CREATING SIMULATED MATERIALS FROM LIDAR DATA USING AI FOR PARAMETER OPTIMIZATION

Dipl.-Inf. Gregor Jochmann[(1)], Prof. Dr.-Ing. Jürgen Roßmann[(2)]

(1) RIF e.V, Joseph-von-Fraunhofer-Str. 20, 44227 Dortmund, Germany, gregor.jochmann@rt.rif-ev.de

(2) MMI, RWTH Aachen University, Ahornstraße 55, 52074 Aachen, rossmann@mmi.rwth-aachen.de

1. ABSTRACT

Especially in spaceflight and robotic exploration applications, virtual environments are essential tools in the planning and design stages of technical components and entire missions. To be able to produce realistic data from simulated optical sensors, these virtual testbeds do not only require accurate digital twins of the sensors themselves. The virtual testbed also needs to contain accurate representations of the objects in the sensor's environment, including their shape and surface material properties. In this paper, we describe our process of creating digital twins of real-world materials from 3D LiDAR reference measurements. We use NVIDIA Material Description Language implement the materials for our virtual robotics and sensors testbed VEROSIM. MDL allows us to accurately model the reflective and transmissive characteristics of material surfaces. Then, we explore parameter optimization with a reinforcement learning agent architecture using the TensorFlow Agents library.

2. INTRODUCTION

In our current research project ViTOS-3, we use our virtual robotics and sensor testbed VEROSIM to support the development and validation of spaceflight-capable LiDAR sensors. In this publication, we analyze creation of digital twins for a set of materials used in spaceflight applications like orbital rendezvous and robotic exploration missions.

Measuring, analyzing, and modelling material surface properties with respect to LiDAR sensors is an active topic of research with a wide range of applications like sensor simulation and algorithmic environment perception used for autonomous robots and driver assistance functions. Reference [1] analyzes the influence of materials on the accuracy of LiDAR measurements. References [2] and [3] demonstrate successful segmentation of LiDAR point cloud data into different material classes. In [4], the authors measure angle-dependent reflectance values of different materials relevant for automotive applications with a time-of-flight camera using modulated light at a wavelength of 945nm. They compare their measurements to the NASA ECOSTRESS spectral library [5] [6] and discuss how their measurements can be used for LiDAR models of other wavelengths for some material classes (concrete, asphalt, rubber, rock), but not for organic material (wood and other vegetation), where the reflectance drastically changes depending on the wavelength.

As can be seen in [4], multiple measurements of the same material class can result in vastly differing reflectance curves with reflectance values varying up to factors of 4 for some materials. The authors use gaussian distributions to describe the characteristics of the material class. In our case, we used a 3D LiDAR sensor to measure the reflectance values over small patches of material samples. We used a selection of 11 material samples including paints, metals, and specialized spaceflight materials (see Figure 1). This reference experiment is discussed in subsection 5.1. In subsection 5.2, we then create a model for simulated materials exhibiting the measured characteristics when scanned by a digital twin of the LiDAR. Scanning two-dimensional patches instead of a single, centered measurement point on the material captures the characteristics in reflectance variance better and consequently enables us to build a better model for the surface material. Subsection 5.3 discusses selection of the best parameters to use for the material, including an attempt at training a reinforcement learning agent for the task.

We use NVIDIA Material Description Language (MDL, [7]) to implement the model of the material. The Material Description Language specification provides a range of bidirectional scattering distribution functions (BSDF) for the definition of material surface properties for use in rendering and sensor simulation applications. The three main BSDFs are used to specify diffuse, specular, and glossy light interaction. Each of these have parameters to control reflection tint (color), reflection factor, and roughness. Additionally, the language defines modifiers and combiners to implement more complex light interactions like layered materials or Fresnel reflection.

