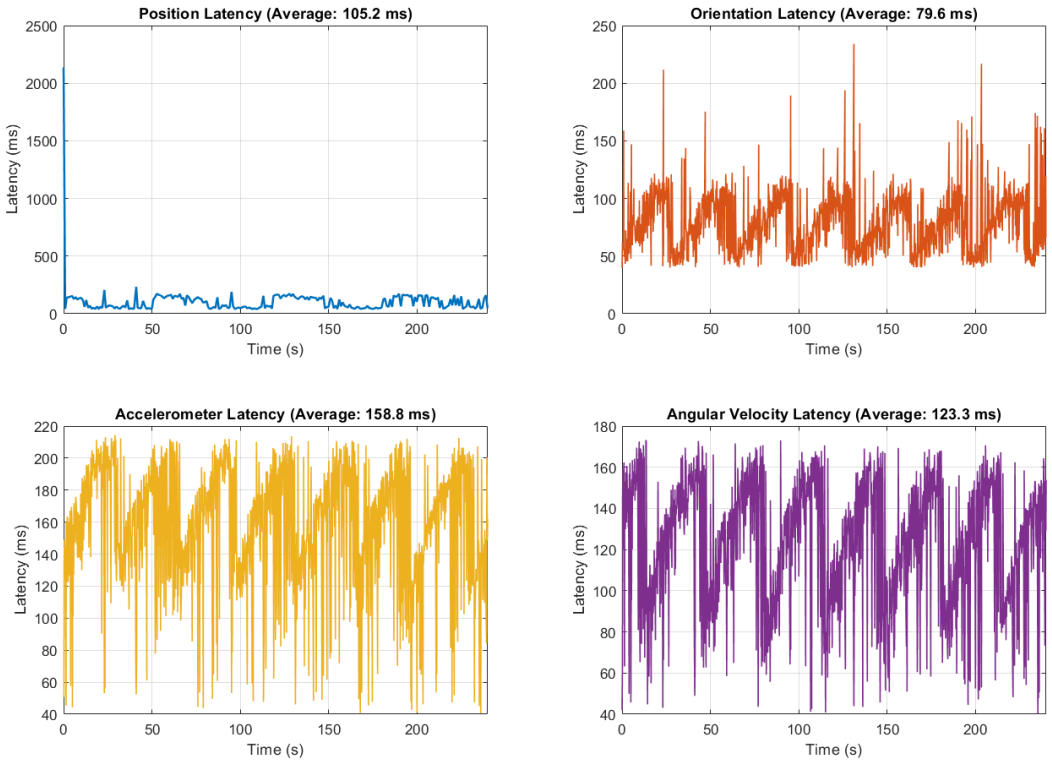

Figure 11: Latency plots for sensors

As depicted in the Figure 11 above, the IMU high-frequency sensors had a clear saw-tooth pattern, which is expected due to the nature of the batch transmission of the 100 Hz data packets. The sawtooth pattern repeats at approximately every 35 sec. On the other hand, the Position sensor displayed a steady state following an initial connection peak.

To provide a comprehensive statistical breakdown of this network performance, the latency data was aggregated and analyzed. An average latency of 116.7 ms was observed for all transmitted data streams. The results for each individual sensor can be seen in Table 2.

Table 2: Latency statics

Sensor Type	Average (ms)	Median (ms)	Std. Dev. (ms)	Min (ms)
Position	105.2	72.6	139.7	40.2
Orientation	79.6	77.6	26.7	40.0
Accelerometer	158.8	166.3	38.6	40.6
Angular Velocity	123.3	129.4	31.3	40.3

3.2.1 Analysis of Latency Discrepancies

Some interesting insights can be noted based on the statistical summary. The first thing that should be mentioned is that the Orientation (Yaw) sensor was the fastest, registering the smallest mean latency of 79.6 ms, as well as the smallest standard deviation of 26.7 ms. These better results confirm the choice made in Section 2.7 to utilize mostly the yaw data when applying the constant-velocity tracking model due to the instant changes in the direction.

Secondly, there is a large outlier of the maximum latency of the Position (GPS) sensor, which is 2137.0 ms. Such a huge delay is explained by the time it takes to acquire satellites at the start-up phase of the hardware. Nevertheless, due to its polling rate of just 1.0 Hz its median is very low – only 72.6 ms.

Finally, the average latency for the entire system is equal to 116.7 ms, which is a good result for consumer-level smartphone internet connectivity. It ensures that the virtual pedestrian can replicate the movements of the actual user in nearly true-time mode and, thus, creates a reliable platform for testing collision avoidance mechanisms of the ego-

vehicle. This approach can be used for developing, evaluating and demonstrating AV operation [28].

