Hardware-in-the-loop and Autonomous Driving Simulation with TurtleBot3

Catarina Ramos, Pedro Dias and William Fukunaga
Faculty of Engineering of the University of Porto,
up201406219@fe.up.pt, up201404178@fe.up.pt, up201405119@fe.up.pt.

Abstract - Autonomous vehicles have become a reality in recent years, but as the interest increases and the investigation in the field expands, the need of having a safer, cheaper, time controlled and appropriate testing environment is also in demand to give the researchers the support they need. To ensure the maximum conditions for the success of the investigations in these domains, this paper presents the development of an integrated platform that aims to close the gap between hardware in the loop and autonomous driving simulation. The platform consists of the integration of SUMO, an urban mobility simulator, and ROS, a robot operating system, where a *turtlebot3* can be commanded through ROS and be submerged in the virtual environment where it has to follow the rules and protocols that were established. This allows engineers to concentrate on the software and hardware devices of primary interest while keeping the convenience of the simulation and insight into vehicle dynamics. By using a single physical device integrated with a virtual simulation, the costs of testing and research time are reduced.

Index Terms - Integration, Simulation, SUMO, ROS.

1. Introduction

This study enters the domain of Hardware in the Loop Simulation (HLS) and Autonomous Driving Simulation (ADS).

HIL is a type of real time simulation used to test a controller design and check if a physical model is valid. For this purpose, a virtual and a real representation of the controller that its intended to be tested is used. HIL allows to control and create complex situations in the virtual environment, including the virtual representation, to test the physical model in real-life situations that are hard to meet/not practical or expensive to be tested in real environments [8].

ADS is a tool whose purpose is to test the several intricacies of the different modules that make an autonomous driving system. It also allows the analysis of autonomous driving behaviour in a network of urban roads with vehicular traffic in a safe and controlled environment [9].

Autonomous vehicles have become a reality in recent years and with that, the necessity to integrate hardware with a virtual environment to test, develop and to increase their autonomy. This relation allows engineers to concentrate on the software and/or hardware devices of primary interest while keeping the convenience of the simulation and insight into vehicle dynamics.

The goal of this study is to create a platform that integrates SUMO with ROS and test it by inserting a real robot (turtlebot) in the virtual environment. These tests verify the reaction and hardware of the robot when data inputs in the sensors are introduced via the virtual world.

The basis of this simulation study is to answer the question: Can a virtual agent inside a SUMO simulation be controlled through the movements of a (real) turtlebot using ROS? With this hypothesis, issues arise: How to calibrate the time and movement difference between the turtlebot and SUMO? How to deny the turtlebot movement when its agent representation in the simulation has its path blocked? How does a turtlebot respond to elements inside the simulation?

At the end of this platform development, it is expected that further experiences can occur between these environments and in these contexts without having to lose time, money or imposing risks.

This article contains a literature review section in which a few other articles and projects are mentioned about their relation with this article and their importance to the area of study. Also, a methodological approach where a description of the project with its environments and rules. After, the discussion and analyses of the results obtained from the study and some conclusions with notes for future works and applications.

2. Literature Review

Gazebo is an open-source 3D robotic simulator that was integrated with ROS that allows simultaneous development of the robot and their sensors with robot control algorithms allowing us to have a good example of the integration design and process.

Analysing the integration of Gazebo [4] and the proposed with Sumo, Gazebo has a more generic scope allowing it to be used in a more varied number of cases but comes at a cost, it does not have such refined implementation of traffic systems compared to SUMO.

At this time, there is no open-source tool that allows the development of automotive robots that integrates traffic simulation with virtual and real robots.

3. Methodological Approach

3.1. Problem Formalization

The goal is to integrate the ROS robot in a virtual environment such as SUMO. For this purpose, each integrated robot will have a main controller in the ROS and SUMO side to ease the communication between these two.

On the SUMO side, TraCi is used to retrieve information about the simulation and communicate it to ROS.

