A Mixed-Reality Simulator for an Autonomous Delivery System Using Platooning

Prem Chand Pavani
CIAD (UMR 7533)
Univ. de Technologie de Belfort-Montbéliard
Belfort, France
email: prem-chand_pavani@etu.u-bourgogne.fr

Pierre Romet
CIAD (UMR 7533)
Univ. de Technologie de Belfort-Montbéliard
Belfort, France
email: pierre.romet@utbm.fr

Franck Gechter
CIAD (UMR 7533) and MOSEL LORIA (UMR CNRS 7503)
Univ. de Technologie de Belfort-Montbéliard and Univ. de Lorraine
Belfort and Vandoeuvre, France
email: franck.gechter@utbm.fr

El-Hassane Aglzim
ISAT-DRIVE
Univ. Bourgogne Franche-Comté
Nevers, France
email: el-hassane.aglzim@u-bourgogne.fr

***Abstract*—Developments in the field of autonomous vehicles have encouraged research to innovate technology to solve everyday problems. E-commerce has been on the rise and, freight transportation is considered an environmental nuisance, especially in the city centers. Electric vehicles have been proposed to reduce the environmental impact of transit vehicles. A package delivery system using a platoon of autonomous electric delivery vehicles and established public transport networks in cities can be employed to solve these problems. But autonomous vehicle testing is a point of concern for authorities and the public alike. This paper acknowledges this problem of validating the algorithms used to create an autonomous delivery system using an innovative solution. A Mixed-Reality simulator based on Unity3D and Robotic Operating System was successfully created to test autonomous vehicle platooning.**

***Keywords*—Mixed-Reality; Platooning; Autonomous Vehicles; Vehicle-Hardware-in-the-Loop (VeHiL).**

I. INTRODUCTION

For several years, a significant effort has been made to optimize the transport of goods in urban and peri-urban centers. These improvements are aimed at reduction of the secondary effects of transportation, such as congestion, noise pollution due to the dense traffic flow, or air pollution due to conventional vehicles. Current legislative regulations limit the size and weight of transport vehicles, delivery hours during the day, and suggestions to construct central distribution units close to the city. Other changes include the addition of distribution circuits for tricycles or compact electric vehicles. Nevertheless, these new guidelines and distribution routes are far from reducing the negative impact of transit vehicles circulating in space dedicated to the public. Gechter et al. [1] discusses the solutions available to improve the fright transportation in the city center and proposes a comprehensive model to use the public transportation system and autonomous subnormal-sized electric transits by forming a platoon. Gechter et al. [1] clearly defines the advantages of using platoons of autonomous vehicles that serve as the base for this paper which is part of the project SURATRAM (Système Urbain et Rural Autonome de TRAnsport de Marchandises).

Electric freight vehicles have been at the forefront of combating climate change due to excess production of carbon dioxide. Electrification makes the most sense in an urban environment for short commutes where the combustion engine is the most inefficient. Most cities have separate lanes for public transport that trace the entire city perimeters. Large freight vehicles, which are not nimble on narrow city roads, can use these routes to reduce the congestion. Programming a fleet of autonomous robots to follow the existing public transport entities on these less-used routes by using platooning is a concept that can prove advantageous. Autonomous vehicles require extensive testing and, current simulation tools cannot replicate real-world conditions with a hundred percent accuracy and do not account for few critical or complex scenarios. On the other hand, on-road testing is either forbidden or limited by law in most countries. To address issues of autonomous vehicle testing, we are developing a Mixed-Reality (MR) simulator to validate the use-case of platooning and the algorithms.

Kalra et al. [2] demonstrates that autonomous vehicles would need tests over 14 billion kilometers of on-road testing that could take over 400 years with a fleet of 100 agents running every single hour of the year with a supervisor in the vehicle. Hussein et al. [4] introduces the framework used in simulation using Robotic Operating Software (ROS) and Unity3D to optimize experimentation using smart vehicles. A MR platform based on UDP communication was built using the AIM simulator and an autonomous vehicle to test scenarios at an intersection [5]. AIM does not provide a realistic image of the natural world and has limited application. Another MR simulator was created in [7] using Gazebo and ROS to achieve interaction between simulated and real objects while updating the simulation according to the movement of the robot in the real world. The usage of many commercially available robots is simplified using Gazebo as they are integrated into the simulator. But Unity3D has a more complex and adaptable physics engine compared to Gazebo, giving better realism during testing. Simulation of other elements like traffic can be easily programmed to recreate real-life scenarios. Unity3D

Copyright (c) IARIA, 2021. ISBN: 978-1-61208-898-3

also allows training of our driving models using Machine Learning algorithms which is not possible using Gazebo.

