ROS and Unity Based Framework for Intelligent Vehicles Control and Simulation

Ahmed Hussein[1], Fernando García[1] and Cristina Olaverri-Monreal[2] Senior Member IEEE

***Abstract*— Intelligent vehicles simulations are utilized as the initial step of experiments before the deployment on the roads. Nowadays there are several frameworks that can be used to control vehicles, and Robot Operating System (ROS) is the most common one. Moreover, there are several powerful visualization tools that can be used for simulations, and Unity Game Engine is on the top of the list. Accordingly, this paper introduces a methodology to connect both systems, ROS and Unity, thus linking the performance in simulations and real-life for better analogy. Additionally, a comparative study between GAZEBO simulator and Unity simulator, in terms of functionalities and capabilities is shown. Last but not least, two use cases are presented for validation of the proposed methodology. Therefore, the main contribution of this paper is to introduce a methodology to connect both systems, ROS and Unity, to achieve the best possible approximation to vehicle behavior in the real world.**

I. INTRODUCTION

There are various tools to study the effect of in-vehicle systems or new vehicular technologies on road users behavior. Simulation technologies are used to validate algorithms and identify potential problems before real-life experiments, thus avoiding issues due to software aspects in the implementation. Driving simulators represent a model of a real environment and are ideal platforms to evaluate upcoming technologies without jeopardizing road safety. Furthermore, they make it possible to develop controlled experiments that provide insights into what can be improved and adjusted.

Autonomous vehicles, as machines being capable of accomplishing actions automatically without human intervention, can be considered as robots and as such, there are several frameworks that can be used to simulate their automated features. An example of them is the open source based Robot Operating System (ROS), which provides libraries and tools to support the creation of robot applications. On the other hand, the Unity 3D game development platform provides an open source environment, which makes it possible to develop and easily modify graphical environments.

There are several attempts towards the link between ROS and Unity, for different and specific purposes. In 2013, authors in [1] introduced Unity-Link protocol, which is a stream-oriented, layered and componentized design. The link was through Python scripts to send and receive data over TCP/IP sockets, however a detailed analysis of the performance was not available, since it was out of their scope. Later in 2014, authors in [2] proposed a methodology to connect from ROS to Unity, to send data over TCP/IP sockets. This was the first attempt to utilize $rosbridge$ library [3], which was responsible for taking JSON [4] strings and converting them to ROS messages, and vice versa. They verified their approach through leveraging a mobile robot via virtual environment display and interaction in Unity.

On the other hand, in 2015, authors in [5] presented a 3D simulation system for miniature unmanned aerial vehicles. The exchange of data packets between the Unity server and the ROS client was achieved by TCP/IP protocol. On the client side, a ROS node was developed $unitysocket$, which communicates to the preset IP address of the Unity server. Authors validated their approach through various experiments and proved the ability to glue ROS and Unity together for near real-time Unmanned Aerial Vehicles (UAV) simulations [6]. In 2017, authors in [7] presented several scripts which establish the connection and allow for invoking ROS services. The objective was to monitor and control industrial robotic arms and the approach was validated with satisfactory results.

Also in 2017, authors in [8] proposed a connection mechanism to bind ROS and Unity for virtual reality applications. The link was again through $rosbridge$ library, utilizing both JSON and BSON strings and converting them to ROS messages. The results from this approach were validated using human-robot interaction in a virtual reality simulator. The idea of $rosbridge$ library to connect ROS and Unity obtained the best results, therefore Michael Jenkins created $unityros$ library [9]. This Unity library was implemented to send and receive data from and to TurtleBot [10]. Then the library was enhanced in [11] to include more messages and move one step closer to the generalization of the usage independent from the platform[12].

We introduce in this paper a generic methodology to connect both systems, ROS and Unity. The approach was tested using intelligent vehicles, to achieve the best possible approximation to vehicle behavior in the real world. Additionally, a comparative study between GAZEBO simulator and Unity simulator, in terms of functionalities and capabilities is described.

