19th International Conference on Modelling and Applied Simulation
17th International Multidisciplinary Modeling & Simulation Multiconference

ISSN 2724-0037 ISBN 978-88-85741-48-5 © 2020 The Authors.
DOI: 10.46354/i3m.2020.mas.004

Using a virtual dataset for deep learning: Improving real-world environment re-creation for human training

Troyle Thomas[1], Jonathan Hurter[1] , Terrence Winston[1], Dean Reed[1,*], and Latika “Bonnie” Eifert[2]

[1]Institute for Simulation and Training, 3100 Technology Pkwy, Orlando, FL 32708, USA
[2]U.S. Army Futures Command, CCDC-SC, STTC, 12423 Research Pkwy, Orlando, FL 32826, USA

[*]Corresponding author. Email address: dreed@ist.ucf.edu

Abstract

An ideal tool to re-create real-world environments as virtual environments, by utilising real-world imagery, would be effective and efficient. These virtual environments have applications in training. In this paper, a focus is given on how to detect real-world objects from an unmanned aerial system’s sensors, and in turn, inject corresponding objects into a virtual environment. As a step towards this ideal tool, the You Only Look Once (YOLO) object detection model (a type of machine learning algorithm) was trained on virtual models of poles (e.g., light poles), and in turn, tested on recognising poles. A precision-recall curve was used for performance results. Final analysis suggests a large domain gap between the virtual models used and their real-world counterpart, due to the fidelity of the virtual models; our 3D-modelling technique is contrasted with other techniques from previous literature. Further, this paper details a novel Unity Terrain Importer tool, as it applies towards a re-creation pipeline. The importer tool is oriented to reduce current fidelity and performance limitations in the Unity game engine.

Keywords: Machine learning; Unity; terrain fidelity; training; virtual environments

1. Introduction

One form of training involves virtual simulations; these simulations can offer lifelike scenarios to prepare for real-life scenarios. A virtual simulation involves humans operating systems that are synthetic. Similarly, serious games can offer virtual systems to play and learn within; nevertheless, serious games favour fun at the expense of fidelity, when compared to educational simulations (Aldrich, 2009). Training can happen in a Virtual Environment (VE) that mimics reality to promote appropriate transfer to a Real-World (RW) referent. Further, live simulations (i.e., where humans operate in real environments) for training can also involve the use of VEs, where a correlation is made between live, or RW, features and VE features. For example, Augmented Reality (AR) may be used to bridge the real and the virtual: when a Soldier is practising shooting in a RW environment, an AR overlay of a moving person, or virtual agent, may be used as a target. This moving person could hide behind hills that provide cover, where a hill’s visual and physical profiles closely match between the RW and the VE.

Ideally, VEs for live and virtual simulation training would not only mimic reality with high fidelity (i.e., realism) for the sake of ecological validity, but also be constructed rapidly and efficiently. The ideal method envisioned is for a Machine Learning (ML) algorithm to automatically detect RW objects through an

Unmanned Aerial Vehicle's (UAV's) sensor-feed, before these objects would be automatically imported into a VE; further, the ML algorithm would be trained on virtual data, as humans may cause errors when annotating data, human annotation requires extensive time, hard-to-access (e.g., remote-located) objects are accessible from a computer via 3D models, and a range of conditions expected for outdoor environments (e.g., variations in weather and lighting) may be easily replicated. Ultimately, the VEs would be constructed through automated generation. Militaries are stakeholders for such live and virtual simulation training. Nevertheless, re-creating scenarios specific to RW environments has potential for other training needs.

The ideal of an effective and efficient environmental re-creation tool frames the issues tackled by this paper. We investigate the solution of training a ML algorithm, specifically the You Only Look Once (YOLO) object detection model, to detect poles (e.g., light poles) from overhead imagery. Testing this approach is one issue covered in this paper. Another related issue addressed by this paper is covered by a discussion of a novel tool, the Unity Terrain Importer (UTI) tool, for handling terrain in the Unity game engine. In our discussion of the tool, terrain relates to an environment, as a terrain is a digital representation of a geographical area, and can be contained within terrain meshes and Unity terrain objects.