Analyzing the convenience of use and the verisimilitude of the simulation to the real world, we propose an MR simulator for our autonomous delivery solution using platooning based on Unity3D and ROS to test our autonomous driving algorithms. The paper is organized as follows: Section II consists of the present technology of simulators used in the literature, while Section III details the MR simulator framework. Section IV provides the results and analysis of our proposal, and Section V concludes the presentation with improvements and the future scope of the project.

II. State of the Art

As we climb up the various levels of driving automation for on-road vehicles defined by the Society of Automotive (SAE), the complexity of the systems keeps growing. Complex systems include dozens of Electronic Control Units and sensors [8]. Integration of these components becomes challenging, and the necessity of alternative methods to on-road testing becomes evident. Currently, automotive manufacturers are working extensively with XiL in the system development cycle to increase productivity [3].

The first step is to develop the actual model of the plant or hardware in a simulation environment that represents the influential features of the system [9]. A controller is conceived to alter the output of the plant as per the application. This method is known as Model-in-the-Loop (MiL) testing. The behavior of the simulated plant model is governed by the controller logic. MiL is a good starting step to perform controlled tests of the system in a virtual world. When a MiL produces satisfactory results, a code generated from only the controller model replaces the controller block. We simulate using the controller block made of the code with the software model from the previous step [10]. This process is known as Software-in-the-Loop (SiL) testing. The outcome of the test is compared with those obtained in the MiL testing. We alternate these two steps until a reliable algorithm is produced. It is still important to test the controller on the real hardware as the simulated model is based on certain important parameters. Hence in the last step, the simulation model is replaced, entirely or partially, by the actual hardware to test the accuracy of the controller logic, hence the name Hardware-in-the-Loop (HiL) testing [11]. Another class of testing depends on the use of a prototype in the development cycle. The prototype is examined in real-life conditions (like test tracks or on-road) to have the best results. A prerequisite to this is the need for extensive infrastructure. There are several limitations to the on-road testing from the government due to safety concerns. A hybrid that combines the HiL and prototype testing is Vehicle-Hardware-in-the-Loop (VeHiL) that allows manufacturers to test their approach in the initial stages of the development. The method is flexible and convenient as we can moderate all the environmental conditions. The tests are conducted on a chassis dynamometer which is usually heavy and requires plenty of space [3].

User interactive maps using Augmented Reality (AR) (projection of virtual elements in the real world [6]) gives us an immersive experience during navigation through streets. Another impressive technological advancement is the ability to explore the world in Virtual Reality (VR), where we can interface with the virtual elements or create a virtual world. A composite of these technologies is MR, where the physical entities can interact with elements in the virtual world [6]. Integration of MR in autonomous vehicle testing provides versatility, increasing the duplicability of real-world conditions in confined spaces. MR has been used previously to provide driving assistance of welfare vehicles using a virtual platoon control method allowing novice users to control the kart with ease [12].

Given the state-of-the-art, we understand that VeHiL testing is advantageous but requires an actual vehicle or prototype in the testing loop alongside a simulation, as defined in [3]. We built our VeHiL around a Radio Controlled (RC) car to reduce the infrastructure requirement and allow flexible testing of our platooning application using the MR simulator. Developing algorithms using a 1:1 model of the bus and vehicle is financially inconvenient due to costs of fuel and renting a bus. Large vehicles require space for testing. The nearest testing ground is 30kms away from the laboratory. Travelling to the site to validate small changes in the algorithms or ideas is logistically cumbersome and not time efficient. A detailed explanation is provided in the next section.

III. Mixed Reality Simulator

Evaluating the advantages and disadvantages of the available testing methods, we propose a VeHiL test using an MR simulator using Unity3D and ROS. We will use Unity3D to create a platoon simulation using a bus (representing the public transport) and a Hyundai Kona (representing an electric transit van). Simulation test cases include the car following the bus, parking the vehicle on the side of the road at bus stops to avoid congestion, and dynamically attaching to a bus on a different route to the present one at an intersection to deliver at a specific location. Using the schedule data provided by the public transport provider in Belfort, Optymo, we can synchronize this change of route fluidly. In this paper, we discuss only the platooning test case. We will start our VeHIL tests using a scaled model of the car, in the form of an RC robot equipped with sensors like LiDAR, Ultrasonic Distance Sensor (UDS), Inertial Measurement Unit (IMU). We use ROS to manage data and control the robot. A schematic with the framework of the MR simulator is presented in Figure 1. A comprehensive description of each component is provided in the upcoming few subsections.

