MIPRO 2020, September 28 - October 02, 2020, Opatija, Croatia

A Simulator for Training Human Operators of a Remote Controlled Anti-Terrorism Ground Vehicle

Juraj Fulir, Željka Mihajlović, Marija Seder
University of Zagreb Faculty of Electrical Engineering and Computing
Zagreb, Republika Hrvatska
{juraj.fulir, zeljka.mihajlovic, marija.seder}@fer.hr

***Abstract*—Remote controlled robotic vehicles are expensive and require high maintenance. In an effort to reduce costs of training human operators for an anti-terrorism ground vehicle we introduce a simulator implemented in Unreal Engine 4 as an affordable solution. We describe several features that need to be considered when simulating a remotely controlled vehicle guided solely by its integrated cameras. Additional steps are shown for achieving realism and user immersion while maintaining an interactive rendering speed. The cost of rendering multiple cameras is analysed and steps for remedying it are shown.**

***Index Terms*—simulator training; remote controlled ground vehicle; anti-terrorism**

I. Introduction

Remote controlled vehicles are used in dangerous situations and often assist humans through close interaction. This requires fast and precise actions from a safe distance by a well trained operator using a very limited field of view. To acquire such skills, a candidate must practice controlling the vehicle in safe environments. This is costly since the candidate must be brought to the training course, additional resources must be spent to run the vehicle and there is a risk of damaging the vehicle or endangering the training environment (pedestrians, animals, training course objects, etc.). Training in a simulator costs virtually nothing and can be especially useful in the early stages of training, offering an approximate experience of operating the vehicle. Additionally, simulators can offer approximate experience of situations that would otherwise be too dangerous or costly to perform in real-life training, e.g. around explosion and radiation hazards [1]–[3]. Experiences gained in a highly realistic simulator could be transferred to real-life skills adding great value its usage.

Simulators have several components that make them realistic in different aspects. The physical simulation of vehicle's movement and its interaction with the world is a crucial component. It makes the experiences gained in the simulator transferable to real-life skills. A replica of the user interface and the ability to connect to an official controller allow the candidate to adjust to the controls and learn efficient usage of limited information available. Scenarios for gaining targeted experiences are deigned by experts and prepare the candidate for real life situations. Finally, increasing the rendering quality helps immersing the candidate into the virtual environment and treating the scenarios presented as real. To maintain this illusion the rendering speed must be high at all times which poses constraints on the amount of detail and dynamics of all of the above mentioned components [4], [5]. Realistic simulators are additionally useful for training machine learning models used in computer vision and robotics [6]–[8].

Game engines are generally designed to offer various libraries, tools and optimised features. These allow fast development and adjustments for leveraging between visual quality and performance according to need [9]. Generally, they support multiple standards for assets and offer their own which enables fast prototyping with high quality results, making them very handy for building simulators. In this work we describe features, encountered problems and proposed solutions for a simulator built using Unreal Engine 4 (UE4) [10] for operating a remote controlled anti-terrorism ground vehicle. The simulated vehicle is a prototype based on the DOK-ING MV-3 model [11].

In section II we review several popular simulators similar to ours built as standalone tools or projects within game engines with usages in training human or AI operators. In section III we discuss implementation details of various physical properties of the simulator and in section IV how the vehicle interacts with its environment. In section V we discuss performance issues that arise from simultaneous rendering of multiple camera views and in section VI methods for achieving a realistic and immersive environment at small rendering cost. In section VII we review results of proposed solutions and finally propose several directions for further research.

II. Related work

As remarked in [9], game engines contain many readily available elements necessary for building a simulator. While some simulators are built as a standalone tool, most modern simulators just offer a layer of abstraction over the physics engine. Usually they offer a set of implemented components [12] or use specialised physics engines with an API as a plugin for game engines [13], [14]. This way the development of the graphical and physical end of the simulator is separated allowing development teams to specialise in their respective areas. With the ever growing interest in game development, the game engines continually increase both the usability and performance allowing the development of realistic simulators.

Gazebo [12] is a standalone simulator popular in robotics. It offers various implementations of joints, actuators and sensors by fostering the power of an external physics engine. Programs developed for a simulated model are transferable to the real model. The simulator however offers modest set of tools for realistic graphics which limits its usage for training human operators. Several solutions were developed for using a game engine to improve visual quality while supporting the development of transferable programs [6], [15].