1.1. State of the art

Novelty exists in the specific case of object detection through a virtual dataset of virtual pole-models shot from an orthographic angle. Nevertheless, the overarching idea of using virtual datasets in the realm of computer vision is viewed elsewhere. For example, semantic segmentation has been investigated, including Khan (2019) focusing on a road-level view, and Ros, Sellart, Materzynska, Vazquez, and Lopez (2016) using urban images derived from a virtual city. For object detection, a focus on vehicles is common (Johnson-Roberson, Barto, Mehta, Sridhar, & Rosaen, 2017; Tian, Li, Wang, & Wang, 2018), and one focus has been given to the singular case of berms (or hills; Reed, Thomas, Reynolds, Hurter, & Eifert, 2019).

This paper furthers past research by using virtual, handmade, and 3D pole-models that were manually placed in the Unity environment with existing terrain. The training dataset consisted of aerial images of the virtual pole-models placed in an existing Unity terrain (with RW poles removed), and the testing dataset consisted of aerial images of RW poles within the existing Unity terrain (with virtual pole-models removed). A second, critical effort is the ongoing improvements in terrain fidelity as expressed in a VE rendered via the Unity game engine. We delve into issues encountered in the Unity engine, while also using the engine as a source of substitute data for the RW-data component when training YOLO. This paper provides approaches and measured performance improvements collected in the course of the ML experiment.

1.2. Application areas

Military training simulations will directly benefit from the application of the concepts and solutions discussed in this paper. The U.S. Army's Common Synthetic Environment (CSE) is a unified simulation environment that Units and Soldiers use for training. Components of the CSE include Training Management Tools (TMTs), Training Simulation Software (TSS), One World Terrain (OWT), Architecture, Information Assurance/Cyber Security, Data, Trainer Interfaces, Integration, and Interoperability. The Synthetic Training Environment (STE) is a cross-cutting capabilities software, with application(s) and services for Live, Virtual and Constructive (LVC) training that will provide the Soldier with the repetitions necessary to rapidly master collective multi-echelon skills. These skills lead to success in Multi-Domain Battles. The OWT component provides the digital representation of the dynamic Operating Environment (OE) needed to support training; OWT consists of the collection, creation, storage, distribution, and runtime components of terrain sources; and OWT provides the common, correlated terrain for all Soldiers utilising training systems. Depending on the needs, terrain-processing services can require significant human-in-the-loop efforts to provide digital terrain for virtual games, such as games created through Unity and Unreal. The UTI tool can be used to create VEs, while overcoming the challenges discovered in processing large data sets.

Other application areas might also benefit from downstream uses of our approach. The serious game framework of Flood Action VR (Sermet & Demir, 2018) incorporates RW weather and geographical data to support virtual training concerning preparing and responding to disasters. Perhaps future variations of training similar to Flood Action VR could be enhanced (in both fidelity and on-demand needs) by automatic 3D-object placement. Another potential downstream use of the re-creation framework is for building high-fidelity outdoor settings for virtual field trips; one example of using a virtual field trip to a simulated nature reserve is found in work by Harrington (2010).

2. The Unity Terrain Importer Tool

The RW re-creation pipeline necessitates an engine for rendering a VE. For the ends of this paper, the Unity game engine was chosen for re-creation. Nevertheless, Unity contains fidelity and performance limitations with terrain files. This paper responds to the constraints of the terrain-import process of Unity by detailing the UTI tool software. Importable terrain in this context refers to mesh formats, which are a method for storing representations of geographical areas. The tool outputs to a Unity terrain object, which is the program-specific representation of a geographical area (requiring height values and potentially containing terrain textures and 3D models), that we further optimised for performance and fidelity within the game engine. Notably, solutions are given that result in updates in the areas of 32-bit terrain generation, additional import methods, and dynamic terrain adjustment. Before discussion, a few terms should be clarified: a terrain mesh is a 3D representation of a geographical area (requiring height values and potentially containing terrain textures and 3D models), a terrain texture is an image file that is laid on top of the terrain to visually represent the terrain's outward appearance, and a terrain tile is used to describe one portion of a larger terrain whose area is divided into multiple sections of a specified size.