The remainder of this paper is organized as follow; Section II introduces briefly ROS framework, and Section III introduces briefly Unity Game Engine. The methodology to use Unity as a simulator for ROS-based applications is presented in Section IV. Afterward, to validate the proposed work, two uses cases were examined and discussed in Section V.

[1]Intelligent Systems Lab (LSI) Research Group, Universidad Carlos III de Madrid (UC3M), Leganés, Madrid, Spain {ahussein, fegarcia}@ing.uc3m.es

[2]University of Applied Sciences Technikum Wien, Information Engineering and Security Department, Vienna, Austria olaverri@technikum-wien.at

 DOI: 10.1109/ICVES.2018.8519522

Finally, Section VI concludes the paper and presents possible future work.

II. ROS Framework

ROS stands for robot operating system, which is an open source programming framework for robotics, whose development began in the late 2000s at Stanford University and Willow Garage [13]. To better understand its benefits, let's consider how robotics projects were developed before ROS existed. The kinds of commercially available systems have historically involved software control systems that were proprietary and highly specialized for their intended tasks. Each system was different, therefore lack standardization, and as a result suffered from long development times.

ROS changed this by a framework for developing a software that facilitates and even encourages the sharing and reuse of good ideas. The goal was not to have one general purpose software programming framework that will solve all the problems with robotics development. However ROS has maximized the utility of having open source programming framework, by addressing many of the common problems robotic developers face.

ROS framework can be classified into several modules, as depicted in Figure 1. It enables modular software development by providing a library of reusable code packages that are free and available. ROS also provides a run-time environment that supports near real-time communication between system elements and data sharing. Moreover, ROS provides programming standards, which are useful for creating and using ROS supported codes in a repeatable and reliable way. There is also a suite of ROS development tools that are helpful in monitoring, troubleshooting and visualizing the system. Furthermore, ROS has a vibrant community of tens of thousands of users and contributors all over the world.

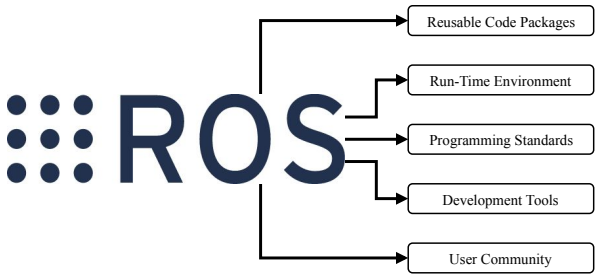


Fig. 1. ROS framework modules

III. Unity Game Engine

Unity Technologies released the very first version of Unity in mid 2005 targeting only OS X development [14], since then, Unity has developed and upgraded the support to more than 25 platforms including virtual and augmented reality. Unity is a platform independent game engine, where games are created by manipulating 2D or 3D objects and attaching several components to them. It has a powerful PhysX physics engine, render engine, collision detection engine among others. Furthermore, it supports high-fidelity 3D environments and allows sensors modeling [15]. Unity engine utilizes a standard mesh for the game object, which includes the location of all vertices of the object shape. This allows the physics engine and collision detection engine to enhance the authenticity and realism for the behavior of the dynamic game objects in the simulator. Moreover, Unity render engine provides several shaders and graphics effects to enhance the authenticity and realism for the 3D virtual environment [16].

Furthermore, Unity has a wide range of online tutorials, which facilitates getting started with the development from day one. Unity has a well-documented API, along with an engaged vibrant community. Unity provides the opportunity to modify the virtual environment using an editor, or by manipulating it directly in the game window. It also provides a scripting language for the developer to define behaviors and controllers.

