Simulation Framework to Train Intelligent Agents towards an Assisted Driving Power Wheelchair for People with Disability

Giovanni Falzone, Gianluca Giuffrida[a], Silvia Panicacci[b], Massimiliano Donati[c] and Luca Fanucci[d]

Dept. of Information Engineering, University of Pisa, Via G. Caruso 16, 56122, Pisa, Italy

Keywords: Multi-Agent System, Simulation Framework, Reinforcement Learning, Assistive Technology, Power Wheelchair, Assisted Driving.

Abstract: Several million people with disabilities exploit power wheelchairs for outdoor mobility on both sidewalks and cycling paths. Especially those with upper limb motor impairments have difficulty reacting quickly to obstacles along the way, creating dangerous situations, such as wheelchair crash or rollover. A possible solution could be to equip the power wheelchair with a neural network-based assisted driving system, able to detect, avoid or warn the users of obstacles. Therefore, a virtual environment is required to simulate the system and then test different neural network architectures before mounting the best performing one directly on board. In this work, we present a simulation framework to train multiple artificial intelligent agents in parallel, by means of reinforcement learning algorithms. The agent shall follow the user's will and identify obstacles along the path, taking the control of the power wheelchair when the user is making a dangerous driving choice. The developed framework, adapted from an existing autonomous driving simulator, has been used to train and test multiple intelligent agents simultaneously, thanks to a customised synchronisation and memory management mechanism, reducing the overall training time. Preliminary results highlight the suitability of the adapted framework for multiple agent development in the assisted driving scenario.

1 INTRODUCTION

In the recent years, more and more interest has been directed to the world of assistive technology, in order to help people with disabilities in their everyday life. Power wheelchairs, voice recognition programs, screen readers, prosthetic and robots are only some examples of assistive devices (US Department of Health and Human Services, 2018; Giuffrida et al., 2019).

Considering physical disabilities, they refer to impairments of parts of the body, resulting in some limitations in mobility (GPII DeveloperSpace, 2020). Approximately 13% of the U.S. population, among adults aged 18 and over, reported having physical disability in 2013 (Courtney-Long et al., 2015), with a similar prevalence in other industrialised countries and over the years.

Power wheelchairs are surely the most significant example of assistive devices for mobility-impaired people. They are wheelchairs with an electric motor rather than manual power and they are usually driven by the user with a joystick. Therefore, they provide a greater degree of freedom. However, they rely heavily on the users' skills, which may not be sufficient to react quickly to dangerous situations, such as the presence of static and dynamic obstacles, roadblocks and other impediments.

Some research teams have tried to overcome collision and obstacle avoidance problems for power wheelchairs, exploiting autonomous driving tools and using different sensors (e.g. self localisation algorithm, on-board lidar, stereoscopic camera and spherical camera) (Leaman and La, 2017; Nguyen et al., 2013a; Nguyen et al., 2013b).

The idea of this project is to realise a smart wheelchair with assisted driving: it should allow the user to drive with confidence during its daily life, through obstacles (both static and dynamic), and, at

[a] https://orcid.org/0000-0003-3306-5698
[b] https://orcid.org/0000-0003-2628-4382
[c] https://orcid.org/0000-0002-6063-7180
[d] https://orcid.org/0000-0001-5426-4974

Falzone, G., Giuffrida, G., Panicacci, S., Donati, M. and Fanucci, L.
Simulation Framework to Train Intelligent Agents towards an Assisted Driving Power Wheelchair for People with Disability.
DOI: 10.5220/0010190301890196
In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART 2021) - Volume 1*, pages 189-196
ISBN: 978-989-758-484-8