Simulators can implement various levels of physical complexity. Some were developed to achieve high precision simulation of vehicles and their surroundings. These include modelling the deformation of ground under construction vehicles [16] and precise mechanical response for various vehicle performance assessments [13], [14]. These simulators require high-end or even specialised hardware for the complex physical simulation which raises the cost of usage. Other simulators focus on the concept of serious gaming, focusing on immersion and rapid decision making while achieving a decent amount physical reality. These simulators are often used by the military for strategy training of teams. Due to their lower simulation costs, these simulators often offer state of the art real-time graphics quality and can be run on consumer class hardware. Simulators that require both high fidelity graphics and precise physics are often used for training and validating machine learning algorithms [8], however they require specialised hardware clusters. Lower complexity solutions also exist for the domain of machine learning [6], [7].

Multi-view rendering is common in multiplayer games where multiple users remotely interact with the same virtual scene. These are usually centralised systems with a server entity which processes some or all of the physics or rendering related tasks of multiple clients. The server can be one or more specialized machines with various connection strategies. [17] These approaches differ from ours as we focus on single user interaction and require the entire simulator to run on a single consumer class machine. This gives us the possibility of connecting an external remote controller and controlling the simulated vehicle for training purposes. In virtual reality applications two cameras close together are used to create a stereo image of the scene which is a special case of multi-view rendering. Due to proximity of cameras scene elements visible from one camera are similarly visible from the other, so some resources or even pixels can be reused when rendering the other camera. In our case vehicle mounted cameras are distant from each other and their fields of view are mostly exclusive which reduces the effectiveness of reusing information.

III. Physical fidelity and movement

The PhysX Vehicles plugin for UE4 comes packaged with the engine and is a highly configurable plugin, containing implementations of several powertrain models for cars. It uses the PhysX vehicle template for simulating vehicle movement. However, it currently supports up to 4 wheels and doesn't support the application of torque to the left and right wheels independently, referred to as differential steering. This makes the plugin unusable for simulating tracked vehicle movement. The PhysX Vehicle SDK also contains an implementation of tank movement which supports differential steering, however we find it unusable for our application. The wheel collision with the ground is approximated using a ray-cast in the $-\vec{z}$ direction of local space. Based on the length of the ray a suspension spring is simulated and used to suspend the vehicle body. This approximation works well on relatively flat terrain and when vehicles aren't expected to climb on steep surfaces. However, we expect the vehicle will have no suspension and at low speeds on rough terrain ray-casts without suspension perform unnatural hard oscillations. Since we expect the intended environments will contain rubble and sharp fragments we replace the single ray-cast collision with full body collision.

The tracks are approximated with a finite number of wheels and a cuboid shape to fill the gaps between wheels, as shown in Fig. 1. The wheels are represented by spherical collision bodies and are positioned at locations of the wheels supporting the track in the real vehicle, shifted lower so the track's outer perimeter touches the perimeters of wheels. This way the weight distribution necessary for drive force calculation is well approximated. Wheels are connected to the body using a movement restricted bone, allowed only to freely rotate in the direction of movement, around the vehicle Y axis, where the X and Z axes are respectively the forward and up direction. Between the wheels there are gaps present, which can cause the vehicle to get stuck in sufficiently thin and tall obstacles. The real vehicle has hard track guides which prevent this effect. To approximate this we use an additional collision body in the shape of a cuboid. The cuboid is placed slightly above the track's outer perimeter to allow the wheels to contact the ground, but block sharp obstacles from getting between the

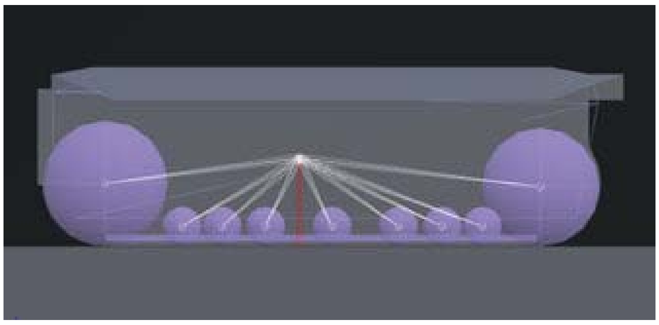

Fig. 1. A view of the vehicle with highlighted collision bodies and bones. The purple objects represent collision bodies, connected to the red root bone by individual white bones. The cuboid fills the space between the wheels and prevents obstacles from jamming them, as expected in the real vehicle. Note that the collision between the overlapping bodies is turned off.

wheels and jamming them. To reduce the energy loss resulting from using only the wheels as force applicators, we remove all friction of the cuboid.