A. Smart Robot

The MR simulator is based on an electric Kona, which would act as our autonomous delivery vehicle. During the VeHIL test phase, we did not have the Kona at our disposal. To proceed with the development of the simulator, we used an RC car to replicate the characteristics of the Kona. Hence, the

Copyright (c) IARIA, 2021. ISBN: 978-1-61208-898-3

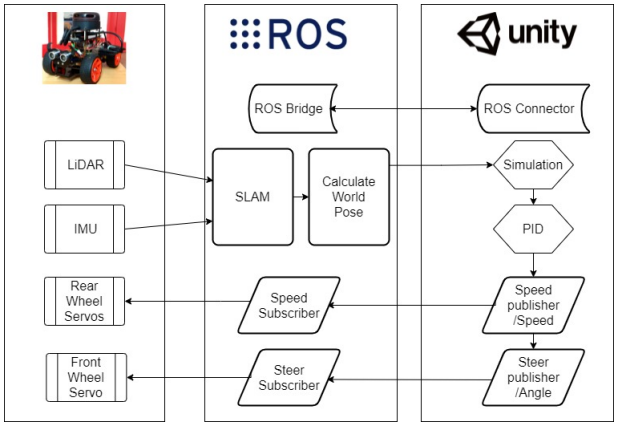

Figure 1. Mixed-Reality simulator framework.

results obtained are not entirely homologous to experiments performed with a Kona due to a mismatch of the physics model of the robot and simulation. We normalized the parameters of the Kona, like torque and turning radius, in the simulation to suit the handling of the robot. Using the PiCar-S kit V2.0 from the robotics company SunFounder, we were able to customize the robot by adding a LiDAR and an IMU, in addition to the UDS to replicate the functionality of the actual vehicle (Figure 2).

Out of the box, the PiCar depends on a Raspberry Pi (RPi) 4B for its computational power running Ubuntu (based on Linux; officially supported by ROS). The propulsion system constitutes of two pulse width modulation (PWM) motors. A 2D LiDAR from SLAMTEC (RPLIDAR A2) used in the project produces up to 8000 samples/second and has a range of 12m with a resolution of 0.15m. A MEMS-based IMU (MPU6050) fuses the programmable 3-axis accelerometer (2g-16g), and 3-axis gyroscope (250°/sec-2000°/sec). An UDS with a range of 0.02m-4m works in the frequency range of 40kHz. The manufacturer provides a library for the RC car, coded in Python that allows users to easily set up the robot.

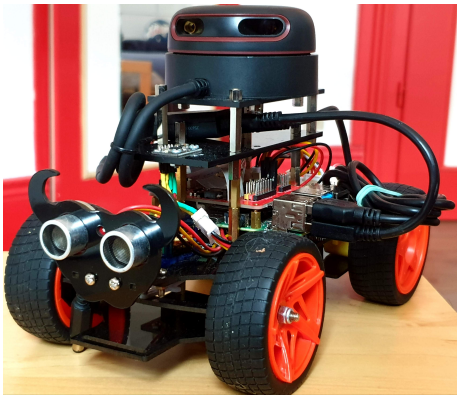

Figure 2. Smart robot equipped with a LiDAR, an IMU and an UDS.

B. Unity3D Simulation

To perform the MiL test, we chose Unity3D to run our initial experiments. The simulation primarily consists of two objects, a bus, and a car. The simulation aims to reproduce a platoon that includes a public transport entity (in our case, the bus) and an electric delivery van (Figure 3). The bus will act as the leader of the platoon, with the car behaving as the follower. The bus follows a predefined path, and the car can autonomously follow the bus based on the telemetry data of the leader. The data collected from the follower vehicle is transmitted to the robot. A feedback loop updates the location of the car in the simulation per the real world. The bus is a representative model of Lion's city hybrid buses used in the city of Belfort, manufactured by MAN. The car is comparable to a Kona electric, produced by the Korean manufacturer, Hyundai. We modeled a Kona electric in the simulation as we have the actual vehicle, fitted with a RADAR, two LiDAR's (one frontal, one on the roof), and a Global Navigation Satellite System with an integrated IMU. Unity3D has a configurable physics engine that can adapt to the vehicle using parameters like mass, the center of gravity, or the drag coefficient. Unity3D also allows users to incorporate the tire model by providing the forward friction and sideways friction values (extremum slip, asymptote slip). The simulation also accounts for the unsprung mass and suspension system (damping rate, suspension distance, force application point distance). The maximum torque values and speed limits of both vehicles are programmed. These features can help program physics of most vehicles in Unity3D, making the simulation versatile to test other platooning projects.