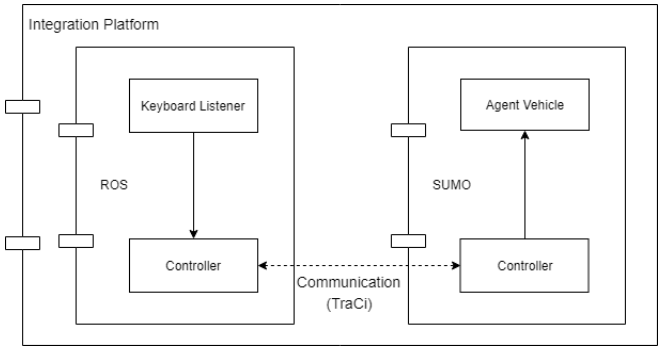

Image 1 - Integration of SUMO with ROS

To be integrated, a ROS robot must send an integration_request to SUMO. Then, it replies with the information about the virtual representation, such as the vehicle id. After the integration is made, the system is apt to receive direction instructions.

Each turtlebot connected to ROS is controlled by the keyboard's arrow keys. The correspondent keyboard listener in ROS, for that specific turtlebot, has to communicate these instructions to SUMO so that the virtual representation acts as the real one.

This sequence can be viewed in the following *Image 2.*

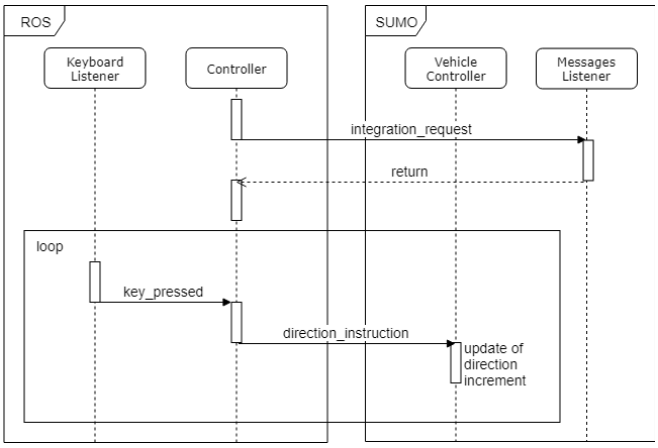

Image 2 - Sequence Diagram for the integration and the communication of a possible direction instruction.

When a simulation step is given, SUMO will check the viability of the current direction increment regarding the current position of the virtual vehicle and, if it doesn't cause any collisions or impossible movements, it will send to ROS, the current direction instruction increment for that step. Then, it will delay the update of its position so that the two environments are synchronized. After this delay, the two environments update their positions with the same direction instruction.

This sequence can be seen in the *Image 3.*

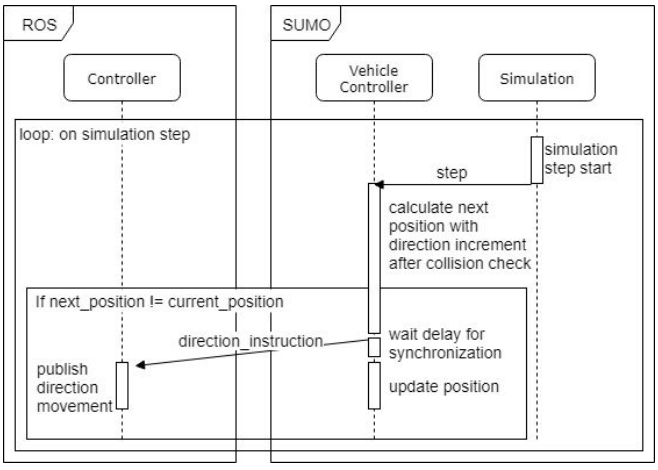

Image 3 - Sequence Diagram for the update of the current position when a simulation step is given.

For this purpose, there are some issues that need to be analysed:

- Delay in message transition: During the simulation, there may be some delay period that needs to be taken into account. This delay can be critical to the simulation since messages received after a long time can create an irreversible action that ruins the simulation. For example, the execution delay (see image above) between the execution of an instruction in the SUMO and the execution in the real turtlebot. To ease this issue, see section 3.3.4.