The vehicle can be controlled via discrete and continuous input signals. Discrete signals are used for discrete actions such as changing a gear or setting the handbrake. Continuous signals are used for throttle and steering. However, in absence of an analog input device, discrete input devices such as a keyboard can be used by gradually incrementing the input value through time to simulate analog inputs. This allows us to use both input types in case the controller input specification changes. To simulate throttle we set the constant to positive if throttle is non-zero and to negative when throttle is near zero. This produces a linear change which can be easily adjusted to vehicle specification. The vehicle engine supports two gears: one for normal movement and one used for gaining momentum when punching through objects. The sound of the engine is used to give additional information about the engine to the user so we modulate it based on the throttle value, which corresponds to the change in engine RPM.

IV. Interaction with the world

The terrorists can hide behind a solid object or in a building to achieve a strategic advantage. To overcome this the vehicle has a front mountable spike designed to penetrate through certain vertical barriers, such as glass or thin wooden walls. Destructible objects can be simulated and created almost effortlessly using the PhysX APEX plugin which converts solid meshes into broken meshes when hit with sufficient force. Tuning various parameters allows emulating resistance of different materials. Adding sound cues and smoke particle effects to the broken items gives a more believable look to the action of punching through a wall. To keep the rendering speed in a scene overflowed with fragments, each fragment can be given a random lifetime after which it is removed from the scene. The vehicle can be mounted with a shield used to protect the people and items it carries from enemy fire. Over some terrain types dust can be generated. Dust particles are generated randomly with probability proportional to the vehicle velocity. The landscape information is obtained by shooting a length limited ray-cast in the $-\overrightarrow{z}$ direction of the local space.

Due to its high power the vehicle can be used for towing objects in the scene with a chain or a rope. To tow an object the vehicle must first get close enough to the object. To test whether the object is in range for towing an invisible collision volume is placed around the vehicle. The collision volume is set only to query the collision with the world and can be set to filter unwanted collision. If the object is in range and is tagged using a specified string tag, it can be connected by pressing an action button. The object gets connected to the vehicle via a physical constraint in a form of maximum distance of its anchors. To cue the user that a connection was established we add a simulated rope between the two anchors, defined on both objects as sockets.

On the top of the vehicle various items can be carried. The main expected usage is driving wounded people away from the danger zone and bringing heavy equipment where it is needed. To carry an object we define a virtual socket on the vehicle. We reuse the same collision body used for towing to check if an object is in range for carrying. On the press of an action button the object gets attached to the specified socket and can be carried. Prior to attachment we must disable collision of the body and the simulation of physics, otherwise the attachment constraint becomes unstable and breaks. When detaching the item, the collision and physics properties are restored and the object is placed to the right of the vehicle.

V. User interface and display

The vehicle will have multiple cameras mounted on various locations to allow the user a view of the vehicles' surroundings. The video feed must be sent to a distant operator via a low bandwidth communications channel which restricts the view to a single camera at a time. Future work includes optimisations that will enable simultaneous feed from multiple cameras so we focus on that scenario. Additionally, we render an operator's view of the scene which simulates the expected usage. Fig. 2 shows the user interface and the humanoid used for the operator's view. The third-person view towards the vehicle is in the bottom left corner.

To display images from multiple cameras first we render them to textures and then draw them on the head-up display. Each rendered camera requires a separate render query which increases the rendering cost. Since camera views are will be small in the user interface, we reduce the resolution of target textures. The image quality reduction isn't noticeable, but the rendering cost is decreased.