Based on the comparison of different game engine [17], Unity is a feature-rich and highly flexible editor. As shown in Figure 2, it is characterized by all-in-one editor, where it is possible to implement the project on any operating system, additionally it supports 2D and 3D development with features and functionality needs across genres. Moreover, it has an advanced AI path-finding tools, which includes a navigation system that allows to create Non-Player Characters (NPCs), which can intelligently move around the virtual environment. Another feature is the rapid iteration, where play mode is used for rapid iterative editing and the alterations in the scripts are viewed instantly. Finally, Unity has a vibrant developer community, with thousands of threads for developers to share their ideas, suggestions, information and queries.

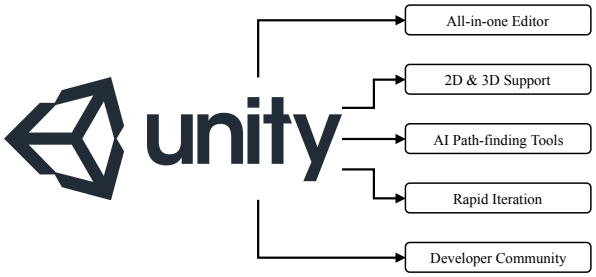


Fig. 2. Unity game engine features

IV. ROS and Unity Link

In this section, the proposed ROS bridge libraries, for both ROS and Unity ends, are defined; followed by an explanation on the system integration through a simple example.

A. ROS Bridge Library

Libraries are required for linking the two architectures, ROS and Unity, which must be implemented in both ends. Figure 3 depicts the simplified general overview of the link, where the ROS framework is on the left side, with all nodes that one might have in the system, in addition to a $unity\_node$, which acts as the link to subscribe and publish messages through the $rosbrdige$ node from and to

Unity. Whereas Unity game engine is on the right side, with all scripts one might have in the system, in addition to ros_script, which acts as the link to subscribe and publish messages through the $rosbridgelib$ library from and to ROS. All transmitted messages between the two architectures are formatted as JSON. The detailed methodology for each architecture is described as follows.

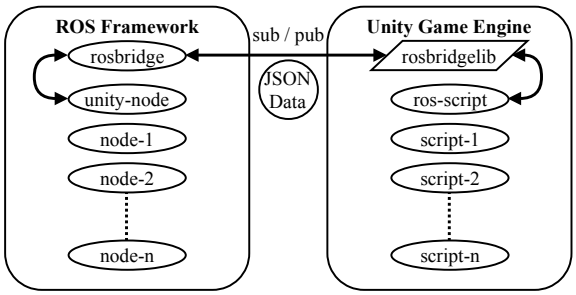


Fig. 3. ROS and Unity link general overview

1) For ROS Framework: the $rosbridge$ library [3] is utilized. $rosbridge$ library provides a JSON API to ROS functionality for non-ROS programs. It is a Python library responsible for taking JSON strings and converting them to ROS messages, and vice versa. In this approach, WebSocket server was selected as the front end, where $rosbridge$ server accepts WebSockets connections and implements the $rosbridge$ protocol. Therefore, after the installation of the library, launch the $rosbridge\_websocket$ node from the $rosbridge\_server$ package, after adjusting the WebSocket IP and port number. Afterward, in the $unity$-$node$, subscribe to the messages from Unity and publish messages to Unity based on the application needs. Since the library is integrated as part of the ROS framework, it accepts any kind of standard or custom messages defined in the package dependencies.

2) For Unity Engine: The $rosbridgelib$ library [12] is utilized. This library is to be included in the Unity project assets folder, where it must be imported in the $ros - script$ file. This is the main file, which setups WebSockets connection to connect to the same IP and port number, which are adjusted by the $rosbridge\_websocket$ at the ROS end. Once the connection is established, it defines the Unity publishers and subscribers. Furthermore, the $Update()$ function provides a mechanism for the callbacks on the rendering thread and deserializes JSON data into appropriate instances of packets and messages. The library does not support every ROS message, the currently available ones are:

- std_msgs
- $geometry\_msgs$
- $sensor\_msgs$
- nav_msgs
- $geographic\_msgs$
- auv_msgs

However, in the case of the necessity of new messages or custom messages, to create the script files for these messages, and include them in the library folder is straightforward.