- Timeline synchronization: During the simulation, the time elapsed in real world can be different from the time elapsed in the simulation, so there is a need to sync the position, velocity and other parameters. To ease this issue, see section 3.3.4.

- Problem with Messages: Loss, duplicates and incomplete/wrong information. To solve this problem, the communication will be made through TCP.

- A published direction instruction on ROS has the duration of one second. This means that during one second, the robot will perform the movement related to the direction instruction taking into account the velocity. However, the duration must be regarding the time of a simulation step. To ease this issue, SUMO will be giving direction instructions to ROS, based on the viability of the last direction instruction received by ROS, on each step. When ROS receives a direction instruction, it will update the 'last instruction time' that was received and publish the instruction. A thread in a infinite loop is responsible to check if the timestamp between the 'last instruction time' and the current time has surpass the time of a simulation step. If it is the case, it will publish an instruction to stop the movement of the robot. This allows the movement to have the maximum duration of a simulation step or until a new instruction is received.

- A ROS vehicle receives direction instructions, such has 'go forward', it is independent of the current angle, because ROS isn't aware of the vehicle position or angle. However, SUMO is dependent of these factors to calculate the next position. To move, it is necessary to know the x, y and angle position after the movement. Thus, correct calculations must be made to calculate the next position when receiving instructions such as 'go forward'.

- A ROS vehicle rotates over its center and SUMO over the center point of the front of the vehicle. To have the same visual rotation and when updating its position, the length of the virtual representation is calculated and the followed movement instructions are made by the virtual vehicle (only):
  - It retreats half its length;
  - Performs the rotation;
  - It advances half its length.

3.2. System Model

The virtual environment is going to be composed with a set of roads connected with each other through an intersection just as *Image 4*. The intersection will have a central semaphore for traffic and passage control. Since all the road entries and exits starts and ends in the intersection, it's not needed to add vehicles in the environment after the simulation has started.

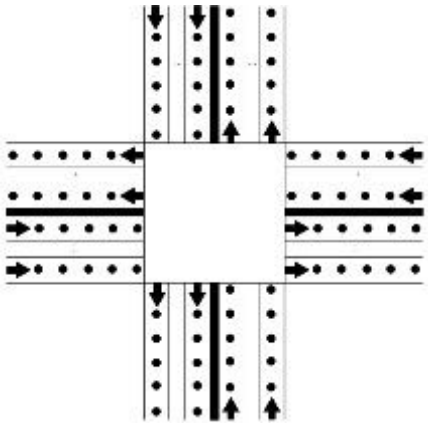

Image 4 - Intersection Model.

Modelling metaphor:

The modelling metaphor used for this model is the Agent-Based Model and Simulation since agents are the best way to represent entities with behaviour.

Input variables:

The controllable variables are: the number of integrated robots within the same virtual environment, the time of the step in the virtual world, the number of independent vehicles and the position and velocity of the integrated robots.

The uncontrollable variables are: number of entries and exits of the intersection, the layout of the roadways and the position of the virtual vehicles.

Output variables:

The output variables are related to the messages received and send by each platform and the delays between them.

Performance Metrics:

To measure the performance, all tests described on section 3.3.2 must be exceeded. The delay between information request messages and replies and the direction instructions' execution in the real and the virtual environment are also going to be taken into consideration.

Decision variables:

The surrounding cars can't allow direction instructions that provokes collisions.

When the semaphore light is red, no vehicle is allowed to pass through the intersection, green otherwises.

In case of collision, a controllable vehicle can change its position if the area of collision decreases. The area of a vehicle is considered to be the surrounding box of the vehicle with a security distance. The area of collision is the area that results in the intersection of the areas of two or more vehicles.

Operating policies:

Since the model is based on an intersection, the model can be considered to have a multi queue, FIFO with one server policy.

States of the system:

For the possible scenarios, see section 3.6.1, the system contains information such as the ones marked in *Table 1*. Since there can be more than one robot, entry and negotiation, depending on the scenery, the system will contain the targeted information related to each one.