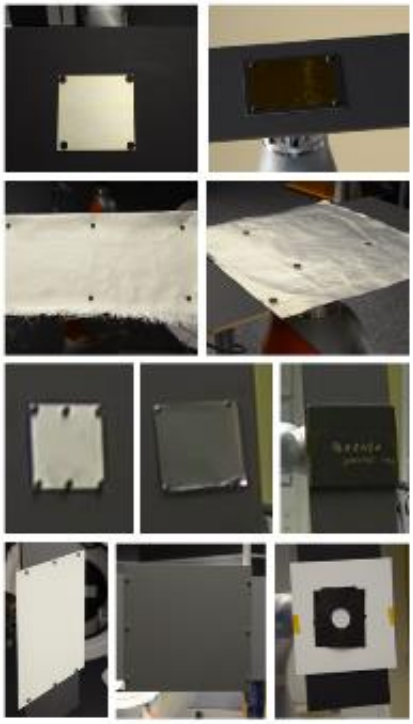

Figure 1: materials scanned in the reference experiment

3. SIMULATION ENVIRONMENT

The simulation environment used to compare the simulated materials with the reference experiment is our virtual testbed VEROSIM [12]. The software has modules for kinematics, rigid-body-dynamics, and the simulation of optical sensors (raytracing and rasterization-based). The testbed also includes a collection of processing, analysis, and data I/O tools, as well as a python interpreter. VEROSIM is centered around the concept of digital twins [13], giving users the option to have comprehensive digital representations of systems and objects, including their interfaces, properties, and functionality.

In the MDL creation phase, we use the virtual testbed to simulate and measure the parametrized base material that is used in each concrete material. For the evaluation of the simulated materials, we recreated the reference experiment in the virtual testbed, and used a digital twin of the LiDAR to scan the MDL materials (see Figure 2). We used raytracing LiDAR simulation [14] to generate the simulated point clouds and intensity measurements and exported the data for analysis and comparison with Python scripts.

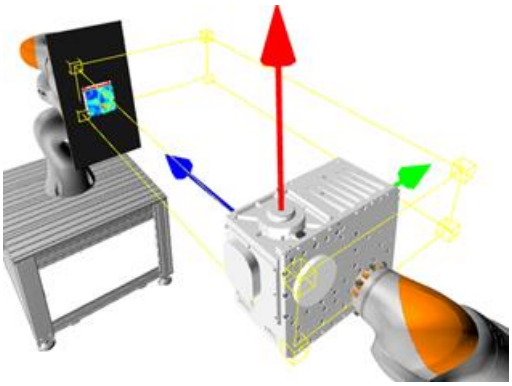

Figure 2: Digital twins of LiDAR, robots, and material sample in the virtual testbed

4. METHODOLOGY

4.1 Reference measurements

To create the reference measurements, a spaceflight-capable LiDAR scanned eleven differed material samples under 90 different viewing angles from 0° to 89° from normal angle (i.e., normal to shallow) at a distance of one meter. Details of the experiment setup are discussed in [8]. The LiDAR produced a 3D point cloud that included reflection intensity for each datapoint. The intensity was recorded in raw sensor readings ("digits", d), which (according to manufacturer specifications) correspond to received power P following equation (1):

$$P(d) = 1.0042e^{-6} * 1.00422^{d} \qquad (1)$$

For each material, we defined a quadratic region of interest (ROI) that was free of mounting utilities (see Figure 3). The edge lengths of the region ranged from 3cm to 8cm. With a spatial resolution of the LiDAR of 4mm, this resulted in hit counts in the normal view of 7 x 7 for the smallest regions to 20 x 20 for the largest regions.

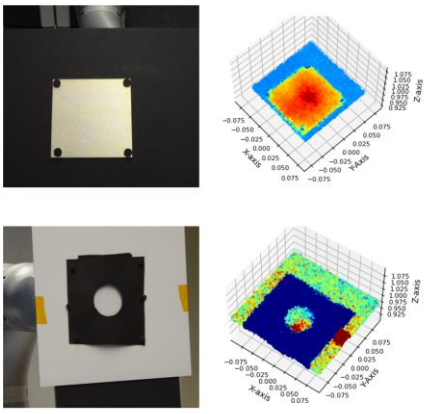

Figure 3: Unfiltered measurements with mounting utilities visible in the point clouds.