To allow the user to focus on a single camera with more detail, action keys are added to cycle through different

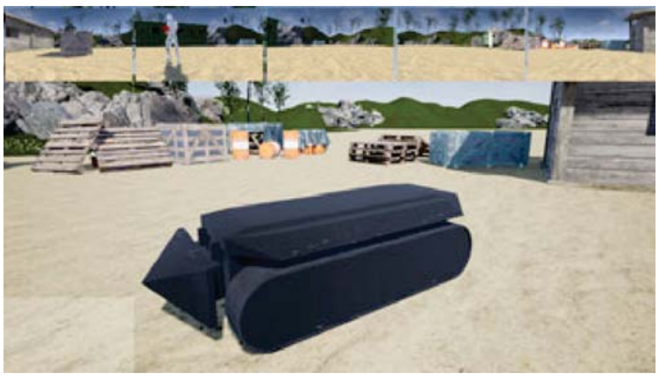

Fig. 2. The user interface displays all camera sources in reduced resolution to preserve interactive rendering speeds. The active camera is indicated by a transparent rectangle, in this case the third person view. The vehicle cameras shown on top are in order: rear left, front left, front, front right and rear right.

cameras. When a camera is selected it is displayed across the screen in its full resolution and its window is marked with a semi-transparent window. The small window of the current camera is not shown because it requires an additional render query which unnecessarily degrades performance, as discussed in section VII-B.

VI. Visual fidelity and performance

Higher visual fidelity allows the user immerse into the environment and approach the simulator training more seriously. High quality models, realistic textures and adding as many varieties as possible are essential parts. Colour varieties provide good variation and can be simulated efficiently using Perlin noise mapping based on the world xy coordinates. Geometry can be shared between instances to save memory at the expense of similarity between them. A different approach would be to build trees procedurally with unique and randomised shapes using algorithms described in [18]. However, the cost of building and rendering lots of trees in this way would be overwhelming considering the number of trees and the amount of work needed for other tasks such as physics simulation and world interaction. On the other hand, simulating a natural environment requires more than just realistic visuals. If a natural scene is static it looks very artificial. Adding just a small amount of branch movement simulates wind and increases fidelity of the natural environment. The wind can be simulated within UE4 using the 'SpeedTree' plugin and compatible tree models.

It makes sense to put certain vegetation on landscape types that in nature support it. This is achieved by limiting on which landscape layer foliage can be put. Instead of putting instances one by one, they can be massively placed using brushes or generated procedurally. Procedurally generated instances can be placed using procedural volumes or using the landscape material. Procedural volumes can be placed in the scene and define the bounds within which vegetation can be spawned. Landscape material can be used to procedurally spawn multiple different instances on specific landscape layers. This is a very simple method that is very configurable and yields great results. However, it still doesn't support collision on spawned meshes which makes it useful only for certain types of vegetation. Brushes are used to paint on the terrain texture where instances can be spawned and the instances are statically placed. This method allows for a finer control of instance distribution and supports meshes with collision, which makes it useful for placing trees, rocks and other objects.

In practice, cameras of remote controlled vehicles can be occluded by clouds of dust from vehicle movement. To simulate dust generation we cast a ray from each track. Based on the type of hit material, we generate an adequately tinted dust sprite depending on probability proportional to track speed. Dust generation should not be too frequent to avoid generating non-transparent clouds and randomisation gives a more natural look to the simulated effect.

VII. Results

All experiments were executed on a single Lenovo Legion Y520 laptop with an Intel i7-7700HQ chip, 8 GB of RAM and a NVIDIA GTX 1050 graphics card with 4 GB of VRAM. These specifications are representative of the expected systems to run the simulator in practice. The Unreal Engine version is 4.22.1 with the OpenGL version 4.6.0 backend and PhysX 3.4 as the physics engine.

A. *Track collision using the wheel collision body*

Fig. 3 depicts traces of the two approaches in implementing track collision without suspension. The ray cast collision shows a sharp spike when it encounters the edge of an obstacle causing the entire vehicle to jump unnaturally. Using the full collision body for wheel collision the trace is much smoother and provides a more natural transition. In high speed vehicles with suspension the effect of sharp collision is softened using the simulated suspension springs and the jumping effect is unnoticeable. As we expect this tracked vehicle to operate at low velocities without suspension, the full wheel collision is a better approach to implementation.