the same time, let the user control the wheelchair whenever it will not end up in a dangerous situation. The user is then able to drive himself, but a request of intervention could be raised to the autonomous driving module, elevating the autonomous driving system to a supervisor role. Since a power wheelchair speed is generally limited at $10km/h$, the sensors to mount fall in the short and medium range category. Moreover, due to high cost and high power consumption, the lidar technology (Rasshofer and Gresser, 2005) becomes unfeasible for a smart wheelchair. The selected sensors are then high resolution RGB and depth cameras (Yin and Shi, 2018), together with accelerometers, gyroscopes and tilt sensors. Obstacles can be detected using semantic segmentation of RGB images (Dai et al., 2016). An artificial intelligent agent based on Reinforcement Learning (RL) allows to detect harmful situations and control the vehicle (Caltagirone et al., 2017; Lillicrap et al., 2015), taking in input segmentation and depth images and the other sensors measurements. However, RL algorithms should be trained and tested in virtual environments and then, once the agent has learned, moved into the real world. But the existing simulators for wheelchairs are usually designed to train people in driving a power wheelchair (Pinheiro et al., 2016; Faria et al., 2014; Pithon et al., 2009), not to train intelligent agents, and so they focus on the user interface more than on the scalable architecture (Schöner, 2018).

In this paper, we present a simulation framework for assisted driving on power wheelchair to train multiple synchronised intelligent agents in parallel. It allows to test different neural networks (NNs) while reducing training time. At the end of the training phase, the best performing agent will be mounted on-board the power wheelchair.

After this introduction, Section II describes the software framework, highlighting the chosen simulator and its adaptation for multiple agents parallel training and for the power wheelchair use case. Reinforcement learning algorithms are discussed in Section III, while Section IV presents the preliminary results. Finally, the conclusions are drawn in Section V.

2 SOFTWARE FRAMEWORK

In order to realise an assisted driving power wheelchair, a decisional NN trained by a RL algorithm has been chosen to implement obstacles avoidance in dangerous situations. Hence, before mounting it on a prototype, an interactive simulator has been selected and improved for:

- training multiple artificial intelligent agents;
- testing multiple artificial intelligent agents;
- supporting the dynamic of a power wheelchair;
- generating dedicated sensors;
- generating multiple and dynamic environments;
- synchronising the simulated environment w.r.t. the sensors dynamics and power wheelchairs.

The entire system exploits a client-server architecture with synchronous remote procedure calls (RPC) from each Client to the Server Simulator. It is organised on different levels of abstraction in order to decouple both the operations of interaction with the simulator and of management of the NNs. Figure 1 reports an overview of the entire software architecture, separating the three main components: the Server Simulator, the Manager and the Agent. The latter two compose the Client entity, which directly communicates with the Server Simulator.

The system is designed to be scalable and modular, allowing to run multiple Agents with different NN implementations at the same time with a single Server Simulator.

2.1 Server Simulator

The Server Simulator module should implement a roadside environment in several cities, some of them to be used for training and others for testing. It should also provide a variety of vehicles and sensors and the possibility to change the position of a sensor with respect to the vehicle itself. Moreover, all the sensors should be configurable with respect to the range of action, the field of view, the Signal to Noise Ratio (SNR), etc. For these constraints, we selected the autonomous driving simulator CARLA.

CARLA (Dosovitskiy et al., 2017) is an open-source autonomous driving simulator used in research to develop and evaluate autonomous agents for automotive system. It is implemented as a layer on top of Unreal Engine 4 (UE4) (Epic Games, 2019), which exploits NVIDIA PhysX to simulate the physics of the environment with particular attention to the vehicle characteristics, such as mass, dumping factor, friction of the wheel, moment of inertia, etc.

The simulated maps focus on the urban environment: there are many different maps with different urban objects whose aim is to simulate a variegate real city scenario. Within the simulator, it is possible to configure different actors, represented by pedestrians (e.g. children and adults, women and men), and dynamic objects, mainly represented by