We chose an intermediate data representation we call *intensity grid* to define the simulated materials and compare simulated with measured datasets. The intensity grid is calculated from the 3D point cloud and corresponding intensity values for each of the 90 angle-steps. The region of interest is subdivided in evenly spaced intervals. At each grid position p_{GRID}, we collect the nearby measurement points and calculate a weighted average of their intensities $\mathrm{I}(p_{GRID})$, with weight depending on their distance to the grid point as shown in equations (2) and (3):

$$\mathrm{I}(p_{GRID}) = \frac{\sum_{i=0}^{n} w_i I_i}{\sum_{i=0}^{n} w_i} \tag{2}$$

$$w_i = \begin{cases} \frac{1}{|p_i, p_{\mathrm{GRID}}|^4}, & \text{if } p_{GRID}[0] - p_i[0] < t \text{ and} \\ & p_{GRID}[1] - p_i[1] < t \\ 0, & \text{else} \end{cases} \tag{3}$$

with p the positions in local sample coordinates, and t the distance threshold, selected as distance between grid positions. We transform the point cloud into the sample reference frame to make the calculation independent from range errors of the LiDAR sensor and inaccuracies from the manual measurements of sample holder and material thickness.

Combining all 90 intensity grids of a material lets us define the absolute reflectance intensity curve for each position on the intensity grid. To do so, we calculate the incidence angle for each grid position as angle between the normal vector of the sample holder and the connection between the grid position and the LiDAR optical center as shown in Figure 4.

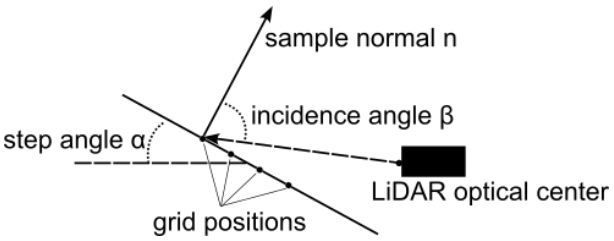

Figure 4: Calculation of incidence angle for the grid positions

Scanned at shallow angles, the reflectivity of most of the selected materials was so low that the LiDAR receiver failed to register returns for some or even most of the emitted pulses (see Figure 5). The angle at which the returns started to vanish is highly dependent on the material. For most materials, the incidence angle at which we could observe missing returns was around 80°, but for some materials, this started as early as 30° ("honeycomb") or 50° (kapton netting, silica black, black paint). Missing returns lead to zero-values in the intensity grids, which required special handling during evaluation (see below).

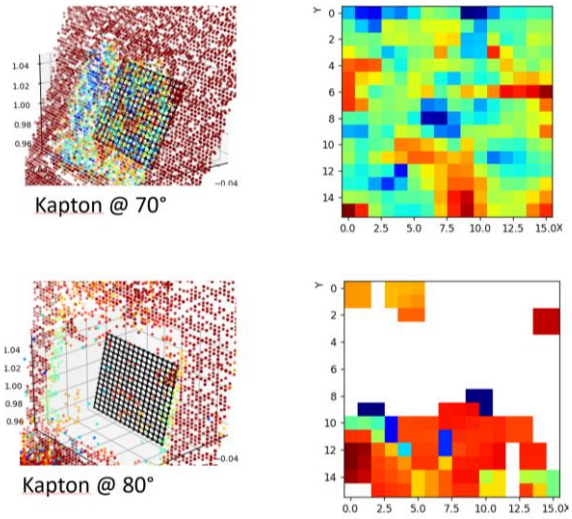

Figure 5: Kapton scanned at incidence angles 70° and 80°. Left: Point cloud and grid locations, right: color-coded intensity grid values. White pixels denote missing values due to insufficient return intensity.

4.2 MDL material

We define an MDL material for each scanned reference material. We use a basic (diffuse or glossy) BSDF, and the *df::measured_factor* modifier to multiply the intensity at different positions on the surface with angle-dependent factors so that they match the absolute reflectance intensity values from the reference material.