2.1. Terrain generation precision

One issue from importing extensive point-cloud data (where the latter is transformed into terrain meshes and textures) is the loss of runtime performance due to a high level of vertices. Decimation on the terrain mesh may be completed to improve performance, at the expense of fidelity; or the Unity terrain-objects feature may be used to reduce a terrain mesh's performance load. Although the latter Unity terrain-object solution improves performance, it is limited by using a 16-bit heightmap for generation; such a solution is also subject to a loss in fidelity. Specifically, this heightmap represents different terrain heights by varying pixel values, comprising a 16-bit system.

To overcome the aforementioned Unity limitations—both performance issues while importing large terrain meshes and the limited fidelity of 16-bit heightmaps—we created the UTI tool. The UTI tool processes and converts terrain meshes to Unity terrain objects with 32-bit representation to achieve higher levels of accuracy. This achievement is done by sampling heights on a terrain mesh at a user-defined interval and applying the sampled values to a new Unity terrain object. This achievement bypasses the need for utilising the antiquated heightmap import method. While developing the tool, Unity's terrain system was found to limit the output of the terrain object to 5 significant digits of precision, regardless of the input data's fidelity; this limit would conventionally challenge attempts to create a 32-bit terrain. A novel solution involving terrain-object transform manipulations was tested and verified to achieve levels of precision that approached 32-bits of precision. Ultimately, there was no appreciable difference in fidelity when comparing the resulting terrain object to the terrain object generated with the original method. The generated terrain object maintains fidelity with the original terrain mesh, as scaling of the terrain object matches the terrain mesh, and the outputted terrain object becomes textured with the correct terrain textures.

2.2. Performance metrics

Unity terrain objects have seen several advancements in the way of performance: measurable performance increases result from a terrain-specific Level of Detail (LOD) system and a GPU-instanced render path. A GPU-instanced render path allows for a significant reduction in draw calls and, consequently, a large increase in graphical performance. These Unity performance improvements make Unity terrain objects the ideal choice to represent large geographical areas. This ideal choice is reinforced by our own data when comparing the performance of a textured terrain mesh with the performance of the UTI tool (see Table 1). The performance tests were recorded on a Windows machine equipped with an Intel Core i7-6700K (4 physical cores @4.00GHz), an NVIDIA GTX 1070 with 8GB VRAM, and 64GB of RAM.

Table 1. Performance comparison between terrain solutions.

Terrain Format	Average FPS	RAM Usage	CPU Usage	GPU Usage
FBX terrain mesh	11	5.5GB	15%	100%
Dynamic terrain	90	6.8GB	22%	35%

Rendering performance is significantly increased by using dynamic terrain conversion when compared to using FBX filetype meshes. We note a reasonable increase in CPU utilisation as a tradeoff.

It is still possible when converting meshes that span several kilometres into a dynamic terrain, to run into scenarios where the terrain-object result produces performance below a threshold of thirty FPS. Steps were taken to mitigate remaining performance challenges by dynamically managing Unity's Pixel Error attribute. Pixel error is defined as "the accuracy of the mapping between Terrain maps (such as heightmaps and Textures) and generated Terrain. Higher values indicate lower accuracy, but with lower rendering overhead" ("Terrain Settings," n.d.). By adjusting the value of pixel error, the outputted terrain object will modify its visual detail proportional to distance (for example, see Figure 1). The closer the rendering camera is to the terrain object, the more detail is provided. The distance where the higher-detail terrain features become visible is what the pixel-error value represents. Higher pixel-error values are not entirely ideal, however, as they can result in severe visual defects in their temporary decimation of the terrain object.