B. System Integration Example

For the system integration, a full example in both ends, ROS and Unity is developed in [18]. The repository holds a ROS package with the $unity-node$, in addition to the Unity project with the $ros - script$. The Unity project is based on the introduction tutorial example "Roll-a-ball", where the objective is to control a ball on a board with the keyboard arrows to collect moving cubes, as shown in Figure 4.

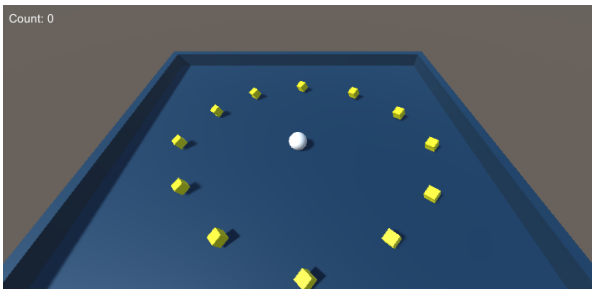


Fig. 4. Roll-a-ball Unity Game

Figure 5 describes the architecture of both ends. On the one hand, in the ROS end, the $unity - node$ publishes the desired speed of the ball and the desired pose of the ball to be. Additionally, $unity - node$ subscribes to the current pose of the ball. All the topics are subscribed and published by the $rosbridge$ node and convert from and to JSON data to be sent to Unity. On the other hand, in the Unity end, the $rosbridgelib$ convert the incoming and outgoing JSON data for the link with ROS, where the $ROSController$ script is the main script for the initialization of the connection and handling the subscribers to the ball desired pose and desired speed, along with the publisher for the ball current pose. Therefore, this ensures the ability to control the ball motion in Unity through ROS node.

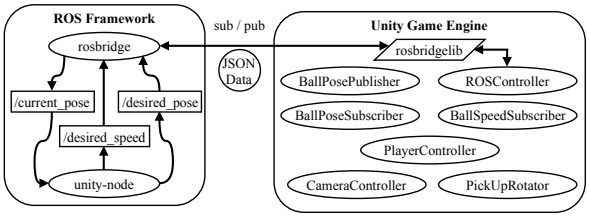


Fig. 5. Roll-a-ball architecture for both ROS and Unity

This simple example is to show how the architecture works, however, for validation of the proposed methodology, two uses cases were selected and presented in the next section.

In order to integrate the package to the system, the procedure is as follows, validated by the example repository available in [18].

C. In ROS

First, install the $rosbridge-suite$ [3] to the system. Then, create a new node responsible for the communication with

Unity. In the main loop of the node, define the necessary subscribers for the messages published by Unity, and the necessary publishers for the messages published to Unity. In the callback of the subscribers, execute the necessary functions to use the data from Unity accordingly. To run the package, create a launch file, which includes the $rosbridge\_server$ node and the implemented communication node. A configuration file is optional, to choose the communication port and other parameters.

D. In Unity

First, clone the $rosbridgelib$ [12] as sub-module to the project assets. Then, create a new script responsible for the communication with ROS. The script class must inherit the $MonoBehaviour$, and in the Unity $Start()$ function initialize the $ROSBridgeWebSocketConnection$ parameter with the $rosbridge\_server$ IP address and port. This ensures the link through the WebSockets, either using the same machine, e.g. localhost, or using different machines in the same network. Afterward, add the necessary subscribers for the messages published by ROS and the necessary publishers for the messages published to ROS. Each subscriber or publisher requires a separate script, which is discussed below. Last step is to execute the $Connect()$ function to initiate the subscribers and publishers. It is worth-noting, to include the $Disconnect()$ function in the Unity $OnApplicationQuit()$ function, in order to discontinue the flow of messages.