To control the reflectivity at different positions of the simulated object, we use the texture coordinates of its geometry. We define a regular grid with the same resolution as our intensity grid over the texture coordinates and create a mapping from texture coordinates to the corresponding grid position. We use *df::normalized_mix* with weights based on this mapping to select which reflectance curve is used at different locations on the geometry as shown in Listing 1.

```
material astroQuartz (int resolution, bool base_diffuse, float base_value, float base_rough)
= let
{
   bsdf base_bsdf = base_material(base_diffuse, base_value, base_rough)
                                                .surface.scattering;
} in material (
   surface: material_surface(
      scattering: df::normalized_mix(components: df::bsdf_component[](
         //note: weight_for_grid() returns
         //   1.0 for UV coordinates matching the grid position, 0.0 else
         df::bsdf_component(
            weight: weight_for_grid(x:0, y:0, grid_resolution: resolution),
            component: df::measured_factor(
               values:texture_2d("./astroQuartz/x0_y0.bmp", tex::gamma_linear),
               base: base_bsdf)),
         df::bsdf_component(
            weight: weight_for_grid(x:1, y:0, grid_resolution: resolution),
            component: df::measured_factor(
               values:texture_2d("./astroQuartz/x1_y0.bmp", tex::gamma_linear),
               base: base_bsdf)),
         ...
```

Listing 1: MDL code structure of our material. BSDF selection based on texture coordinates (yellow). Each component (purple) stores the reflectance curve factors (blue) for a single grid position.

The *df::measured_factor* modifier multiplies an existing BSDF (specified as parameter *base*) with angle-dependent factors. The factors are encoded in a grayscale image with a resolution of one pixel per degree. Since not all grid positions are scanned under every viewing angle, we interpolate missing values with the Navier-Stokes method (using the *inpaint* function of the OpenCV library [11]) and extrapolate missing values at the beginning and end using linearization.

To calculate the multiplication factors relative to the reflection intensity of the base material, we built an environment in our testbed in which a simulated LiDAR scans a virtual sample using the base BSDF. The MDL generator module calculates two intensity grids: one from the output of the simulated LiDAR scanning the base material, and one from the replay data for the current reference material. By dividing the absolute intensity values from both grids at every grid position, the module calculates the relative value required for the *df::measured_factor* modifier function.

Initially, we chose an ideal diffuse white BSDF as base for each material. Tests quickly revealed that using individually parametrized base materials yielded better results. This is caused by the type of encoding of the modification factors - the modification factors can only be encoded as an 8-bit grayscale image and cannot exceed a factor of 1.0. This leads to resolution issues and impossible-to-represent modification factors greater than 1.0 for some materials and/or viewing angles. Selection of suitable base material parameters is discussed in the next section.

4.3 Parameter optimization

Instead of manually choosing the three free parameters of the base material (*base_roughness* (float), *base_value* (float) and *base_is_glossy* (bool)) for each material sample, we trained a reinforcement learning agent to optimize these parameters. By doing so, we wanted the agent to learn a model that could generalize to selecting good parameters for unknown/future measurements of different materials without having to search the whole parameter space for every material. In typical reinforcement learning setups, the agents take multiple actions sequentially in a dynamic environment to influence their state and the environment to optimize a reward function. In our case, the state consists of the intensity grids of both sweeps (the reference material and the simulated material with the current parameters), and the action is to set all three base parameters to a value between 0 and 1 (note that for the *base_is_glossy* parameter, we convert the floating-point of the action to a boolean by rounding to 0 or 1). After each action, the simulation is re-run to produce a new MDL material variant and to scan it with the simulated LIDAR. Each episode ends after a fixed number of actions (n=10, chosen arbitrarily as the system state in our case only depends on the most recent action taken and not on the history of previous actions). After each episode, we select the next reference material, randomly select initial base material parameters, and continue the training.

The reward is calculated from comparison between the intensity grids of the reference data and the output from the simulated LiDAR. We took the absolute and relative intensity differences at each grid position to calculate mean absolute error, mean relative error and root-mean-square error over the entire 90-degree sweep. The inverse of the average root-mean-square error is used as reward function, the other metrics are used for manual analysis and evaluation purposes as shown in Figure 6. Grid positions where the reference intensity grid has no data are ignored for all metrics. Once the relative amount of missing data points passes 40%, we ignore all further measurements as the LiDAR data is unreliable (this is the "step limit" from Figure 6).