Using waypoints, we set the path for the bus to follow. At the beginning of the simulation, we congregate all the waypoints and store them in an array to keep track of the number of points. To steer the bus, we calculate a relative vector to the upcoming waypoint from the current position of the bus; this returns a value between [-1, 1], indicating the direction (negative value implies that the point is to the left of the heading of the bus and on the contrary, a positive value indicates a point on the right). To determine the steering angle, we multiply the relative vector with the maximum steering angle. When the bus is within 5m of the current waypoint, we calculate the steer value to the next waypoint in the array. To control the speed of the bus, we apply a torque to the rear wheels of the bus corresponding to the distance to the next waypoint and reduce this value as we approach the waypoint or if we must navigate a sharp turn. We update our steering and torque values at a fixed period of 20ms or at a frequency of 50Hz. A similar method is deployed on the follower (Kona) to allow the car to move relative to the leader. Analogous to the bus, we use a tracker point placed on the rear axle of the bus to determine the angle of steering required on the car. A Proportional-Integral-Differential (PID) controller (Gain values P: 80, I: 30, D: 35) adjusts the speed of the Kona proportionately to the distance between the two vehicles. The distance is measured from the

Copyright (c) IARIA, 2021. ISBN: 978-1-61208-898-3

front of the Kona to the tracker point. The aim is to maintain a safe distance of 10m which accounts for emergency braking. The PID controller gain values can be updated during the simulation. The PID controller recalculates the speed of the Kona every 100ms (10Hz) by adjusting the error at the rate of 50Hz increases the data queue to be transmitted significantly. As the simulation does not emulate the physics of the robot, a novel PID controller (Gain values P:10, I:7, D:6) and a separate code for steering were adapted to the pace and the turning radius of the robot.

The company Siemens, developed an open-source library in C# to communicate with ROS from .NET applications like Unity using TCP/IP sockets (available on GitHub as ros_sharp). The library includes standard message publishers, subscribers, and Action servers. We created a new object that contains a ros_connector, two data publishers, and a Pose stamped subscriber. The ros_connector script helps us connect to the ROS server running on the robot. We modified two 32-bit float (std_msgs ROS) publishers to transmit the speed and steering angle. We adopted a pose stamped subscriber from the ros_sharp library to receive the coordinates of the robot generated by the Simultaneous Localization and Mapping (SLAM) algorithm.

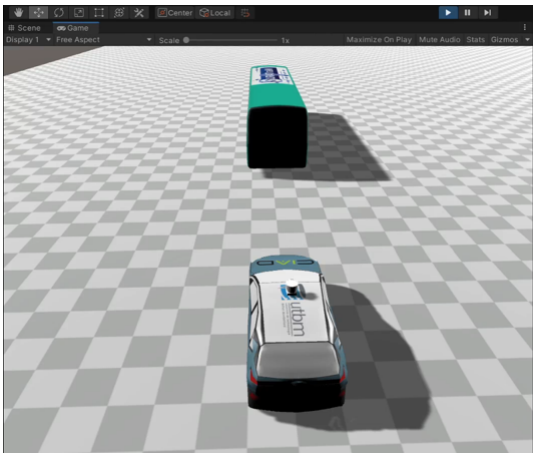

Figure 3. Simulation of the platoon in Unity3D.