In the Unity $Update()$ function, first the $Render()$ function is executed, in order to refresh the incoming and outgoing messages. Followed by the code portions to update the parameters for publishing to ROS. Later on, for each implemented subscriber, the class must inherit $ROSBridgeSubscriber$, where the message topic is defined, the message type, the parsing function using $JSONNode$, and finally the callback function. On the other hand, for each implemented publisher, the class must inherit $ROSBridgePublisher$, where the message topic is defined, the message type, the parsing function using $JSONNode$, and finally the parsing function using the message type.

V. Validation Use Cases

Two use cases were selected for validation of the proposed methodology to generically link ROS and Unity. The first use case is using the mobile robot TurtleBot simulator, and the second one is using 3DCoAutoSim driving simulator.

A. TurtleBot

TurtleBot is a series of inexpensive robots developed specifically for ROS. The first TurtleBot prototype was created by Willow Garage in 2010. Willow Garage is a group that develops hardware and open source software for robotic applications. TurtleBot name comes from the ROS tutorial simulator, which was named after the cursor name of the Logo software. TurtleBot2 was released in 2012 and TurtleBot3 in 2017 [10]. TurtleBot3 designs are clearly smaller than the previous TurtleBots. TurtleBots included an embedded computer, which was replaced by TurtleBot3 single PCB computer, Raspberry Pi 3 [19]. The robot has also been made very modular, as the whole frame has built-in screws for easier mounting [20]. TurtleBot3 is available as two models; "Burger" is smaller, and "Waffle" is larger. The models chassis consists of identical mounting plates, but have different components. The most technical difference between the models is the 3D camera included in "Waffle" TurtleBot3 [21], both platforms are shown in Figure 7.

Figure 6 shows the designed environment of dimensions of 6×9 meters. The red-filled circles simulate the possible target locations to reach by each of the platforms. The environment on the left was created in GAZEBO [22], while the one on the right was created in Unity. The connection of the platform functionalities in GAZEBO are straight forward, since the platform is supported by the simulator. On the other hand, for the Unity simulator, the proposed libraries were utilized to link the platform ROS-based environment with the designed simulator, to publish the lidar and position data from Unity, at the same time to subscribe to the planning and control commands from ROS.

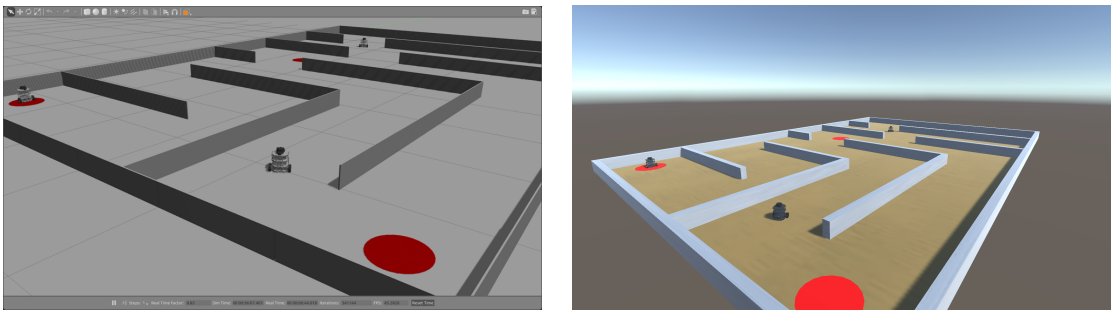
The platforms performed SLAM to generate the static map of the environment using the on-board lidar scan. The five tasks and three platforms represent a small-scaled problem, after executing the solving algorithm presented in [23], the obtained results are reported in Table I.