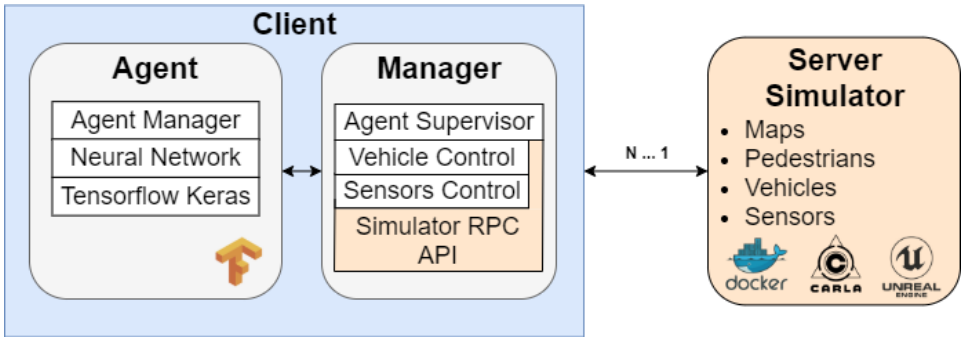


Figure 1: System architecture.

cars, trucks, motorbikes and bikes. The dynamic objects follow the road rules, stopping to the semaphore or waiting for people to cross the street.

The sensors available within the simulation cover the most used ones in the autonomous driving world: RGB camera, depth camera, radar, lidar, GPS and Inertial Measurement Unit (IMU). Moreover, the simulator provides the semantic segmentation of the RGB camera and the collision sensors that detects all the hits suffered by the vehicle.

The Server Simulator exposes the RPC APIs, becoming a stand-alone component, running on top of a docker container (Merkel, 2014): it gives to the Manager the control of the simulation environment, monitoring its parameters and the actors in the simulated world. In this way, the Manager can decouple the Server Simulator from the autonomous Agent which implements the decision component.

However, CARLA is confined in the simulation of automotive vehicles with the standard interface composed by steering, throttle and brake. We can adapt CARLA for the simulation of a different type of vehicle, modifying the control of the car, so that it acts as a power wheelchair. We have simulated the power wheelchair with a BMW Isetta, since it is small and it has a shape similar to a bounding box. The controllers needed to set the accelerations and the velocities of the centre of mass of the car itself are in charge of the Manager.

2.2 Client Overview

The Client represents a simulated power wheelchair, which interacts with the Server Simulator and controls the environment and the vehicle through the Manager, together with the Agent which implements the decisional NN. More Clients can run during the same simulation (on a single server), allowing the parallelisation of the Agents training.

More in detail, the Manager exploits the simulator RPC APIs to control a car to move with the dynamics of a power wheelchair, which uses a joystick as user interface. In fact, the joystick controls directly the torque of the two DC motors and maps its position in the desired linear and angular velocity. The linear velocity is first limited and adjusted with two Proportional-Integrative (PI) controllers, by acting on the throttle and the brake, and finally set in the simulator. In this way, we can avoid peaks of acceleration, which could bring to simulation inconsistencies. Instead, the angular velocity is directly set in the simulator through RPC.

The Agent takes decisions on moving the joystick, as a consequence of the observed state, which is provided by the Manager. The observed state is defined as the position, the speed, the direction of the simulated power wheelchair, the input joystick, which expresses the user's will, and the environment configuration (i.e. the sensors measurements) inside the Server Simulator, in a given moment of simulation.

The sensors mounted on the simulated wheelchair are the RGB and Depth camera and 3 axes accelerometer, gyroscope and tilt sensor.

Then, the NN inputs are the semantic segmentation derived from the RGB image, the depth image, the acceleration on x, y and z axes, the roll and pitch angles and the input joystick position. The yaw and the gyroscope values are available, but we have decided not to use them as input for the NN.

The Agent decision (i.e. the output of decisional NN) is a discreet joystick position chosen from all possible user's joystick outputs (red dots of Figure 2). For our purpose, the NN has to learn to follow the will of the user whenever it represents a safe option. Instead, when the Agent predicts an imminent collision or a harmful situation, the user choice must be bypassed and the Agent has to act as supervisor avoiding the dangerous state but trying to follow the user's will as much as possible.