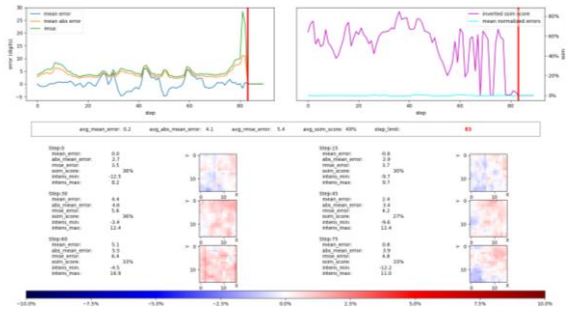

Figure 6: Comparison between reference measurements and simulated material

For the agent, we chose the twin-delayed deep deterministic policy gradient algorithm (TD3 [10]), an evolution of the deep deterministic gradient policy agent (DDPG [9]). This choice was based on our previous good experiences with the algorithm and the fact that it supports continuous action spaces. Our implementation uses the TensorFlow Agents framework [15]. We did two separate training runs. Because the simulation of the MDL materials with high-resolution intensity grids (16x16) was taking so long, we started with a low-resolution setting (2x2) to see if the approach and parameters were useful at all. With the low-resolution setting, each training step took about one minute, resulting in an episode time of about 10 minutes. The shape of the neural networks and the hyperparameters used can be seen in Table 1.

Table 1: Hyperparameters for TD3 agent training

Training set name	TD3 #1	TD3 #2
Actor network layers (fully connected)	(250, 250, 250)	(64, 64)
Critic network layers [observation, action, joint] (fully connected)	(250)	(64, 64)
Learning rates (actor, critic)	0.001, 0.0001	
Discount factor γ	0.9	
Epsilon (without decay)	0.3	
Target update τ	0.01	
Exploration noise	0.2	
Target policy noise	0.2	
Initial collect steps	200	
Replay buffer size	1000	
Batch size	512	

Unfortunately, neither of the training runs produced a model that performed as well as expected (see Figure 7). Despite seeing good parameters in the initial random guessing phase that fills the replay buffer, the agent quickly converges on parameters that are not optimal, and sometimes worse than the initial random parameters at the beginning of an episode. Further training did not improve the performance. In a third training run, the agent converged to even worse parameters with choosing a glossy base material, despite using a diffuse base would have given better rewards independently of the other two factors.

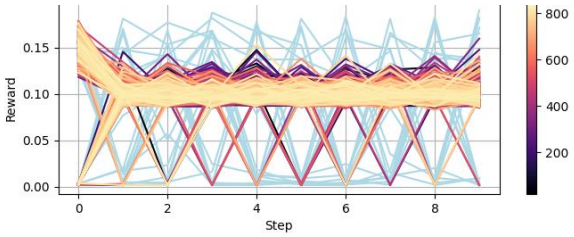

Figure 7: Training of a TD3 agent. Each line is a training episode. After the initial random-guessing phase to fill the replay buffer (light blue), the actions taken by the agent are often worse than the initial random guess (step 0). An increase in performance with increasing episode number is not observable.

We could not determine the cause for this unexpected behavior. Just adding more training episodes did not help – the agent just seemed to be stuck, not fully exploring the parameter space (see Figure 8). We used standard parameter values and recommendations from literature.

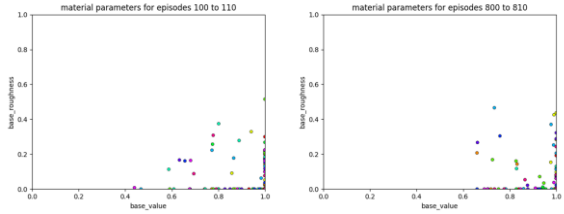

Figure 8: Material parameters chosen by the agent during training episodes 100 to 110 and 800 to 810 for a single material (astroQuartz). The agent is no longer exploring the full parameter space and oscillates around suboptimal values. Note: The agent chose a diffuse base material, so that we could plot the remaining parameters in 2D.

As a comparison and to get useful materials for our simulation, we fell back to an analytic grid-search strategy. We sampled the parameter search-space in small regular intervals and simulated every combination of parameter values. The optimal values produced by this search resulted in materials which accurately reproduce the intensity grids from the reference material, with less than 2% relative per-position error across all steps and grid positions – see Figure 6.