With large datasets, the pixel-error value must be raised to allow for optimal performance at runtime to accommodate the amount of data being displayed on the screen, resulting in a terrain that is visually degraded when viewed from more considerable distances. A singular pixel-error setting for every section of the environment generally yields either low performance in some areas or too many visual defects in other regions.

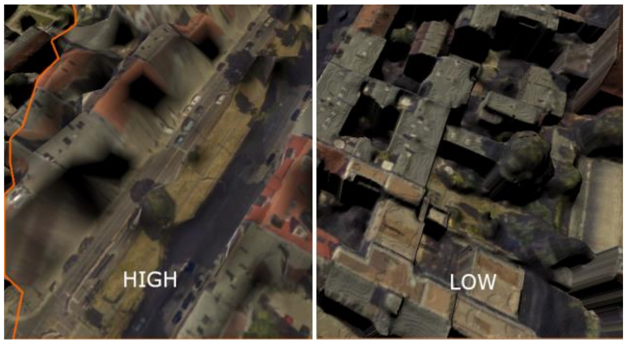

Figure 1. Comparison of terrain at a far range, with a high pixel-error value (left) and low pixel-error value (right)

Our solution to minimise the negative visual artefacts while maintaining rendering performance is to dynamically adjust the pixel error. At runtime, distances between the rendering camera and the Unity terrain objects are utilised to make real-time adjustments to the pixel-error value of each terrain object in the scene. Closer terrain objects are assigned values close to the minimum, and farther terrain objects are assigned values farther from the minimum (see Figure 2). This adjustment solution eliminates the downside of severe decimation when using a uniformly distributed higher-pixel-error value, as those higher values are now restrictively applied to only the terrain tiles farthest away from the camera. The results of the runtime adjustment solution indicate significant performance increases (see Table 2).

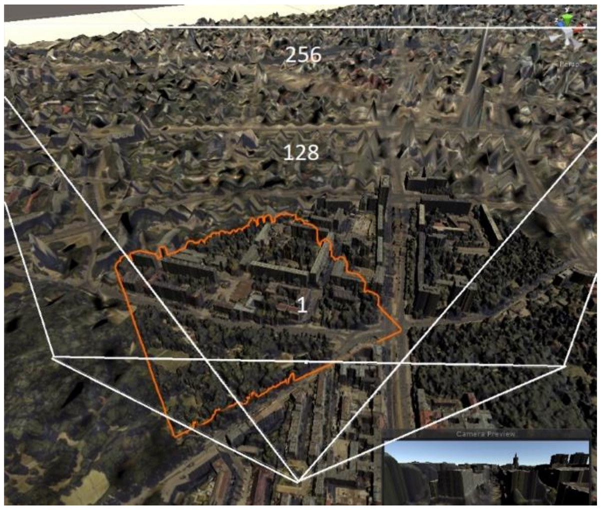

Figure 2. Visualisation of pixel-error adjustment based on camera location in a scene; distance overlay elements are shown in white; a single terrain tile is outlined in orange

Table 2. Example performance impact of pixel-error value and the adjustment solution.

Pixel Error	Average FPS
1	7
10	25
100	100
Dynamic	90

2.3. Future developments

Despite the improved performance the UTI tool offers when compared to the equivalent terrain in mesh format, there are still challenges to be faced in the realm of performance and fidelity.

Although the pixel-error value associated with a terrain object is recognised in section 2.2 as valuable for managing larger terrains, there are still even larger terrains that can demand too many resources for a machine to handle, due to the memory usage involved in such a scene. A terrain paging system could be utilised at runtime to dynamically load terrain objects into a scene as needed and minimise resource consumption (Tian & Lou, 2018).

The inability to render complex geometries, such as tunnels or overhangs, remains a limitation of Unity terrain objects. Further investigation of this limitation could benefit future iterations of the tool, allowing for a more accurate re-creation of an input terrain mesh.