Therefore, the Agent has to minimise the error between the chosen action and the user's will. This

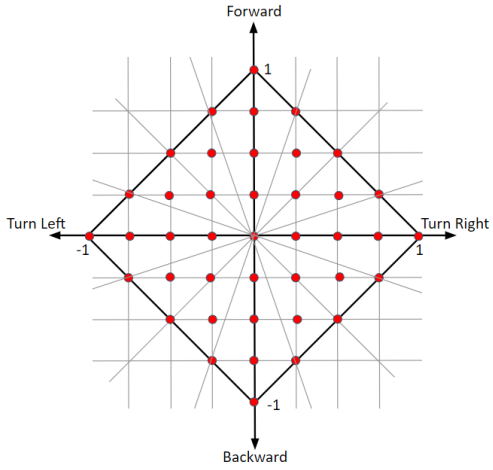


Figure 2: Continuous user and discretised Agent joysticks.

error is the euclidean distance between the input joystick and the output joystick, and it represents the first Key Performance Indicator (KPI), measuring how the agent learns to follow the user's will. The maximum acceptable value is 0.1768, which happens when the user's joystick lies in the centre of the square with base 0.25 (the step between two red dots in Figure 2), according to the following equation:

$$max\_distance = \sqrt{\left(\frac{0.25}{2}\right)^2 + \left(\frac{0.25}{2}\right)^2} = 0.1768 \quad (1)$$

The ability of the Agent to avoid an obstacle or in general a harmful state is measured by counting the number of steps in which the Agent is alive. Each time the Agent dies, a new episode starts. The Agent dies both for crashing with obstacles and if it has reached the maximum number of steps (10,000). For this reason, the second KPI is the length of each episode, which is equal to 10,000 in the best case.

The two KPIs must be evaluated jointly, since the Agent has to follow the user's will except in case, following it, the Agent would end up in a crash.

2.3 Agent Supervisor

The Agent Supervisor module is inside the Manager and implements the interface used by the Agent. It receives all the events and data coming from the sensors in the Server Simulator and it hides all the details of the simulation environment to the Agent, which receives only the observed state. It also computes the rewards and the punishments that the Agent collects for doing a certain action and being in a certain state.

Considering the task of calculating the score of the Agent, the Supervisor gives a positive base reward (+1) when the distance between the output joystick, chosen by the Agent, and the input joystick is less than the maximum accepted distance value (Equation 1). The Supervisor punishes the Agent every time each sensed measurement falls in a range of non-acceptable values, shown in Table 1. So, all the absolute values between 0 and the minimum value of the penalty range are accepted and do not generate any reward or penalty, except for the joystick, for which the base reward is received. Otherwise, if the value of a sensor falls in its penalty range, this value is normalised in the window $[0,1]$ (intensity) and the punishment is given by multiplying the intensity with the sensor penalty (-2). At last, when the absolute value of a sensor is higher than the maximum predefined value, the Agent receives the crash penalty (-100). The Supervisor gives the same penalty to the Agent also when a collision happens. The entire mechanism of rewards and penalties is reported in Table 2.

For each step of execution, the total score is computed by summing the reward with any penalties.

Table 1: Penalty ranges.

Sensor	Penalty window (absolute values)
Joysticks' distance	[0.1768, 2]
Accelerometer (x, y, z)	[[20,40],[20,40], [20,40]] m/s^2
Tilt sensor (roll, pitch)	[[5,20],[20,30]] °

2.4 Agent Memory Management

Since each Agent has to store the states during the episode, a replay memory mechanism is required.

For NNs which exploit the time sequentiality inside a fixed size batch input, the samples are stored from the newest to the oldest in a fixed size

Table 2: Supervisor rewards and penalties.

Static rewards and penalties		
Type	**Value**	**Description**
Base Reward	+1	for a "zero" joysticks' distance
Crash Penalty	-100	collision, rollover, high accelerations

Dynamic penalties		
Type	**Value**	**Description**
Joysticks' distance	-2	multiplied by the event intensity
Accelerations	-2	
Angles	-2	

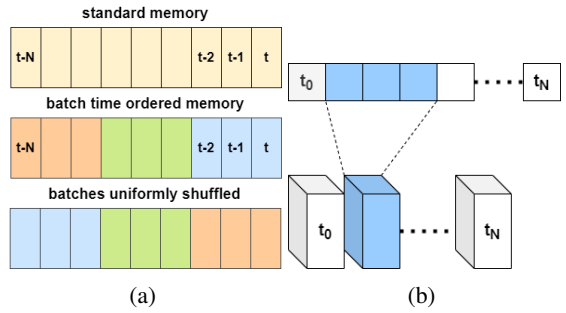


Figure 3: Memory management: (a) grouped sequential memory; (b) samples in another dimension.