TABLE I
TurtleBot3 allocation results

Best Allocation Cost [m]	Average Comp. Time [s]	Deviation Error [%]
12.76	1.04	3%

B. 3DCoAutoSim

3DCoAutoSim is an abbreviation for "3D Simulator for Cooperative Advanced Driving Assistance Systems (ADAS) and Automated Vehicles Simulator". It is a vehicle driving simulator with high-quality 3D visualization based on Unity, which makes it possible to emulate a variety of environments. The simulator is an extension of the capabilities presented in [24], where a driver-centric driving platform visualized the mobility behavior of other vehicles based on traffic models and a TraCI protocol allowed communication between Unity and the microscopic traffic simulator Simulation of Urban MObility (SUMO). And the work in [25], which calculated the optimal speed while approaching an intersection by retrieving the traffic light timing program from the road infrastructure, namely Traffic Light Assistance System. Finally, the work in [26], in which Vehicle Ad-hoc Networks (VANET) communication capabilities were used to assess different information paradigms. The simulator is implemented using Unity since it is a powerful 3D visualization tool, which is platform independent and has a strong physics engine. The simulator architecture hierarchy is divided into several categories; environments, scenarios, features, outputs, devices, and mode.



(a) GAZEBO based simulation (b) Unity based simulation

Fig. 6. TurtleBot3 platforms simulated environments

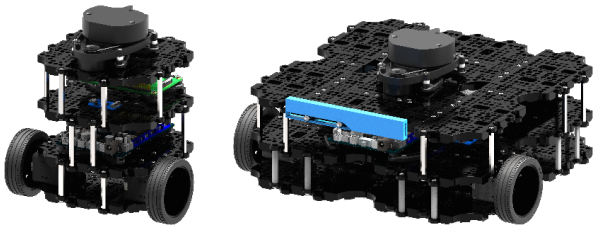


Fig. 7. TurtleBot3 platforms Burger (left) and Waffle (right) [20]

Connecting the simulator with the ROS framework is an advantage, in order to link the the emulated world with the real world vehicles controllers. The connection integration is based on the proposed library to publish and subscribe to as many topics as required for the desired objective, for instance automated driving. Therefore, a comparison was carried out to validate the connection by controlling the vehicle using ROS packages.

In order to manually control the vehicle in the simulator, a Thrustmaster T500 RS controller is used as the steering wheel, in addition to its throttle, brake and clutch pedals, and the TH8 RS gear shifter, as shown in Figure 8. The controller is connected to a car play seat, to simulate a real vehicle and the simulator visuals are displayed using overhead HD beamer with resolution of $1400x1050$, in addition to a five point one surround sound system.

Based on the work presented in [27], the selected scenario was the surroundings of the University of Applied Sciences Technikum Wien, in the city of Vienna. The trajectory was a total distance of 2.6 km, which included intersection, traffic lights, a roundabout and pedestrians crossing. In order to evaluate the functionality and efficiency of the ROS controller, two evaluation metrics are calculated for the vehicle position to self-drive the desired path and compare it to the theoretical path obtained from the path planning algorithm presented in [28].

Figure 9 depicts the trajectory by the vehicle for one of the carried-out experiments, where the green point represents the starting position, and the red point represents the final position. The ROS controller path tracking delivered the best results in terms of deviation from the lane center. This was due to the fact that it relied on the Ackermann modeling for the simulation of kinematic and dynamic parameters [28]. These experiments validated the functionality and efficiency of the proposed approach to link Unity and ROS together for the best outcome.

VI. Conclusion

The technological advances in the Intelligent Transportation Systems (ITS) are exponentially improving over the last century. The objective is to provide intelligent services for the different modes of transportation, towards a better, safer, coordinated and smarter transport networks. However, to test all possible outcomes in the development of automated vehicles, a simulator is required that emulates vehicles and linked to the controllers and provides a linking device, system, framework. Accordingly, this paper introduces a method to connect a powerful simulation tool, Unity game engine, to the Robot Operating System (ROS) framework.