Fig. 3 also points out a problem with the approach of approximating a track using multiple wheels. In the upper image obstacle clearly penetrates the track and occasionally causes the vehicle to get caught in it. The naive implementation using only wheel colliders suffers from the same problem. The addition of a cuboid to the track collision keeps the vehicle suspended over the obstacles and greatly reduces

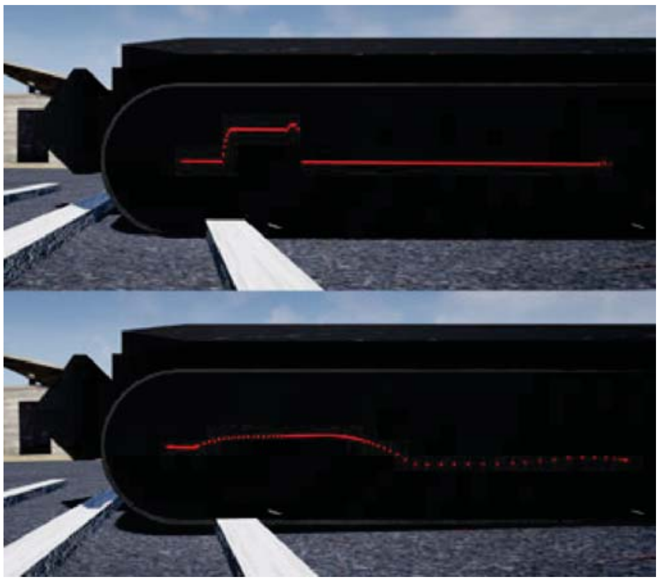
Fig. 3. The trace represents the movement of a front wheel through multiple consecutive frames. The top image shows a sharp wheel jump when the ray-cast collision detects the difference in surface elevation. The bottom image shows a smoother trace using the full collision body of the wheel. The traces in the figure are only indicative and do not match the exact horizontal coordinate of the obstacle due to perspective projection of the camera.

the chance of jamming the wheels. The vehicle still changes elevation as it traverses over individual obstacles due to the height difference between wheels and the added cuboid, but in practice the effect is barely noticeable. When traversing a regular pattern of obstacles, the vehicle still might get stuck if none of the wheels have contact with the ground. However, this scenario is very improbable due to multitude of conditions needed for it to happen that are not common in the expected environments. These include: pattern of obstacles must match the pattern of passive space between the wheels, vehicle must be perpendicular to the obstacle and vehicle must be stationary due to no friction constraint on the cuboid.

B. *Measuring the cost of multiple cameras*

The user interface requires simultaneous display of the feed from multiple vehicle mounted cameras. Rendering multiple cameras causes additional queries for scene rendering which adds a significant penalty to the total time cost per frame. The cost of displaying a high resolution interface is proportional to the expected cost of streaming a video from a remote vehicle through limited bandwidth wireless communication. We test the effect of interface resolution, number of simulated cameras and the resolution of camera views on the drop of the overall frame rate.

We use two resolutions expected to be found in the device for remote control: 1920×1080 and 1280×720. For each device resolution we test several scenarios, depending on the number of cameras: a single camera (front), 3 cameras (front), 5 cameras (3 front and 2 back) and the same 5 cameras with an additional third-person view. Vehicle mounted cameras were rotated from the front facing one by $\pm 45°$ for front ones and $\pm 135°$ for rear ones. The front facing camera was always used as the active camera displayed in the background. For each of these setups we test several resolutions of camera views. Since our interface divides the horizontal screen space into 5 parts, we ensure both tests include the camera resolution that is exactly one fifth of the screen resolution, which will maintain the original pixel density. The camera view resolutions sorted by the number of pixels are: 128×72, 256×144, 384×216, 512×288. Larger resolutions were not tested as they show no visible improvements in image quality in direct display and are not to be expected in practice. For each experiment we record 1000 frames after a first 5 second delay. The delay allows the resources to load and the frame rate to stabilise for an accurate reading. The physics engine in conjunction with the game engine produces stochastic vehicle trajectory for the same starting conditions. To reduce the possibility of unfair estimation while maintaining tractability, each of the above experiments was repeated 5 times. Finally, the temporal cost of each frame over all repetitions were averaged to produce the final results displayed in Fig. 4.