Scenery	A	B
Vehicle Position	x	x
Vehicle Velocity	x	x
Semaphore Light	x	x
Collisions		x

Table 1 - States of the system for each possible scenery.

Entities of the system:

The entities present on the system are: the virtual and real robots and the semaphore.

Reference models:

There is an integration platform with ROS and GAZEBO (See [4] in the References section). This integration platform isn't applied to the context of autonomous driving simulation, but it simulates virtual environments and the insertion and testing of real components such as vehicle like robots (HIL).

3.3. Conceptual & Logical models

3.3.1. Verification

To further verify the integration platform, the tests below must be performed and passed. The real bot representation agent will be abbreviated to "*V_TB*" and the real turtlebot as "*R_TB*".

3.3.2. Tests

Test — Each *R_TB* must be connected to a *V_TB*.

Steps — 1.For each *R_TB* connected to ROS, there is the creation of a *V_TB* in SUMO.

Test — All virtual agents (including *V_TB*) must stop when the semaphore light is red (independently of the direction instructions). If the light is green, the passage through the semaphore is now allowed.

Steps —
1.Go near a semaphore (first position);
2.Wait for it to be with a red light;
3.Give instructions to ROS to move forward;
4.Validate that the direction instructions don't have any effect;
5.Wait for the light to turn green;
6.Give instruction to ROS to move forward;
7.Validate that the direction instruction have effect;

Test — All virtual agents (including *V_TB*) must stop to avoid collisions with other vehicles.

Steps — 1.Give instructions to ROS to move towards a

vehicle to cause a collision;
2.Validate that the ROS instructions to move don't take any effect, so as to avoid the collision

3.3.3. VALIDATION

For this project, there will be no validation performed on the model made. To clarify, validation of the model would require the turtlebot to behave as predicted inside the sumo simulation. However, since the goal of this project is a proof of concept, the validation is not mandatory and can be postponed for future projects.

3.3.4. CALIBRATION

For this project, there are two kinds of calibrations that can be made: calibration of the two environment scales and calibration to prevent big execution delays.

To calibrate the system, it's necessary to eliminate the difference that exists between the real and the virtual environment. For this, it is necessary to adjust the positions and associate scales to several parameters such as: speed (and distances), elapsed time and others. This can be achieved by making scaling variables accessible to the TurtleBot controller, so the controller can adjust the received information to movements, scaled. This information sharing can be performed and the beginning of a run.

The delay in the execution of a direction instruction between the real and the virtual vehicle needs to be compensated. For this reason, between x and x seconds, a message of type 'CALIBRATION' is sent to from the vehicle controller with the time of sent. When ROS receives this message, it changes the message and updates the time that it was received. When the message is received by the controller, the timestamp is calculated and the delay of the controller is update, as it can be seen in the *Image 5*.. This delay represents the delay that the vehicle has to wait after sending a direction instruction and before updating its position (see *Image 3*).

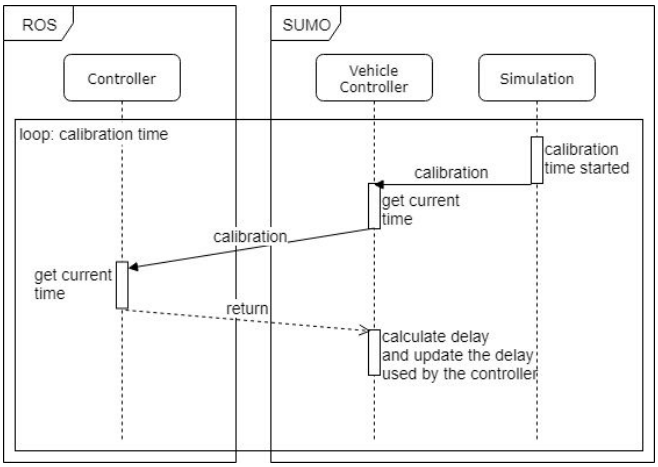

Image 5 - Sequence that represents the calibration between the two environments.