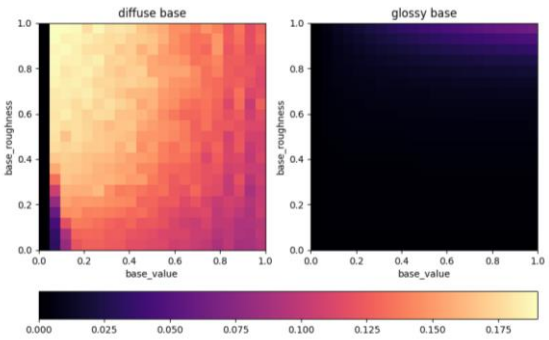

Figure 9: Reward maps produced by grid-search for material astroQuartz using diffuse and glossy base.

The reward plot using the diffuse base (Figure 9, left) shows a distinctive gradient towards maximum reward, except for the sharp falloff at *base_value* zero (in which case the simulated material no longer reflects anything, leading to a reward of zero). Comparing Figure 8 and Figure 9, we can see that the agent is clearly not following the gradient towards higher reward values. After double-checking that we had not made errors like feeding negative reward to the agent or having swapped the order/role of the action parameters, we still have no explanation for the failure of the training. We do not think that the sharp reward falloff from low to zero values of the *base_value* parameter sufficiently explains the observed behavior.

To make sure our implementation is correct, we used the same framework and hyperparameters to train an agent on a scenario that resembled the properties of our application problem while having a reduced number of input and output parameters and complexity. In this scenario, a single floating-point value between zero and 100.0 replaces each of the two sets of intensity grids (reference and simulation result), and the action space is also a single value with the same bounds. The reward function was chosen as the inverse of the difference between reference value and action value (same as in the original application). The reference value was chosen randomly for each episode. The agent could achieve a maximum reward by choosing the action value that was indicated in the top half of its environment/observation vector and repeat the action until the episode is over. The training of this scenario succeeded; we could observe how the agent quickly converged to choosing action values close to the reference value after a few hundred episodes of training. We therefore feel confident that the failure mode observed in the actual use case is caused by some underlying properties of the problem and/or poor choice of hyperparameters.

5. CONCLUSION AND FUTURE WORK

We demonstrated the use of the Material Description Language to describe surface properties independent from a specific programming language or rendering algorithm. The grid-based design of our custom MDL materials allows us to control surface reflectivity at each position of the grid. With this technique, we created accurate digital twins for heterogeneous real-world materials. We also introduced our concept of intensity grids as a two-dimensional representation of a region-of-interest from three-dimensional point-cloud data with intensity measurements.

We did not succeed in training a reinforcement learning agent to optimize the parameters of the materials, and used a grid-search across all parameter combinations to find good parameter sets instead. More research is needed to determine why the RL training did not follow the clearly existing gradients towards higher rewards, and instead converged to selecting parameters at the opposite end of the gradients in the action-space. Discussions about DDPG failure modes can be found in [16] and [17], but the root cause of the failure mode discussed in [16] (sparse reward) does not apply to our training environment. The *extrapolation error* discussed in [17] might be a possible cause, but the publication does not mention our failure mode of the training getting stuck at the wrong end of the action-space. The authors of [17] propose using a different algorithm (Batch-Constrained deep Q-learning, BCQ).

While our use case with a system state exclusively depending on the most recent action (independent of previous actions or its current state) is unusual for the application of a reinforcement learning strategy. But there is nothing in the literature to indicate that this property is required for successful training. To verify this, we implemented a scenario in which the agent training succeeded. It remains to be seen if any of the following changes leads to a successful agent model:

- Choosing different hyperparameters: DDPG (the predecessor of the TD3-algorithm we used) is sometimes reported to be sensitive to hyperparameters.
- Change of different network size/layout: Choosing larger network layer sizes and more layers provides greater representational capabilities for the network. However, this also introduces some downsides, as larger networks can overfit the training data, and/or can suffer from gradient instability.
- Not including the simulated intensity grids into the observation state vector. This halves the size

of the observation vector and makes the agent completely unable to influence the state (since it only contains the reference data). Theoretically, the optimal material parameters exclusively depend on the reference values, so this could speed up the learning process.

- Selecting a different RL agent algorithm: Off-policy algorithms like the TD3 algorithm we used decouple data collection and policy improvement. While off-policy-algorithms are typically more sample-efficient (i.e., requiring fewer training episodes to converge) compared to on-policy algorithms, they are more challenging to stabilize. A comparison with a slower, but more robust on-policy algorithm like proximal policy optimization (PPO) might be required in our use case. By discretizing the action space like we did for the grid-search approach, it is possible to deploy an even wider range of algorithms to the problem.