The vehicle was placed into a relatively open part of the map and forced to drive forward with occasional turning right according to a simple delay based flip-flop automata. The

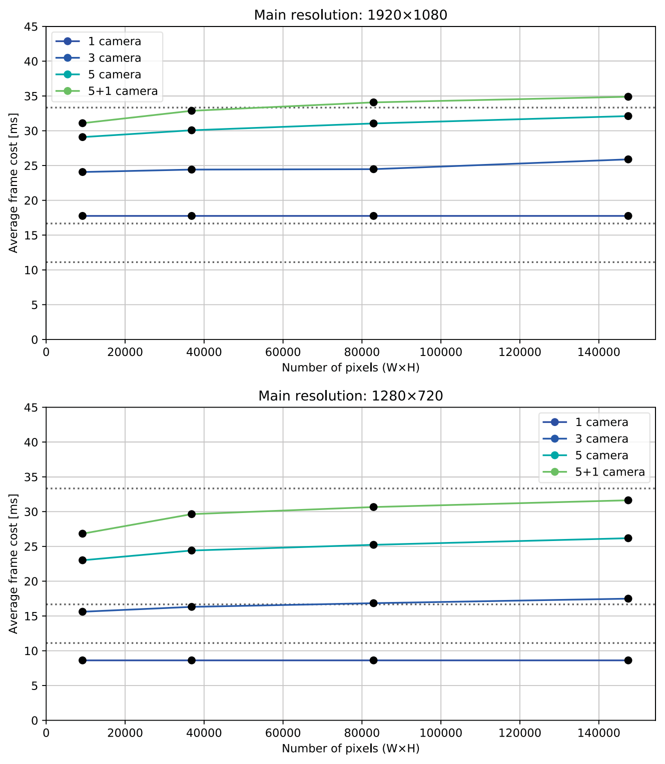

Fig. 4. Graphs represents the average time to render a frame over a number of pixels for each of the camera view resolutions. Each line represents a setup depending on the number of cameras. Each graph presents measurements at a given resolution of the main viewport. The dotted lines represent the rendering costs at 30, 60 and 90 FPS. The values in the X-axis represent the number of pixels in a given camera view resolution.

turning was stronger then usual to allow for the vehicle to turn quickly enough so that all cameras catch a part of the dust particles before they disappear. The dust particles cause significant drops in frame rate due to being semi-transparent and numerous. By allowing all cameras to capture it, we guarantee that some setups aren't falsely better. Dust particles are important for our testing since we expect dense generation during the training procedures to increase candidate robustness. The surroundings include large and small objects, both of static and dynamic nature. In the beginning of simulation three destructible objects were dropped from the sky and broken to increase the number of dynamic objects and scene complexity.

Fig. 4 shows results of the experiments. The larger resolution of the main viewport shows generally larger costs of rendering a frame. We can see that even the lightest configuration on high resolution viewport doesn't achieve the 60 FPS threshold. This suggests a possibility of using various methods of upsampling to utilise the speed of rendering a low resolution screen by sacrificing some visual details. By adding additional cameras to the user interface the costs rise significantly. Each camera approximately adds additional 2.5 ms, rapidly escaping the 60 FPS threshold for the lower resolution viewport. This indicates a large advantage of hiding as much of camera sources as possible in the user interface.

This is the reason we hide the current view as described in section V. Note that this makes the cost for the one camera configuration same for any camera view resolution since it is hidden from the user interface. The cost of adding additional cameras is more or less the same for all scenarios.

Increasing the camera resolution, within reasonable bounds, has little effect across different number of cameras compared to the cost of drawing the main viewport and requiring several rendering queries. Therefore it is not necessary to reduce the resolution of views at the cost of aliasing effects and detail loss in an already small image. In our case, fifth of the screen resolution maintains pixel density of the views and is enough to display the scenes. However, we observe slightly better image quality on a resolution larger then the optimal one.

VIII. Future work

The current implementation is sufficient to give initial experience in manoeuvring the vehicle over rough terrain and in tight places, using the limited information available through the vehicle mounted cameras. A designed user study of candidates would reveal the true effectiveness of the simulator in training. Due to time and space constraints we believe that the effectiveness study is outside of the intended scope of this paper and is left as future work.