C. ROS

ROS acts as a middleware to manage the data generated by different sensors and helps various programs running on the robot communicate. It also behaves like a control mechanism for the robot by collecting the data from the simulation. We used ROS Noetic Ninjemys that has good community support and compatibility with the packages required for this project. The first package that is vital is the rosbridge_suite library. The library contains a rosbridge_server package with rosbridge_websocket launch file that creates a server with the IP address of the RPi. A simple listener coded in Python can subscribe to the /speed and /steer topics and receive the 32-bit float messages from Unity3D over the server. Using a modified library provided by SunFounder, we can manipulate the robot according to the received data. The LiDAR point-cloud data is visualized in RVIZ using the package provided by SLAMTEC. This point cloud is accessible from the topic /slam. We used a Python script to decode data from the MPU6050. The linear acceleration and angular velocity values are published using the topic /imu. Next, to fuse the data from the IMU and the LiDAR to form a 2D map of the environment, three SLAM packages are currently available: gmapping [9], Cartographer [10], and Hector SLAM [11]. In our case, we are fusing data from LiDAR and IMU, so hector_slam seems to be the best choice. The SLAM module, based on an Extended Kalman Filter (EKF), generates coordinates of the robot's location in the real world, published using the topic /slam_out_pose. We relay the position to Unity3D via the ROS server, where a Pose Stamped (geometry_msgs ROS) subscriber node converts the data into Unity3D coordinates.

IV. Experiments & Results

An oval path (L1: 30 units, L2: 20units) is drawn for the bus to trace using waypoints in Unity3D. Each experiment corresponds to the bus completing one full revolution (unless specified) of the oval while being followed closely (10m distance) by the car in the simulation. This section will provide details regarding the analysis and results of the performance of our MR simulator, based on three main criteria: time delay, deviation of simulation from the real-world position, and analysis of the SLAM algorithm.

A. Time Delay

As we are working on a real-time system, it is crucial to determine the time delay between the transmission of data, the actuation of the system, and the feedback. Anticipating the delay can improve the efficiency of control strategies. A ping of 20ms is commonly observed in wireless connections working at 2.4GHz but, the delay varies for each query. ROS is known to have delays in communication between nodes, which can influence the delay in the system. We identified three components that make up the total delay in our system include: messages to reach ROS from Unity; the time necessary to process the data; update world pose data from ROS to Unity.

1) From Unity to ROS: The pose stamped messages (geometry_msgs ROS) were sent from Unity using the ros_sharp library to understand the delay during transmission of data from Unity to ROS. The data is converted to suit the coordinate system in ROS (right-handed, with X forward, Y left, and Z up) as Unity uses a left-handed, Y-Up, Z-forward, and X-left convention. The pose message header contains the time at which the data was generated during the simulation. When the data reaches the ROS subscriber node, the information is parsed, and the output is printed on the command window with time in the Unix format (nanoseconds). The session gets recorded into a rosbag file and saved in a CSV file format. Using a python script, timestamps of the sent and received messages are separated from the CSV file and converted into readable date and time format. Matlab is used to import the data and construct a graph describing the delay for each packet

Copyright (c) IARIA, 2021. ISBN: 978-1-61208-898-3

of data to reach ROS from Unity over the number of frames. For each frame, we generate one message.

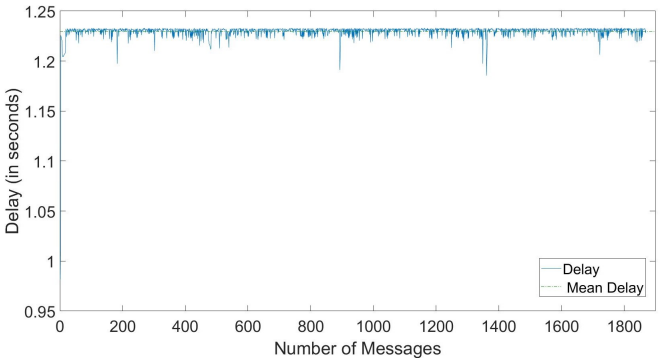

Figure 4. Delay of each message received in ROS from Unity3D.

Tested over 1870 messages (at 50Hz Frame rate), we observe that the first message arrives in ROS after 0.98 seconds, after which the delay for each consecutive packet of data fluctuates between 10ms-20ms (Figure 4). So, the robot is lagging the simulation by 1.3 seconds. The rosbridge socket server is CPU intensive and takes time to process substantial amounts of inbound data. Since the ROS server and simulation are running on different machines, the latency increases significantly. The variation in delay between messages is known as jitter. This is an issue of using the 2.4GHz band as most appliances use the same frequency range along with other people living in the neighborhood, causing significant variations in delay between queries.