The proposed approach utilizes libraries in both sides. Each library is based on JSON API to encode and decode messages across the different software using WebSockets. Several experiments were carried out for validation of the proposed work on different platforms in the context of ITS. The results show the viability of the proposed approach to emulate ideal conditions. While the proposed approach obtained successful results, there were some aspects of the problem that were assumed for simplification. These assumptions could be addressed in the future work, to increase the reliability of the approach.

ACKNOWLEDGMENT

This research work was partially performed during a research stay with Dr. Olaverri-Monreal, which enabled the academic agreement to strengthen scientific and educational cooperation between the UAS Technikum Wien, Austria and the University Carlos III de Madrid, Spain. It is also supported by Madrid Community project SEGVAUTO-TRIES (S2013-MIT-2713) and by the Spanish Government CICYT projects (TRA2015-63708-R and TRA2016-78886-C3-1-R).

References

[1] A. B. Lange, U. P. Schultz, and A. S. Soerensen, "Unity-link: A software-gateware interface for rapid prototyping of experimental robot controllers on fpgas," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2013, pp. 3899–3906.

(a) Pedals (b) Gears shifter (c) Steering wheel

Fig. 8. Simulator controllers

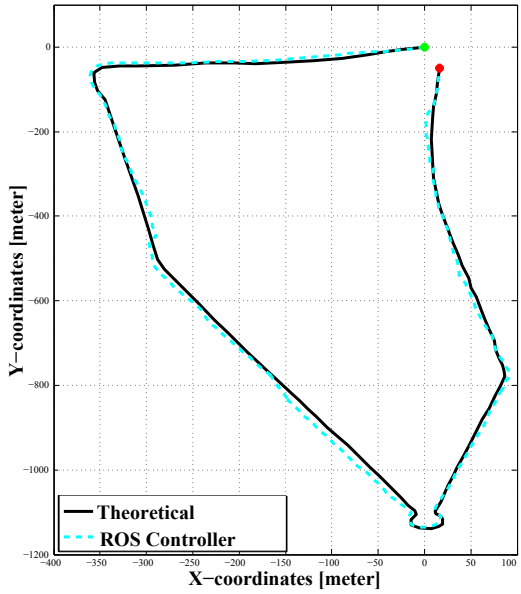


Fig. 9. ROS controlled path against theoretical one

[2] R. Codd-Downey, P. M. Forooshani, A. Speers, H. Wang, and M. Jenkin, "From ros to unity: Leveraging robot and virtual environment middleware for immersive teleoperation," in *IEEE International Conference on Information and Automation (ICIA)*. IEEE, 2014, pp. 932–936.

[3] R. T. Jonathan Mace and J. Lee, "Ros bridge suite," *GitHub Repository*, pp. [Last Accessed: 2018–05–30], 2018. [Online]. Available: http://wiki.ros.org/rosbridge_suite

[4] D. Crockford, "The application/json media type for javascript object notation (json)," *IETF*, pp. 1–10, 2006.

[5] W. Meng, Y. Hu, J. Lin, F. Lin, and R. Teo, "Ros + unity: An efficient high-fidelity 3d multi-uav navigation and control simulator in gps-denied environments," in *41st Annual Conference of the IEEE Industrial Electronics Society (IECON)*. IEEE, 2015, pp. 2562–2567.

[6] Y. Hu and W. Meng, "Rosunitysim: Development and experimentation of a real-time simulator for multi-unmanned aerial vehicle local planning," *Simulation*, vol. 92, no. 10, pp. 931–944, 2016.

[7] E. Sita, C. M. Horvath, T. Thomessen, P. Korondi, and A. G. Pipe, "Ros-unity3d based system for monitoring of an industrial robotic process," in *IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 2017, pp. 1047–1052.

[8] Y. Mizuchi and T. Inamura, "Cloud-based multimodal human-robot interaction simulator utilizing ros and unity frameworks," in *IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 2017, pp. 948–955.