3.4. CODING

To handle the SUMO simulator, a python script will be used to act as a server which will receive all requests to make the necessary changes to the simulation. The ROS middleware will be handled in python to receive the information gathered from the turtleBot and send it to the SUMO server.

ROS is a free and open-source framework with several years of development focused on large-scale service robots like the Stair project at Stanford University. The philosophical goals of ROS can be summarized as: peer-to-peer topology, support for multiple programing languages, microkernel design with large set of tools to run various components.[2]

SUMO is an open-source traffic simulation package developed since 2001. Can be defined as a purely microscopic traffic simulation where each vehicle is given explicitly a set of informations. The most common applications is vehicular communication, route choice and dynamic navigation, traffic light algorithms, surveillance systems. In recent years, has been used in several projects like iTetris, CityMobil.[3]

TraCI is an open-source interface for coupling road traffic and network simulators, that allow control over vehicle mobility and get information such as pre-crash sensing, approaching emergency vehicle.[7]

3.5. Data Requirements

3.5.1. Input Data

3.5.1.1. Data Sources

ROS

Input data from ROS is required to represent the turtlebot inside the simulation, the running simulation needs to know:

- TurtleBot linear velocity;
- TurtleBot angular velocity.

SUMO

The SUMO simulation will receive a configuration file describing the layout of the roads and information about the number and types of agents. In this file it's possible to set car characteristics like size, representation inside the simulation, max acceleration, max speed. Also, it will receive some simulation settings for example start time, end time, ticket delay, viewport and the scale between the two environments (see section 3.2.3).

After the integration, input data from SUMO is required so that the communication between the two environments and the synchronization is eased. The input data provided are:

- The virtual representation id;
- The time of a simulation step.

3.5.2. Output Data

3.5.2.1. Data analytics

Some data from the simulation will be covered and analysed to produce conclusions. This data will include:

- the delay between the movement execution in both environments;
- the delay between sending and receiving a message;

There can also be some real-time information:

- Messages sent and received;
- Refused instructions;
- Current position of the virtual representation.

3.5.2.2. Data visualization techniques

To observe the simulation during a run, it is possible to use a GUI that describes the agents involved and the paths possible to traverse. It's also possible to see real-time information such as: messages received and their information (refused/accepted direction instruction and others) through the controller of the turtlebot in ROS.

3.6. Simulation Scenarios

3.6.1. Reference Scenarios

3.6.1.1. Scenario A

The environment is composed by an intersection with a semaphore at the center. The turtlebot (real vehicle) is going to be contained in this virtual environment and has to obey the semaphore rules.

3.6.1.2. Scenario B

Same base scenario as A but with the introduction of independent vehicles (pure sumo vehicles that are not controllable).

3.6.2. Simulation Plan

The number of simulation runs are as many as necessary to reach the project goals with success and with the minimum flaws possible.

After a run starts, it's not necessary to have a warm-up since the virtual vehicles will be placed randomly along the roadways. Otherwise, it would have been necessary to wait a x time so that the vehicles would disperse. However, to reduce the delay between the execution of a direction instruction in both environments, a previous run is needed to collect data, if it doesn't already exist, to measure the average delay between executions (view section 3.2.3 last paragraph).

The simulation run doesn't have to have a specific duration. However, it's advised to test all the implemented functionalities/features and if some of them have flaws, the simulation must stop with the intent to correct them and then repeat again the simulation.

The focus of this project will be the Scenario A, since it involves most of the rules needed to the correct integration of the turtlebot with the SUMO simulator.

4. Assessment/Results/Discussion

After the implementation, the previously described scenarios where executed multiple times and the results obtained are:

Scenario A Executed 10 times

Parameter	Average	Min	Max
Execution Time (s)	39	20	78
Message Delay (ms)	0.00967	0.00008	2.42161
Total messages	312	194	582

Scenario B Executed 20 times

Parameter	Average	Min	Max
Execution Time (s)	72	48	114
Message Delay (ms)	0.03620	0.00019	5.661
Total messages	481	250	604

In scenario A, compared to scenario B, the simulations ran in shorter times since the its goal was much simpler and faster to obtain, as seen by the execution times and the total messages sent.