2) Processing Delay: Every processor takes time to make sense of the received data. We would like to measure the delay between the time when the message is received in ROS to when the robot starts moving. This is measured to show the worst-case delay observed in the system moving from complete rest. Two separate loops are used to process the speed and steering angle data received in ROS, increasing the processing time of the data. When a message is received, the content of the message and the time get printed on the command window. The IMU is sensitive enough to detect every movement. A message with the current time is printed on the command window once there is a change in acceleration in the X-axis above a threshold value of 0.2g. The difference between the timestamp of the first message received from Unity and the timestamp when there is motion provides the time required for processing.

Over nine trials performed to determine the delay between the reception of command and actuation of the servo motor - we observe a mean delay of 1.5 seconds (Figure 5). This loss in time can be attributed to the time delay for the first message to be received in ROS, which is around 0.98 seconds, as mentioned in the previous experiment. So we can deduce that the processing delay is approximately 0.5 seconds which can be due to the sensitivity of the motors to low-speed inputs at the start.

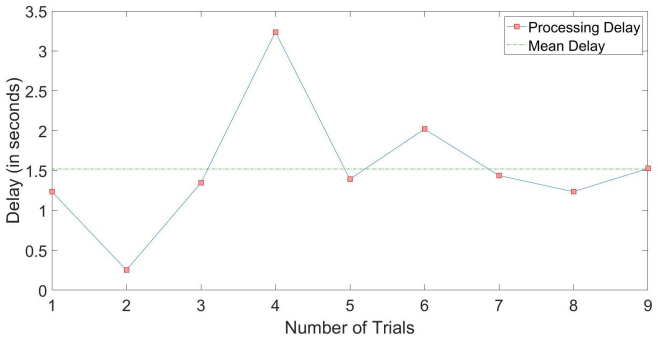

Figure 5. Delay between the reception of message and the robot moving.

3) From ROS to Unity: Similar to the delay observed during the relay of data from Unity to ROS, there is a delay during the transmission of data from ROS to Unity3D over the wireless network. We would expect the delay to be around 0.02seconds. To estimate this delay, we generate a Pose Stamped message after the world position of the robot is calculated by the SLAM module and relay this data to Unity. The pose-stamped data from ROS is converted to suit the coordinate system in Unity by the subscriber node in Unity3D. The pose stamped messages contain the timestamp representing the time when they were generated in ROS. Once the message is received in Unity3D, we log the current time. These logs can be accessed from the player log editor. The timestamps are imported into Matlab, and we create a graph between the number of frames and delay in seconds.

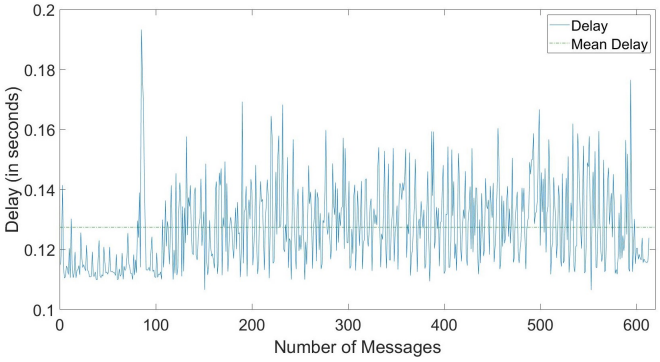

Figure 6. Delay of each message received in Unity3D from ROS.

A comparison between the timestamps of the data reveals that - on average, there is a delay of 0.13 seconds. It varies between 0.11-0.19 seconds (Figure 6). The obtained delay is significantly higher than expected. In addition to the jitter observed while using the 2.4GHz frequency band, we have a delay during the exchange of data between the node responsible to publish the pose stamped messages and the WebSocket server in ROS. Since the server is running on the RPi and the subscriber on the computer, we have a latency between the two host machines and is a factor in the high delay values observed.

Copyright (c) IARIA, 2021. ISBN: 978-1-61208-898-3

B. Deviation from actual World Pose

Since the physics model of the simulated entity and the robot are different, it is interesting to see how this difference affects our simulation. We will be able to judge if there is a need to mimic the physics of the actual vehicle in the simulation. The coordinates produced by SLAM module are used to update the simulation. Another object that contains a pose stamped subscriber is created in Unity3D. This object represents the path traced by the robot in the real world. A vector between the path taken by the virtual car and the path traced by the robot is recorded every second in the simulation.