First In First Out (FIFO) memory, they are grouped maintaining sequentiality inside groups and, before each training step, the groups are uniformly shuffled (Figure 3a).

On the contrary, if the networks require the time sequence as another input dimension, the standard FIFO memory is navigated grouping the samples in a new dimension (Figure 3b).

2.5 Clients Synchronisation

Since the Server Simulator is unique in the entire system and the Clients can be multiple, a synchronisation mechanism is required between each Client and the Server and among the Clients.

One Client, the first that starts the simulation, acts as master controlling the simulation, while the others act as slaves. The simulation runs at the speed of the Clients, but it is executed on the Server Simulator, which sends a signal when the simulation step has finished and streams sensors data.

Of course, the Clients go at different velocities, also depending on the NN implemented in the Agent. For this reason, before allowing the server to run the simulation step, the master blocks on a barrier, waiting for all the slaves, in order to avoid that some slower Clients lose some steps. Once the barrier is released, the master sends the request of simulation to the Server (*tick request*), through the synchronous RPC. At the same time, the slaves block on a mutex associated with a condition (*tick wait*), waiting for the *end of simulation* signal, sent by the Server when it has finished to simulate the n-th snapshot of the world. Then, all the Clients are released and block again on a mutex with a condition (*sensors wait*), waiting to receive the sensors data from the Server, avoiding that too fast Clients miss those data after the *end of simulation* signal.

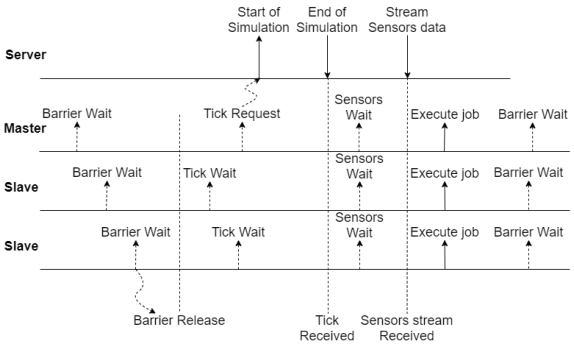


Figure 4: Synchronisation mechanism among clients and server.

Because of this synchronisation mechanism, which is shown in Figure 4, all the Clients execute each step in the same time, at the speed of the slowest, even if NNs have different inference time.

3 REINFORCEMENT LEARNING

Reinforcement learning (Kaelbling et al., 1996) is a machine learning paradigm which allows an autonomous system to create a knowledge of the environment in a game-like situation, exploiting trial and error to come up with a solution to the given problem. The designer sets the reward policy, but the model does not take hints or suggestions for how to solve the game.

The environment is usually described as a Markov decision process in which the probability to take an action in a particular state is given by the expected reward (Figure 5).

In this work, we have used Deep Reinforcement Learning (DRL) (Mnih et al., 2013; Lample and Chaplot, 2016; Zhang and Du, 2019; Balaji et al., 2019) to implement multiple Agents able to follow the user's will and avoid obstacles. The best performing Agent will be mounted on-board the power wheelchair, through a hardware accelerator.

The idea behind DRL is that the NN is trained using a reward function, that is computed every time it picks-up an action for the input state, with the aim of generalising the environment. For each output of the NN, a reward is given by the Agent Supervisor. Hence, after thousands or millions of training steps,

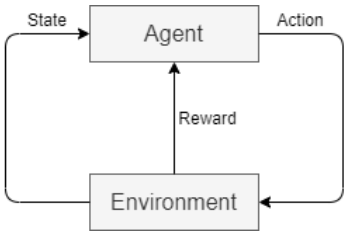


Figure 5: Reinforcement Learning architecture.

the NN learns which is the best action depending on the given state. The aim of the Agent during the training phase is to maximise the rewards received while minimising the punishments.