[9] M. Jenkin, "Unity ros," *GitHub Repository*, [Last Accessed: 2018-05-30]. [Online]. Available: https://github.com/michaeljenkin/unityros

[10] E. Ackerman, "Turtlebot inventors tell us everything about the robot," *IEEE Spectrum*, pp. 1–10, 2013.

[11] M. C. Thorstensen, "Visualization of robotic sensor data with augmented reality," Master's thesis, Universitetet i Oslo, 2017.

[12] M. C. Thorstensen, "Ros bridge lib," *GitHub Repository*, [Last Accessed: 2018-05-30]. [Online]. Available: https://github.com/MathiasCiarlo/ROSBridgeLib

[13] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," *ICRA workshop on open source software*, pp. 1–5, 2009.

[14] D. Helgason, "Unity 1.0 is shipping," *Unity Technologies*, [Last Accessed: 2018-05-30]. [Online]. Available: https://forum.unity.com/threads/unity-1-0-is-shipping.56/

[15] A. S. Kyaw, *Unity 4.x Game AI Programming*. Packt Publishing Ltd, 2013.

[16] S. Christodoulou, D. Michael, A. Gregoriades, and M. Pampaka, "Design of a 3d interactive simulator for driver behavior analysis," in *Proceedings of the 2013 Summer Computer Simulation Conference*. Society for Modeling & Simulation International, 2013, p. 17.

[17] A. Smid, "Comparison of unity and unreal engine," Ph.D. dissertation, Czech Technical University in Prague, 2017.

[18] A. Hussein, "Unity ros roll a ball," *GitHub Repository*, pp. [Last Accessed: 2018–05–30], 2018. [Online]. Available: https://github.com/slmat27/unity-ros-roll-a-ball

[19] E. Upton, "Raspberry pi 3," [Last Accessed: 2018-05-30]. [Online]. Available: https://www.raspberrypi.org/blog/raspberry-pi-3-on-sale

[20] S. Korkalainen, "Turtlebot3-robotit," *Metropolia Ammattikorkeakoulu*, pp. 1–40, 2017.

[21] E. Guizzo and E. Ackerman, "The turtlebot3 teacher [resources hands on]," *IEEE Spectrum*, vol. 54, no. 8, pp. 19–20, 2017.

[22] O. S. R. Foundation, "Gazebo," [Last Accessed: 2018-05-30]. [Online]. Available: https://bitbucket.org/osrf/gazebo

[23] A. Hussein, P. Marin-Plaza, F. Garcia, and J. M. Armingol, "Hybrid optimization-based approach for multiple intelligent vehicles requests allocation," *Journal of Advanced Transportation*, vol. 2018, pp. 1–12, 2018.

[24] C. Biurrun, L. Serrano-Arriezu, and C. Olaverri-Monreal, "Microscopic driver-centric simulator: Linking unity3d and sumo," in *World Conference on Information Systems and Technologies*. Springer, 2017, pp. 851–860.

[25] C. Olaverri-Monreal, J. Errea-Moreno, and A. Diaz-Alvarez, "Implementation and evaluation of a traffic light assistance system in a simulation framework based on v2i communication," *Journal of Advanced Transportation*, 2018.

[26] F. Michaeler and C. Olaverri-Monreal, "3d driving simulator with vanet capabilities to assess cooperative systems: 3dsimvanet," in *Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 999–1004.

[27] A. Hussein, A. Diaz-Alvarez, J. M. Armingol, and C. Olaverri-Monreal, "3dcoautosim: Simulator for cooperative adas and automated vehicles," in *IEEE International Conference on Intelligent Transportation Systems (ITSC2018)*. IEEE, 2018, pp. 1–6, under-review.

[28] P. Marin-Plaza, A. Hussein, D. Martin, and A. d. l. Escalera, "Global and local path planning study in a ros-based research platform for autonomous vehicles," *Journal of Advanced Transportation*, vol. 2018, pp. 1–11, 2018.