Chapter 4. Conclusion and Future Work

The primary objective of this research was to contribute to the further development of a robust Vehicle-in-Virtual-Environment (VVE) framework that can effectively link up the physical mechanics of humans with the physical autonomous vehicle motion both synchronized and replicated in a shared virtual environment. The main objective has been achieved through the development of a seamless MATLAB, Simulink, Unreal Engine, and Virtual Reality platform.

Apart from being applied in Human-in-the-Loop scenarios, the framework provides an easily applicable stand-alone platform for vehicle testing. Through the framework, researchers can efficiently use MATLAB and Simulink to control ego-vehicles within the Unreal Engine without having any pedestrian sensors present in operation.

Though this VVE framework has been able to create an effective basis for AV and ADAS function testing, there remain various areas in which the system may be enhanced and expanded in the future.

Future iterations need to include computer vision and pose estimation technology utilizing camera-based skeletal tracking to better detect more complicated poses of vulnerable road users. It would provide an accurate reflection and response to rare cases when VRUs exhibit extreme physical behavior such as jumping, laying down, or performing hand gestures.

There is also a need for improving the algorithm that performs smoothing of micro-stutters by improving the mathematics behind the Kalman filtering method and prediction of movements. It will increase the quality of the simulation by making the animation more life-like.

To remove any network latency from the method, it is recommended to switch to a specialized hardware setup equipped with a set of high-frequency sensors, rather than using a smartphone as a tracking device, thereby eliminating delays associated with mobile platform network batching.

Current UDP network structure allows for a simple but very efficient scaling process, allowing us to expand the simulation environment in a way that multiple pedestrians, vehicles, and even interactive elements can be added at once.

References

[1] H. Chen, X. Cao, L. Guvenc, and B. Aksun Guvenc, "Vehicle-in-Virtual-Environment (VVE) Based Autonomous Driving Function Development and Evaluation Methodology for Vulnerable Road User Safety," *arXiv preprint arXiv:2501.06113*, 2025.

[2] S. Y. Gelbal, S. Arslan, H. Wang, B. Aksun-Guvenc, and L. Guvenc, "Elastic band based pedestrian collision avoidance using V2X communication," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, Los Angeles, CA, USA, 2017, pp. 270-276, doi: 10.1109/IVS.2017.7995731.

[3] S. Y. Gelbal, B. Aksun-Guvenc, and L. Guvenc, "Collision Avoidance of Low Speed Autonomous Shuttles with Pedestrians," *Int. J. Automot. Technol.*, vol. 21, no. 4, pp. 903-917, 2020, doi: 10.1007/s12239-020-0087-7.

[4] X. Cao, H. Chen, S. Y. Gelbal, B. Aksun-Guvenc, and L. Guvenc, "Vehicle-in-Virtual-Environment (VVE) Method for Autonomous Driving System Development, Evaluation and Demonstration," *Sensors*, vol. 23, no. 11, p. 5088, 2023, doi: 10.3390/s23115088.

[5] S. Y. Gelbal, B. Aksun-Guvenc, and L. Guvenc, "Vehicle in Virtual Environment (VVE) Method," Automated Driving Lab, Ohio State University, Columbus, OH, Tech. Rep.

[6] L. Guvenc, B. A. Guvenc, B. Demirel, and M. T. Emirler, *Control of Mechatronic Systems*. London, UK: IET, 2017, ISBN: 978-1-78561-145-2.

[7] H. Zhou, F. Jia, H. Jing, Z. Liu, and L. Guvenc, "Coordinated Longitudinal and Lateral Motion Control for Four-Wheel Independent Motor-Drive Electric Vehicle," *IEEE Transactions on Vehicular Technology*, vol. 67, pp. 3782-3790, 2018, doi: 10.1109/TVT.2018.2816936.

[8] H. Wang, A. Tota, B. Aksun-Guvenc, and L. Guvenc, "Real-Time Implementation of Socially Acceptable Collision Avoidance of a Low-Speed Autonomous Shuttle Using the Elastic Band Method," *Mechatronics*, vol. 50, pp. 341-355, 2018, doi: 10.1016/j.mechatronics.2017.11.009.

[9] L. Zhenyu, P. Lin, Z. Konglin, and Z. Lin, "Design and evaluation of V2X communication system for vehicle and pedestrian safety," *The Journal of China Universities of Posts and Telecommunications*, vol. 22, no. 6, pp. 18-26, 2015.