3. Materials and Method: Machine Learning Experiment

The ML experiment involved identifying the capabilities of the YOLO object detection model when trained on virtual pole-models (i.e., various classes of light poles and one class of power pole) and then tested on the RW analogue. The procedure involved constructing the VE, collecting the RW terrain-tile analogue, collecting the image dataset from both the VE and RW, training the model on the VE, and then testing on the RW dataset. Note, both the RW and VE datasets refer to different configurations of the Unity environment: the RW is essentially the same environment as the VE, differing only by the lack of any of the introduced pole models (see Figure 3 for a comparison of the VE and RW). This distinction may differ from how the conception of a RW dataset is traditionally defined.

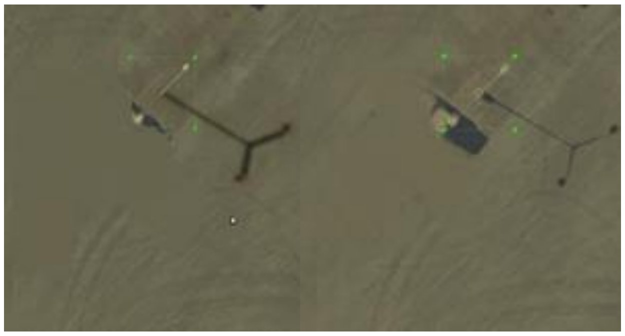

Figure 3. Example of a virtual environment pole-model (left) and real-world pole (right)

Construction of the dataset stemmed from LiDAR scans and manual geotypical placement of the target object (i.e., pole). Collection of the RW terrain tile and height map information came from high-quality data sources. The data was used to create the 3D Unity environment. The terrain tile was then duplicated, and the poles in the imagery were digitally painted away to be replaced with analogous virtual pole-models. The painting was done to prevent the model from already experiencing the RW pole's terrain texture during model training. Thus, the distinction between the RW environment and the VE was based on the presence of virtual pole-models.

All image datasets were collected in the same manner. After the VE Unity environment was created, the image datasets were collected by iterating the camera view through the scene. The camera view was initially oriented to match the RW orientation and then moved by an X and Z delta in a grid fashion to capture the entire scene. Every capture that contained a virtual model was also provided with a coinciding bounding box from the model's bounds. The RW imagery was then captured similarly with the target objects no longer visible, but still using the model-bounds information to create a bounding box. The bounding boxes of the models were projected on to the underlying terrain because, if used directly, the height of the model can incorrectly provide the bounds for the terrain tile for a perspective view. This issue was avoided through an orthographic capture.

The dataset now consists of the captured images and the associated labelled bounding-box annotation. The dataset was then randomly segmented into the training, validation and testing sets (in a 70:15:15 split, respectively) with the appropriate annotations and folder structure for the darknet neural network framework, the latter running under the architecture of YOLO. The training and validation subset consisted of the VE image captures, whereas the testing set used only the RW captures. The validation subset was used to test the model during the training cycle to provide intuition on the performance of the model. The aforementioned process was then repeated for multiple datasets for a regression analysis of the 3D models. This led to the creation of a black, black y-shaped (simply termed, black y), white, white y-shaped (simply termed, white y), and all light-pole datasets. The purpose of this regression analysis was to isolate the 3D-model classes as an independent variable, and thus provide a correlation to the results. This would allow us to identify issues with any particular 3D model.

The results were then expressed in terms of precision and recall for each data set. Precision indicates how many of the model's predictions were correct. Recall indicates how many correct predictions the model determined, out of the total possible correct predictions. Each class has an Average Precision (AP) and Average Recall (AR); both are the average value as the model's prediction threshold changes. A mean Average Precision (mAP) and mean Average Recall (mAR) is provided for multiclass models to provide an overall indicator of the model's performance across all classes. The mAP and mAR are averages of AP and AR across all classes, respectively.

4. Results and Discussion

The results are shown in the form of several trained models in order to provide a regression analysis for each 3D-model class and their impact on the YOLO object-detection model. Figure 4 provides an example of what the output of the model looks like when visualized, with the images showing true-positive examples of the model trained on the light-pole dataset.