The connection is fast even when the simulation contains several virtual agents as proven by the results in scenario B.

There were a few discrepancies in the response times, but even then, the delay is acceptable.

It is necessary to note, to conclude this analysis, that these scenarios were run with only one turtlebot. Therefore, there is no evidence that these results will stay similar when more than one turtlebot is used in the simulation.

5. Future Work

During the development of the platform we discovered that TraCi couldn't provide information about the SUMO-GUI delay. This led to the impossibility of synchronizing the two environments when there was a change in the simulation step duration. In the future, this is a issue that must be attend when TraCi or other mechanism allows the retrieval of this information. Currently the default simulation step duration is 100 ms.

6. Conclusion

While realizing this project, the group learned the complexity of a simulation system and the challenges to integrate a system into a simulation environment.

A physical turtlebot was not able to be immersed in the SUMO simulation due to complexity and time issues. However, a virtual turtlebot was used which has all the necessary features to emulate a physical turtlebot.

Apart from the issue mentioned above, the integration between a turtlebot and a SUMO simulation was completed in its entirety, with the agent respecting the simulation's traffic lights and avoiding collisions.

By recording the synchronization rate between the two sides, it is worth noting that the delay of the response from SUMO is very small.

To finish, the group believes this integration is available to be used as a starting point to more complex projects which use a physical device in a virtual simulation.

7. References

[1] Tatiana Pinho, António Paulo Moreira, José Boaventura-Cunha, "Framework Using ROS and SimTwo Simulator for Realistic Test of Mobile Robot Controllers"," CONTROLO'2014 – Proceedings of the 11th Portuguese Conference on Automatic Control", pp 751-759.

[2] Michael Behrisch, Laura Bieker, Jakob Erdmann, Daniel Krajzewicz , "SUMO – Simulation of Urban MObility An Overview", "SIMUL 2011 : The Third International Conference on Advances in System Simulation", Berlin, Germany.

[3] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust , Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, Andrew Ng, "ROS: an open-source Robot Operating System".

[4] Ilya Afanasyev, Artur Sagitov, Evgeni Magid, "ROS-based SLAM for a Gazebo-simulated mobile robot in image-based 3D model of indoor environment", Kazan, Russia.

[5] Guilherme Soares, Zafeiris Kokkinogenis, Jos´e L. Macedo, and Rosaldo J. F. Rossetti, "Agent-based Traffic Simulation using SUMO and JADE: an integrated platform for Artificial Transportation Systems", Simulation of Urban Mobility: First International Conference, SUMO 2013, Berlin, Germany, May 15-17, 2013. Revised Selected Papers (pp.44-61)

[6] FERRY, W. B., FRISE, P. R., ANDREWS, G. T. and MALIK, M. A. (2002), Combining virtual simulation and physical vehicle test data to optimize durability testing. Fatigue & Fracture of Engineering Materials & Structures, 25: 1127-1134. doi:10.1046/j.1460-2695.2002.00605.x

[7] Axel Wegener, Michał Piórkowski, Maxim Raya, Horst Hellbrück, Stefan Fischer, and Jean-Pierre Hubaux. 2008. TraCI: an interface for coupling road traffic and network simulators. In *Proceedings of the 11th communications and networking simulation symposium* (CNS '08). ACM, New York, NY, USA, 155-163. DOI: https://doi.org/10.1145/1400713.1400740

[8] W. Deng, Y. H. Lee and A. Zhao, "Hardware-in-the-loop simulation for autonomous driving," *2008 34th Annual Conference of IEEE Industrial Electronics*, Orlando, FL, 2008, pp. 1742-1747. doi: 10.1109/IECON.2008.4758217

[9] Ş. Y. Gelbal, S. Tamilarasan, M. R. Cantaş, L. Güvenç and B. Aksun-Güvenç, "A connected and autonomous vehicle hardware-in-the-loop simulator for developing automated driving algorithms," *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Banff, AB, 2017, pp. 3397-3402. doi: 10.1109/SMC.2017.8123155