In this problem, part of the input state (the input joystick) indicates which is the correct action to take in most of the possible states and so we know what the Agent should learn. So, we can force the Agent to learn this behaviour in a supervised way, speeding-up the training of the NN (Action Driven Learning). After the Agent has learnt how to choose actions for the most environment states, we continue with the $\varepsilon - greedy$ RL using a low ε value (Agent Driven Learning). The ε defines the probability to choose a random action in the action space, instead of taking the action chosen by the agent, combining exploration and exploitation (Tokic, 2010).

Therefore, exploiting DRL (Mnih et al., 2013) or Double Deep Q-Learning (Van Hasselt et al., 2015), which solves the problem of overestimation of the action values, we have trained multiple autonomous Agents to move in an environment, respecting the policy defined through the rewards and punishments.

3.1 Neural Network Architecture

With the framework described in Section 2, it could be possible to train in parallel different NNs, varying the type of layers, their size and order.

The Agent with the best performance should be mounted on an embedded target platform; thus, we have to find a compromise on the number of parameters, layer typologies and memory footprint to maintain a low impact on the hardware accelerator. Moreover, the inference time of the NN has to be as short as possible. In fact, it introduces a fixed delay in the joystick control loop. If this delay is longer than the users' reaction time, they may feel out of control of the vehicle. Therefore, we have empirically decided that the maximum possible delay introduced by the agent for our target users is 100*ms*.

In Table 3, the inputs of each NN are reported. IMU refers to the acceleration on x, y and z axes and the roll and pitch angles. All input values have been normalised, to simplify the understanding by the NN.

Table 3: Neural network inputs.

Name	Size	Type
Depth Image	320x240x1	Float32
Semantic Segmentation Image	320x240x3	Float32
IMU	5	Float32
User Joystick	2	Float32

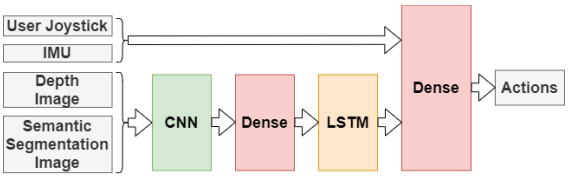


Figure 6: Neural network components overview.

In Figure 6, the most promising NN architecture is reported. It is composed by three main components:

- the CNN block: it is a convolutional block which receives the image taken as combination of the semantic segmentation with the depth one;
- the Long Short Memory (LSTM) (Hochreiter and Schmidhuber, 1997): it maintains the temporal evolution of the images' features through its internal state;
- the Dense layer: it represents the voter of the NN combining the user's joystick and the other sensor measurements with the output of the LSTM.

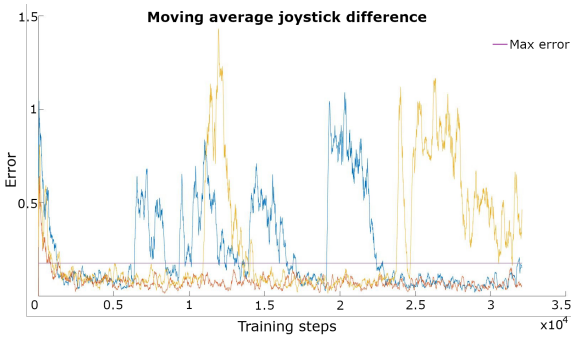
Some other NNs with different architecture and layers have been trained and tested using this framework. For example, a NN with a Convolutional-LSTM (Shi et al., 2015), replacing the CNN block, has been implemented and tested. However, it has been discarded due to the high inference time which arrives up to 1s.

4 RESULTS

The software framework described above has been used to train multiple autonomous Agents able to move in a city following the user's will and avoiding obstacles. The two training phases of each Agent are described in Section 3.