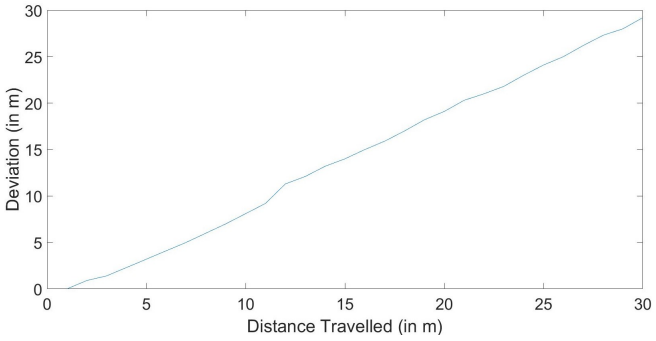

Figure 7. Deviation of simulation from actual world position of the robot.

A comparison between the simulated position and the coordinates of the robot shows a deviation of one meter per every meter of movement in the simulation (Figure 7). At the start of the simulation, the deviation is relatively less as the speed of the vehicles is low. As proven earlier, the RC car lags the simulation by more than a second. Combined with the fact that Unity currently uses a physics model of a car, there is a significant deviation of the path reproduced by the robot in comparison to the virtual car. This result proves the need for an accurate physics model of the virtual vehicle with respect to the real vehicle.

C. Accuracy of SLAM

The closeness of the location provided by the SLAM algorithm to the ground truth will increase our confidence in our measurements, enabling us to safely navigate in densely populated urban environments. To test the accuracy of SLAM, a grid (1 cm^2) paper is laid out on the floor. The RC car is positioned on one of the edges of the grid cell. We run one cycle of the simulation while filming the robot from a bird's eye view. The position data from the SLAM is recorded into a rosbag file. We note the position of the robot from the video by counting the number of cells traveled on the grid at a period of three seconds and the coordinates given by SLAM in ROS.

We plot the two points to recreate the path followed by the robot in the real world and according to SLAM (Figure 8). The SLAM can localize the robot with good accuracy fusing the LiDAR and IMU data. The robot is not able to recreate the oval circuit drawn in Unity3D due to the difference in physics model. Optimizing speed and angle of steer produces a semicircular movement over three revolutions

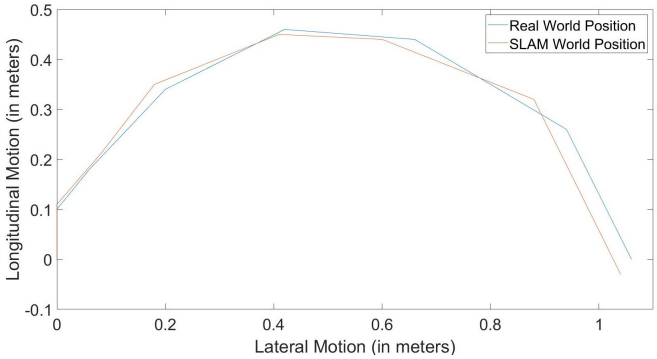

Figure 8. Comparison of position provided by SLAM to actual position.

in Unity, contrary to the other experiments. We could not achieve enough points to evaluate the SLAM with a single revolution of the vehicle in the simulation.

Another graph to depict the deviation over time is plotted by calculating a resultant vector between the two points (Figure 9). We see a mean error of 0.03m between both measurements. The error does not accumulate with time due to the presence of an EKF in the SLAM algorithm, which can predict the trajectory of the robot, reducing the deviation from the actual position. The error can be further reduced by fusing data from an odometer.

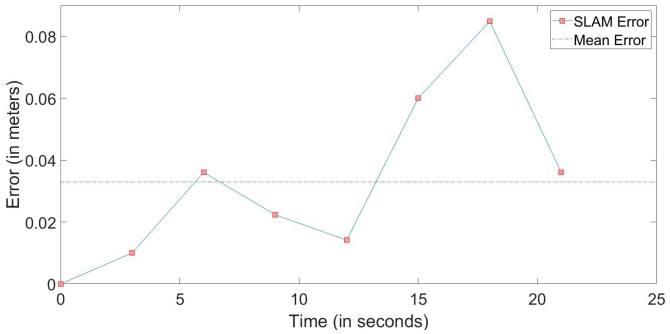

Figure 9. Error between SLAM and actual position.