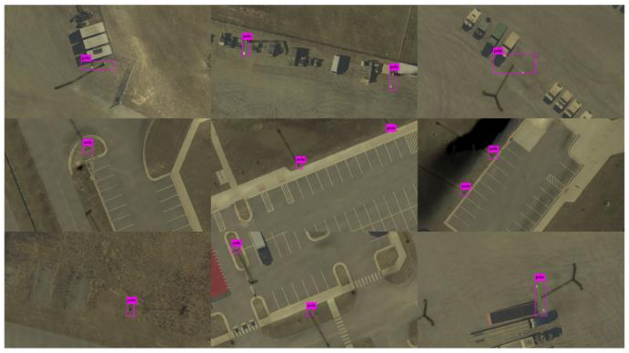

Figure 4. Samples of the trained model's detection output

Each of the following tables (i.e., Tables 3 and 4) show the precision-recall results of the model trained on the stated dataset and tested on the same dataset's VE testing sets, except for one new white light-pole dataset (the last entry in Table 3) that was tested on a VE and RW counterpart. Figure 5 demonstrates the precision-recall curve for each class of the model trained on all light-pole dataset. The precision-recall results of the prior models, when tested on the RW testing set, is not shown since the models failed to provide any detections, had an undefined precision value, and had 0% recall.

As the ML experiment progressed, our initial findings for training the VE model, containing all four 3D light-pole models as a single light-pole class, showed a low level of performance when testing with the same VE dataset. Past work has shown that if the VE model is a sufficient representation of the RW analogue, then detection of the RW analogue is possible using a VE-trained detection model, due to a reduction in the domain shift (Reed et al., 2018). Our initial findings seem to indicate that all our models, some fraction of the model, our class structure, or our training methods have a fault. To pinpoint and address the issue, we generated additional datasets from the four light-pole classes to perform the regression analysis.

Table 3. The Average Precision (AP) and Average Recall (AR) results for trained models. Real-World is abbreviated by RW.

Dataset	Class	AP	AR
Black Light-Pole	black_y_light_pole	0.999	0.192
	black_light_pole	0.995	0.315
Black Y Light-Pole	black_y_light_pole	0.978	0.536
White Light-Pole	white_y_light_pole	0.708	0.261
	white_light_pole	0.998	0.333
White Y Light-Pole	white_y_light_pole	0.891	0.726
All Light-Pole	white_y_light_pole	0.352	0.135
	white_light_pole	0.978	0.110
	black_y_light_pole	0.998	0.115
	black_light_pole	0.996	0.225
New White Light-Pole	white_light_pole	0.710624	0.541283
New White Light-Pole (RW)	white_light_pole	0.837184	0.016116

Table 4. The mean Average Precision (mAP) and mean Average Recall (mAR) results for trained models.

Dataset	mAP	mAR
Black Light-Pole	0.997	0.254
White Light-Pole	0.853	0.297
All Light-Pole	0.831	0.146

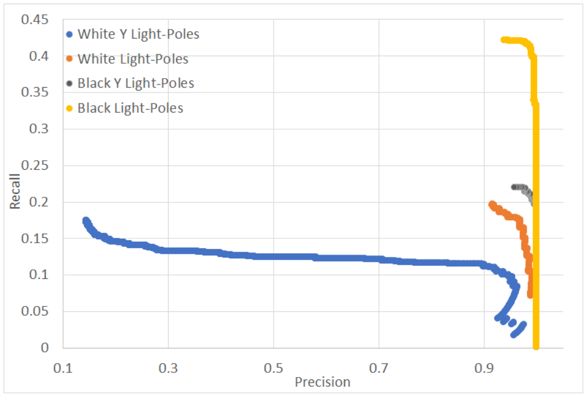

Figure 5. The precision-recall curve for the all light-pole dataset

After the regression analysis completed, we saw the trained models detecting with moderate recall success within the same dataset's test subset. This indicates that multiclass models, such as the one created from the all light-pole datasets, provide the necessary class structure for object detection with any level above 0% confidence. This level of confidence is in comparison to our initial single-class model; the latter identified black light-poles, white light-poles, and power poles as a single class. Our hypothesis for the discrepancy between the multiclass and single-class model result is that the single light-pole class is too diverse to provide reasonable detection confidence.