The KPI used to evaluate the Action Driven Learning phase, where the Agent is forced to follow the user's joystick, is the euclidean distance of the two joysticks. Figure 7a reports the results of the first phase of training for three different NNs. The maximum acceptable error is reached in a short number of training steps for all Agents. However, the two NNs represented with blue and yellow lines do not generalise well the policy for the entire first phase. Thus, the Agent represented with the red line, which implements the NN of Figure 6, was selected for the second phase.

The goal of the Agent Driven Learning phase, where the Agent controls the vehicle moving in the simulated world trying to avoid collisions with objects, is to maximise the length of the episode while following the user's joystick as much as possible.



(a)

(b)

Figure 7: Results of the training: (a) joystick difference - Action Driven Learning phase; (b) episodes length - Agent Driven Learning phase.

Figure 7b reports preliminary results of the second phase, showing the length of the episodes during the training of the NN selected from the first phase (Figure 6). The figure underlines how complex, unstable and slow is the Agent Driven Learning phase.

The simulation framework ran on a server with the following specifications:

- AMD EPYC 7301 with 16-Core Processor;
- 256 GB of RAM;
- NVIDIA Tesla T4 with 16 GB of dedicated RAM.

During the preliminary Agent development, the simulation framework showed the capability to train as many networks together as supported by the platform resources. In our case, we tested up to 6 Clients with the same Server Simulator. All the Clients run on the same dedicated NVIDIA GPU Docker, while the CARLA Server Simulator run on a different GPU enabled dedicated Docker.

The synchronisation mechanism allows to train networks with different inference times, maintaining the coherence between two consecutive simulation steps for all the Agents. Static and dynamic objects have been introduced inside the simulated world, to represent a scenario as much similar as possible to a real one. Thanks to the fault recovery mechanism, the simulator has been able to train multiple agents for more than 600 hours.

5 CONCLUSIONS

People with physical disabilities are about 13% of the U.S. population. Some of them are constrained to use power wheelchairs, and they usually have problems for outdoor mobility, on sidewalks or cycling paths, because of the presence of multiple obstacles and their slowness in reacting. In order to overcome dangerous situations, such as crashes or wheelchair rollover, the assistive technology sector seeks to develop intelligent power wheelchairs that can simplify and increase the autonomy of these people. A simulator where training and testing multiple neural network architectures is needed before implementing artificial intelligence directly on a real power wheelchair. However, if several companies and research teams are investing on frameworks for autonomous driving, few studies have been done in the field of assisted driving for power wheelchairs.

In this paper, we presented a simulation framework to train multiple intelligent Agents for assisted driving. The entire developed framework is scalable and modular, since it is totally decoupled from the client architectures. Each client represents a power wheelchair moving in the simulated world and implements a neural network based on reinforcement learning, with the aim of following the user's will while avoiding harmful situations, such as collisions and crashes. All the clients are synchronised with a mechanism of barriers, mutexes and conditions, so that coherence in the simulated world is always maintained. Then, this simulation framework, which comes from the autonomous driving world, allows us to reduce the training time, thanks to parallelism, and to find the neural network with the best performance for assisted driving. In fact, in the simulation, the vehicle is equipped with customised sensors and is controlled as it acts as a power wheelchair (instead of a car). Moreover, multiple static and dynamic obstacles are injected in the simulated world, making it as real as possible.

Preliminary results highlight the capability of the adapted framework to train together many networks for the assisted driving domain. Once selected the best implementation in simulation, it is possible to migrate the Agent in a real embedded platform mounted on a power wheelchair.

REFERENCES

Balaji, B., Mallya, S., Genc, S., Gupta, S., Dirac, L., Khare, V., Roy, G., Sun, T., Tao, Y., Townsend, B., Calleja, E., and Muralidhara, S. (2019). Deepracer: Educational autonomous racing platform for experimentation with sim2real reinforcement learning.