V. Conclusion and Future Work

As the use of autonomous vehicles becomes more prominent, innovations in the field of testing become critical. This paper evaluated the adoption of an MR simulator with a VeHiL testing to validate algorithms for an autonomous freight delivery system using the public network system proposed in [1]. We can address a few issues to improve the precision of the simulator. First, the use of a 5GHz wireless communication can reduce jitter during data transmission. To reduce the deviation between the virtual and real-world, we can introduce a Kalman filter in Unity3D. Extending the project to support ROS2 can lower the delay caused due to communication between different nodes, as nodes can communicate directly with each other without the need of a ROS Master. The PID controller can be replaced with more robust controllers to

Copyright (c) IARIA, 2021. ISBN: 978-1-61208-898-3

improve follower's capability to keep up with the leader. The fusion of odometer data from the real vehicle can increase the accuracy of the SLAM algorithm. Future research to implement the simulator with the Kona in the VeHiL phase instead of the robot is envisioned, along with testing of other test cases mentioned in Section III. This will avoid issues due to the contrast of the simulation model with the robot. The simulator can be extended to other platooning projects where, researchers can model their vehicles (using the physics model parameters as indicated in Section III: C) in Unity3D along with 3D maps of cities to test their self-driving robots in small spaces, reducing risk of on-road testing and cost of expensive testbeds.

ACKNOWLEDGMENT

This work is done with the support of the Région Bourgogne-Franche-comté, through the SURATRAM Project.

REFERENCES

[1] F. Gechter,et al. "Transportation of Goods in Inner-City Centers: Can Autonomous Vehicles in Platoon Be a Suitable Solution?" in 2017 IEEE Vehicle Power and Propulsion Conference (VPPC), pp. 1–5.

[2] N. Kalra, and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" in 2016 Transportation Research Part A: Policy and Practice , pp. 94, 182–193.

[3] C. Galko, R. Rossi, and X. Savatier, "Vehicle-Hardware-In-The-Loop system for ADAS prototyping and validation", in 2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV), pp. 329–334.

[4] A. Hussein, F. Garcia, and C. Olaverri-Monreal, "ROS and Unity Based Framework for Intelligent Vehicles Control and Simulation," in 2018 IEEE International Conference on Vehicular Electronics and Safety, pp. 1–6.

[5] M. Quinlan, Tsz-Chiu Au, J. Zhu, N. Stiurca, and P. Stone, "Bringing simulation to life: A mixed reality autonomous intersection," in 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 6083–6088.

[6] W. Honig, et al. "Mixed reality for robotics," in 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 5382–5387.

[7] I. Y.-H. Chen, B. MacDonald, and B. Wunsche, "Mixed reality simulation for mobile robots," in 2009 IEEE International Conference on Robotics and Automation, pp. 232–237.

[8] S. Moten, F. Celiberti, M. Grottoli, A. van der Heide, and Y. Lemmens, "X-in-the-loop advanced driving simulation platform for the design,development, testing and validation of ADAS," in 2018 IEEE Intelligent Vehicles Symposium (IV), Changshu, Jun. 2018, pp. 1–6.

[9] A. R. Plummer, "Model-in-the-Loop Testing" Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering in 2006;220(3): pp. 183-199.

[10] S. Demers, P. Gopalakrishnan and L. Kant, "A Generic Solution to Software-in-the-Loop," MILCOM 2007 - IEEE Military Communications Conference, 2007, pp. 1-6.

[11] W. Deng, Y. H. Lee and A. Zhao, "Hardware-in-the-loop simulation for autonomous driving," 2008 34th Annual Conference of IEEE Industrial Electronics, 2008, pp. 1742-1747.

[12] N. Matsunaga, R. Kimura, H. Ishiguro, and H. Okajima, "Driving Assistance of Welfare Vehicle with Virtual Platoon Control Method which has Collision Avoidance Function Using Mixed Reality," in 2018 IEEE International Conference on Systems, Man, and Cybernetics, pp. 1915–1920.

[13] G. Grisetti, C. Stachniss, and W. Burgard, "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters," IEEE Transactions on Robotics, pp. 34-46.

[14] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-Time Loop Closure in 2D LIDAR SLAM," in Robotics and Automation (ICRA), 2016 IEEE International Conference on. IEEE, pp. 1271–1278.

[15] S. Kohlbrecher, et al. (2014) "Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots". RoboCup 2013: Robot World Cup.

Copyright (c) IARIA, 2021. ISBN: 978-1-61208-898-3