Notably, the white y light-pole model had a lower level of detection when paired with any other model, as shown in Table 3 through the all light-pole dataset and white light-pole dataset. This level of detection does not seem to be due to difficulty in detecting the 3D model because the white y light-pole dataset model showed comparable performance to other trained classes. The issue seems to be a competing effect against the white y light-pole and white light-pole class due to the 3D model similarities. When the models were tested on the RW dataset's test subset, every model failed to provide any detection. This indicates that the low level of performance of the single-class model was not due to a specific 3D model; but rather, the domain gap for every 3D model and their RW counterpart is quite large. In order to validate this cause, a handcrafted 3D model for the white light-pole was created with far more detail to show that 3D models were the contributing factor to the low performance. The result of this effort was the new white light-pole dataset and trained model.

The results of the new light-pole dataset, as shown in the row of Table 3 for the new white light-pole dataset, show a reasonable level of performance when testing within the same VE domain. This shows that the model was able to learn and detect the class. The testing on the RW dataset shows few detections, in the form of 0.016116. Although this is very low, this does show that the prior datasets' (i.e., the black, black y, white, white y, and all light-pole datasets) results were, in fact, due to a large domain gap.

Figure 6 shows a subjective comparison between the new white light-pole dataset, and it's RW analogue. We can see that the top row is fairly similar, but features such as object colour, shadow colour, and the indistinctness of the edges are visible. Because we created the 3D model based on a few examples of the light pole, and because the 3D model was placed for other light-pole locations, the model may not match as closely as the reference locations. The bottom row of Figure 6 shows the 3D light-pole model's arm is not exactly matching the length of the RW analogue. The bottom row also shows a few errors in rendering the shadow of the 3D model; although this should not affect the detection, due to the tightness of the bounding box, it does show that there may be other contributing factors to the domain gap.

5. Conclusion

The experiment has explored the effects of using hand-modelled 3D models to train the YOLO object detection. We have shown the impact of class organisational structure and the domain gap between the 3D models and RW analogue. We can conclude that using high-fidelity 3D models is critical to provide a robust RW detection system. We have also shown that handcrafted 3D-modelling can provide adequate training data for use in such a system, but there are many challenges that need to be considered and addressed in order to reach a high level of recall. The challenges include the quality of the 3D models, the volume of diverse samples needed, and the class organisational structure. Running with a larger sample size will likely improve results. The methodology for creating VE models in prior work involved using photogrammetry and LiDAR-based systems (Reed et al., 2018).

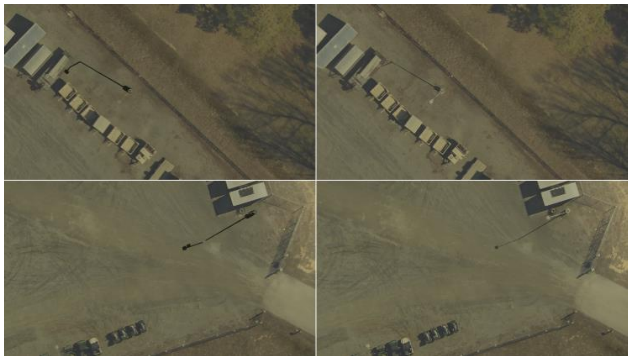

Figure 6. Examples of the new white light-pole dataset (left) and Real World (RW) dataset (right)

Those techniques provided a great level of individual detail to match the RW analogue object (i.e., photographs), were able to reduce the domain gap, and consequently, increase the reliability of detections. The level of model fidelity they previously re-created contrasts with our current method of hand modelling, which was a loose human interpretation. Future work can move to improve hand-modelling techniques by using advanced data-augmentation or model-capture techniques, such as photogrammetry and 3D scanning, to meet the fidelity requirements in order to train deep learning models for RW use cases using virtually generated data. A contrast of various techniques is shown in Table 5; the AR result in Table 5 for Reed et al. (2018) is an estimate supported by a graph in the paper.