Caltagirone, L., Bellone, M., Svensson, L., and Wahde, M. (2017). Lidar-based driving path generation using fully convolutional neural networks. *IEEE International Conference on Intelligent Transportation Systems 2017*.

Courtney-Long, E., Carroll, D., Zhang, Q., Stevens, A., Griffin-Blake, S., Armour, B., and Campbell, V. (2015). Prevalence of disability and disability type among adults — united states, 2013. *MMWR. Morbidity and mortality weekly report*, 64:777–783.

Dai, J., He, K., and Sun, J. (2016). Instance-aware semantic segmentation via multi-task network cascades. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16.

Epic Games (2019). Unreal engine. https://www.unrealengine.com. [Online]; accessed August 2020.

Faria, B. M., Reis, L. P., and Lau, N. (2014). A survey on intelligent wheelchair prototypes and simulators. In Rocha, Á., Correia, A. M., Tan, F. . B., and Stroetmann, K. . A., editors, *New Perspectives in Information Systems and Technologies, Volume 1*, pages 545–557, Cham. Springer International Publishing.

Giuffrida, G., Meoni, G., and Fanucci, L. (2019). A yolov2 convolutional neural network-based human–machine interface for the control of assistive robotic manipulators. *Applied Sciences*, 9(11):2243.

GPII DeveloperSpace (2020). What is physical disability? https://ds.gpii.net/content/ what-physical-disability. [Online]; accessed August 2020.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.

Lample, G. and Chaplot, D. S. (2016). Playing fps games with deep reinforcement learning. *ArXiv*, abs/1609.05521.

Leaman, J. and La, H. M. (2017). A comprehensive review of smart wheelchairs: Past, present, and future. *IEEE Transactions on Human-Machine Systems*, 47(4):486–499.

Lillicrap, T., Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *CoRR*.

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning.

Nguyen, A. V., Nguyen, L. B., Su, S., and Nguyen, H. T. (2013a). The advancement of an obstacle avoidance bayesian neural network for an intelligent wheelchair. In *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 3642–3645.

Nguyen, J. S., Su, S. W., and Nguyen, H. T. (2013b). Experimental study on a smart wheelchair system using a combination of stereoscopic and spherical vision. In *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 4597–4600.

Pinheiro, O. R., Alves, L. R. G., Romero, M. F. M., and de Souza, J. R. (2016). Wheelchair simulator game for training people with severe disabilities. In *2016 1st International Conference on Technology and Innovation in Sports, Health and Wellbeing (TISHW)*, pages 1–8.

Pithon, T., Weiss, T., Richir, S., and Klinger, E. (2009). Wheelchair simulators: A review. *Technology and Disability*, 21:1–10.

Rasshofer, R. H. and Gresser, K. (2005). Automotive radar and lidar systems for next generation driver assistance functions. *Advances in Radio Science*, 3.

Schöner, H.-P. (2018). Simulation in development and testing of autonomous vehicles. In Bargende, M., Reuss, H.-C., and Wiedemann, J., editors, *18. Internationales Stuttgarter Symposium*, pages 1083–1095, Wiesbaden. Springer Fachmedien Wiesbaden.

Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W. K., and WOO, W.-c. (2015). Convolutional lstm network: A machine learning approach for precipitation nowcasting.

Tokic, M. (2010). Adaptive ε-greedy exploration in reinforcement learning based on value differences. In Dillmann, R., Beyerer, J., Hanebeck, U. D., and Schultz, T., editors, *KI 2010: Advances in Artificial Intelligence*, pages 203–210. Springer Berlin Heidelberg.

US Department of Health and Human Services (2018). What are some types of assistive devices and how are they used? https://www.nichd.nih.gov/health/topics/rehabtech/conditioninfo/device. [Online]; accessed August 2020.

Van Hasselt, H., Guez, A., and Silver, D. (2015). Deep reinforcement learning with double q-learning.

Yin, Z. and Shi, J. (2018). Geonet: Unsupervised learning of dense depth, optical flow and camera pose. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1983–1992.

Zhang, Q. and Du, T. (2019). Self-driving scale car trained by deep reinforcement learning.