Dynamic scenarios and specific goals should be specified to immerse the candidate into the simulated environment. Both need to be challenging but representative of real life situations the candidate might encounter. Future work might include scenarios designed in collaboration with experienced pilots to increase the value of experiences gained through the simulator.

When constructed, the original vehicle controller will communicate with the simulator. Although rendering cameras directly to the framebuffer could lower the render time, our goal is to use render targets which will later allow us to simulate the image streaming from a vehicle to the original controller. To compensate the cost of image transfer from the vehicle to the controller, smarter approaches to information presentation and transfer will be investigated.

IX. Conclusion

A simulator of a remote controlled anti-terrorism ground vehicle for usage in human operator training is presented. Several features required by the vehicle and their implementation are described. The implementation of tracks is compared to the existing solution for vehicles in UE4. The costs of displaying multiple cameras for remote control are compared and analysed to point out main problems. The largest cost comes from displaying multiple cameras in the user interface so their count should be specified wisely. Their resolution in the user interface has little cost when within reasonable bounds and can be safely increased to match the needs of the interface.

Acknowledgement

This work has been supported by the European Regional Development Fund under the grant KK.01.2.1.01.0138 - Development of a multi-functional anti-terrorism system (MAS). We gratefully acknowledge the NVIDIA Corporation for their donation of the Titan V GPU used in this research.

References

[1] M. Đakulović and I. Petrović, "Motion planning of mobile robots for humanitarian demining," in *10th International IARP Workshop HUDEM'2012*, 2012.

[2] A. Šelek, D. Jurić, A. Čirjak, F. Marić, M. Seder, I. Marković, and I. Petrović, "Control architecture of a remotely controlled vehicle in extreme CBRNE conditions," in *2019 International Conference on Electrical Drives Power Electronics (EDPE)*, pp. 273–278, Sep. 2019.

[3] J. Peterson, W. Li, B. Cesar-Tondreau, J. Bird, K. Kochersberger, W. Czaja, and M. McLean, "Experiments in unmanned aerial vehicle/unmanned ground vehicle radiation search," *Journal of Field Robotics*, vol. 36, no. 4, pp. 818–845, 2019.

[4] M. Meehan, S. Razzaque, M. C. Whitton, and F. P. Brooks, "Effect of latency on presence in stressful virtual environments," in *IEEE Virtual Reality, 2003. Proceedings.*, pp. 141–148, March 2003.

[5] S. P. Sudarsan, C. J. Cohen, L. Q. Du, P. N. Cobb, E. S. Yager, and C. J. Jacobus, "Influence of video characteristics of simulator images on remote driving performance," in *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, vol. 2, pp. 1062–1066 vol.2, Oct 1997.

[6] Microsoft, "AirSim." https://microsoft.github.io/AirSim. Accessed: 2020-02-06.

[7] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16, 2017.

[8] Nvidia, "Nvidia Drive." https://developer.nvidia.com/drive. Accessed: 2020-02-06.

[9] J. Craighead, R. Murphy, J. Burke, and B. Goldiez, "A survey of commercial open source unmanned vehicle simulators," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 852–857, April 2007.

[10] Epic Games, "Unreal Engine." https://www.unrealengine.com. Accessed: 2019-12-18.

[11] "DOK-ING." https://www.dok-ing.hr. Accessed: 2020-02-06.

[12] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, pp. 2149–2154 vol.3, Sep. 2004.

[13] CM Labs, "Vortex Studio." https://www.cm-labs.com/vortex-studio. Accessed: 2020-01-20.

[14] Mechanical Simulation, "CarSim." https://www.carsim.com. Accessed: 2020-01-20.

[15] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "Usarsim: a robot simulator for research and education," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 1400–1405, April 2007.

[16] D. Holz, A. Azimi, and M. Teichmann, "Advances in physically-based modeling of deformable soil for real-time operator training simulators," in *2015 International Conference on Virtual Reality and Visualization (ICVRV)*, pp. 166–172, Oct 2015.

[17] W. Cai, R. Shea, C. Huang, K. Chen, J. Liu, V. C. M. Leung, and C. Hsu, "A survey on cloud gaming: Future of computer games," *IEEE Access*, vol. 4, pp. 7605–7620, 2016.

[18] H. Nuić and Ž. Mihajlović, "Algorithms for procedural generation and display of trees," in *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 230–235, May 2019.