Table 5. Techniques for modelling and results (at 2 digits of precision) of a trained model when tested on a real-world analogue for Average Precision (AP) and Average Recall (AR).

Technique (Object and Environment)	Object	AP	AR	Paper
Photogrammetry and 3D scanning	Helmet	0.96	0.83	(Reed et al., 2018)
Hand modelling and aerial imagery	Light pole	0.84	0.02	Current paper

Funding

This research was funded by the U.S. Army Futures Command, CCDC-SC, STTC, grant number W911NF-15-2-0099.

Acknowledgements

This research was sponsored by the U.S. Army Futures Command, Combat Capabilities Development Command-Soldier Center Simulation and Training Technologies Center (STTC). However, the views, findings, and conclusions contained in this presentation are solely those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

References

Aldrich, C. (2009). The complete guide to simulations and serious games: How the most valuable content will be created in the age beyond Gutenberg to Google. John Wiley & Sons.

Harrington, M.C. (2010). Empirical evidence of priming, transfer, reinforcement, and learning in the real and virtual trillium trails. IEEE Transactions on Learning Technologies, 4(2), 175-186. doi:10.1109/TLT.2010.20

Johnson-Roberson, M., Barto, C., Mehta, R., Sridhar, S.N., Rosaen, K., & Vasudevan, R. (2017). Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks? Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA), 746-753. doi:10.1109/ICRA.2017.7989092

Khan, S. (2019). Towards synthetic dataset generation for semantic segmentation networks (Unpublished master's thesis). University of Waterloo, Canada. Retrieved from https://uwspace.uwaterloo.ca/handle/10012/15128

Reed, D., Thomas, T., Eifert, L., Reynolds, S., Hurter, J., & Tucker F. (2018). Leveraging virtual environments to train a deep learning algorithm. Proceedings of the 17th International Conference on Modeling and Applied Simulation (MAS 2018), 48-54. Retrieved from http://www.msc-les.org/proceedings/mas/index.html

Reed, D., Thomas, T., Reynolds, S., Hurter, J., & Eifert, L. (2019). Deep learning of virtual-based aerial images: Increasing the fidelity of serious games for live training. Proceedings of the International Defence and Homeland Security Simulation Workshop 2019, 1-9. Retrieved from http://www.msc-les.org/proceedings/dhss/2019/DHSS2019.pdf

Ros, G., Sellart, L., Materzynska, J., Vazquez, D., & Lopez, A.M. (2016). The SYNTHIA dataset: A large collection of synthetic images for semantic segmentation of urban scenes. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 3234-3243. doi:10.1109/CVPR.2016.352

Sermet, Y., & Demir, I. (2018). Flood Action VR: A virtual reality framework for disaster awareness and emergency response training. Proceedings of the 2018 International Conference on Modeling, Simulation and Visualisation Methods (MSV'18), 65-68. Retrieved from https://csce.ucmss.com/cr/books/2018/Conference Report?ConferenceKey=MSV

Terrain Settings (n.d.). Retrieved July 1, 2020, from docs.unity3d.com/Manual/terrain-OtherSettings.html

Tian, F., & Lou, L. (2018). Dynamic scheduling of terrain based on Unity. Proceedings of the 2018 International Conference on Network, Communication, Computer Engineering (NCCE

2018), 1101-1104. https://doi.org/10.2991/ncce-18.2018.186

Tian, Y., Li, X., Wang, K., & Wang, F.Y. (2018). Training and testing object detectors with virtual images. IEEE/CAA Journal of Automatica Sinica, 5(2), 539-546. doi:10.1109/JAS.2017.7510841