[10] T. Hwang, J. P. Jeong, and E. Lee, "SANA: Safety-Aware Navigation App for Pedestrian Protection in Vehicular Networks," in *IEEE International Conference on Information and Communication Technology Convergence (ICTC)*, 2014.

[11] B. Aksun-Guvenc and L. Guvenc, "Robust two degree of freedom add-on controller design for automatic steering," *IEEE Transactions on Control Systems Technology*, vol. 10, no. 1, pp. 137-148, 2002.

[12] X. Wang, S. Mao, and M. X. Gong, "An Overview of 3GPP Cellular Vehicle-to-Everything Standards," *GetMobile: Mobile Computing and Communications*, vol. 21, pp. 19-25, 2017.

[13] M. Janeba, P. Lehoczký, M. Galinski, T. Milesich, J. Danko, and I. Kotuliak, "Evaluation of LTE and 5G Qualitative Parameters for V2X Use Cases," in *2022 IEEE Zooming Innovation in Consumer Technologies Conference (ZINC)*, Novi Sad, Serbia, 2022, pp. 165-169.

[14] M. S. Ahmed, M. A. Hoque, and A. J. Khattak, "Demo: Real-Time Vehicle Movement Tracking on Android Devices through Bluetooth Communication with DSRC Devices," in *2016 IEEE Vehicular Networking Conference (VNC)*, Columbus, OH, USA, 2016, pp. 1-2.

[15] M. T. Emirler, H. Wang, B. Aksun-Guvenc, and L. Guvenc, "Socially Acceptable Collision Avoidance System for Vulnerable Road Users," *IFAC-PapersOnLine*, vol. 49, no. 3, pp. 436-441, 2016.

[16] N. Verma, G. Kumar, A. Siddhant, P. Nama, A. Raj, and A. Mustafa, "Vision based obstacle avoidance and recognition system," in *IEEE Workshop on Computational Intelligence: Theories, Applications and Future Directions (WCI)*, Kanpur, India, 2015.

[17] S.-Y. Wang, Y.-W. Cheng, C.-C. Lin, W.-J. Hong, and T.-W. He, "A vehicle collision warning system employing vehicle-to-infrastructure communications," in *IEEE Wireless Communications and Networking Conf.*, Las Vegas, NV, USA, 2008.

[18] World Health Organization (WHO), "Decade of Action for Road Safety 2011-2020: Saving Millions of Lives," 2011.

[19] M. R. Cantas, O. Kavas, S. Tamilarasan, S. Y. Gelbal, and L. Guvenc, "Use of Hardware in the Loop (HIL) Simulation for Developing Connected Autonomous Vehicle (CAV) Applications," in *SAE World Congress Experience*, 2019.

[20] T. Bokc, M. Maurer, and G. Farbe, "Validation of the Vehicle in the Loop (VIL); A milestone for the simulation of driver assistance systems," in *IEEE Intelligent Vehicles Symposium*, 2007.

[21] L. Wang, W. Huang, X. Liu, and Y. Tian, "Vehicle collision avoidance algorithm based on state estimation in the roundabout," in *Int. Conf. Intelligent Control and Information Processing*, Dalian, China, 2012.

[22] CARLA Simulator. [Online]. Available: https://carla.org/

[23] MathWorks. "MATLAB Mobile." MathWorks Products. [Online]. Available: https://www.mathworks.com/products/matlab-mobile.html

[24] MathWorks. "mobiledev: Acquire data from mobile device sensors." MATLAB Documentation. [Online]. Available: https://www.mathworks.com/help/matlab/ref/mobiledev.html

[25] C. Montella, "The Kalman Filter and Related Algorithms: A Literature Review."

[26] MathWorks. "Vehicle Dynamics Blockset." MATLAB Documentation. [Online]. Available: https://www.mathworks.com/products/vehicle-dynamics.html

[27] Cao, X., Chen, H., Guvenc, L., Aksun-Guven, B., 2025, "Communication Disturbance Observer Based Delay Tolerant Control for Autonomous Driving System," Sensors, Special Issue on Sensor-Based Control and Navigation for Autonomous Vehicles, Vol. 25, pp. 6381.

[28] Bowen, W., Gelbal, S.Y., Aksun-Guvenc, B., Guvenc, L., 2018, "Localization and Perception for Control and Decision Making of a Low Speed Autonomous Shuttle in a Campus Pilot Deployment," SAE International Journal of Connected and Automated Vehicles, doi: 10.4271/12-01-02-0003, Vol. 1, Issue 2, pp. 53-66.

Appendix A. Link to all Simulink and MATLAB files

This is the link for all the MATLAB and Simulink code files:

https://github.com/Ritv1k/ASimulationFrameworkForPedestrian.git