As an alternative to training deep learning agents, the field of classical derivative-free optimization (also known as blackbox optimization) offers a different set of algorithms to quickly find approximately optimal parameters. These algorithms are suitable to problems where the function is noisy and/or time-consuming to evaluate, and its derivative is unavailable or impractical to obtain. Genetic algorithms, simulated annealing, Powell's method, or Nelder-Mead optimization promise efficient localization of good solution, but optimality guarantees can not be given.

6. ACKNOWLEDGMENT

This work is part of the project "ViTOS-3", supported by the German Aerospace Center (DLR) with funds of the German Federal Ministry of Economics and Technology (BMWi) under support code 50 RA 2121.

7. REFERENCES

[1] Voegtle, T., Schwab, I., Landes, T. (2008). Influences of different materials on the measurement of a Terrestrial Laser Scanner (TLS). *Proc. of the XXI Congress, the International Society for Photogrammetry and Remote Sensing*, ISPRS2008. p. 37.

[2] Song, J.-h., Han, S.-h., Yu, K., Kim, Y.-i. (2012). Assessing the possibility of land-cover classification using lidar intensity data, *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 34.

[3] Yuan, L., Guo, J. and Wang, Q. (2020). "Automatic classification of common building materials from 3d terrestrial laser scan data. *Automation in Construction*, vol. 110, p. 103017, 2020

[4] Muckenhuber S, Holzer H, Bockaj Z. (2020). Automotive Lidar Modelling Approach Based on Material Properties and Lidar Capabilities. *Sensors*. Basel 2020;20(11):3309. doi:10.3390/s20113309

[5] Jet Propulsion Laboratory, California Institute of Technology. (2020). ECOSTRESS Spectral Library Version 1.0. Available online: https://speclib.jpl.nasa.gov (accessed on 13 September 2023).

[6] Meerdink, S. K., Hook, S. J., Roberts, D. A., Abbott, E. A. (2019). The ECOSTRESS spectral library version 1.0. *Remote Sens. Environ*. 2019, 230, 1–8.

[7] NVIDIA (2019) - MDL Language Specification, Available online: https://developer.nvidia.com/designworks/dl/mdl-1_6-spec (accessed on 13 September 2023)

[8] Dahmen, U., Priggemeyer, M., & Roßmann, J. (2021). Cyber-Physical Systems and Digital Twins in Practice – A Real-Life Application Example. 8th IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE 2021). Brisbane, Australia.

[9] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. & Wierstra, D. (2016). Continuous control with deep reinforcement learning. *ICLR*.

[10] Fujimoto, S., van Hoof, H. & Meger, D. (2018). Addressing function approximation error in actor-critic methods. arXiv:1802.09477.

[11] Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

[12] Emde, M., Priggemeyer, M., Steil, T., Grinshpun, G., Rossmann, J. (2016). A Virtual Space Robotics Testbed for Optical Sensors in Aerospace, *Proceedings of the 47th International Symposium on Robotics - Robotics in the era of digitalisation* (ISR 2016) VDE Verlag.

[13] Trauer, J., Schweigert-Recksiek, S., Engel, C., Spreitzer, K., Zimmermann, M. (2020). What is a digital twin? – Definitions and insights from an industrial case study in technical product development. *Proceedings of the Design Society: DESIGN Conference*, 1, 757-766.

[14] Thieling, J. and Roßmann, J. (2021). Physical Sensor Simulation for the Verification and Validation of Optical Systems, ASIM 2021.

[15] The TF-Agents authors (2023) TensorFlow Agents. Available online: https://www.tensorflow.org/agents (accessed on 13 September 2023)

[16] Matheron, G., Perrin, N, Sigaudt, O. (2020). Understanding Failures of Deterministic Actor-Critic with Continuous Action Spaces and Sparse Rewards. *Artificial Neural Networks and Machine Learning (ICANN) 2020.* Springer International Publishing. p 308-320.

[17] Fujimoto, S., Meger, D., Precup, D. (2019) Off-Policy Deep Reinforcement Learning without Exploration. arXiv:1812.02900 (2018)