A Framework for Self-Training Perceptual Agents in Simulated Photorealistic Environments

Patrick Mania*
pmania@cs.uni-bremen.de

Michael Beetz*
beetz@cs.uni-bremen.de

***Abstract*— The development of high-performance perception for mobile robotic agents is still challenging. Learning appropriate perception models usually requires extensive amounts of labeled training data that ideally follows the same distribution as the data an agent will encounter in its target task. Recent developments in gaming industry led to game engines able to generate photorealistic environments in real-time, which can be used to realistically simulate the sensory input of an agent.**

We propose a novel framework which allows the definition of different learning scenarios and instantiates these scenarios in a high quality game engine where a perceptual agent can act and learn in. The scenarios are specified in a newly developed scenario description language that allows the parametrization of the virtual environment and the perceptual agent. New scenarios can be sampled from a task-specific object distribution that allows the automatic generation of extensive amounts of different learning environments for the perceptual agent.

We will demonstrate the plausibility of the framework by conducting object recognition experiments on a real robotic system which has been trained within our framework.

***Index Terms*— Self-Training Perception, Robotic Simulation, Unreal Engine, Scenario Description**

I. INTRODUCTION

In recent years computer vision has made impressive performance jumps in terms of recall[1] (the probability that they do not overlook objects in images) and precision[1] (the probability that the detections are correct). Much of this progress is achieved through intensive use of machine learning technology that trains perception capabilities for the respective perception tasks using huge training sets showing the original images and the results the algorithms are to return. In many of these cases the bottleneck is obtaining the huge data sets and in particular the results that should be returned for each image ([2],[3]).

For images made by humans such data can be accessed through the world wide web and websites such as FLICKr, wikimedia, etc. The images that are available for training are unfortunately untypical for robot vision applications. This is because people carefully direct their cameras at objects and scenes they want to capture in a way that those are easily detectable and clearly visible. In contrast, when a robot performs tasks in a kitchen, such as setting the table, many objects are occluded, seen from a bad angle, under inferior lighting conditions, partially depicted, and so on. For some applications, such as autonomous driving, such data sets can be generated by recording image streams and taking images in which objects can be detected well and tracking the objects backwards to generate training data for more typical image distributions[4]. However, these techniques cannot be easily transferred to other robot applications such as doing manipulation tasks in a household.

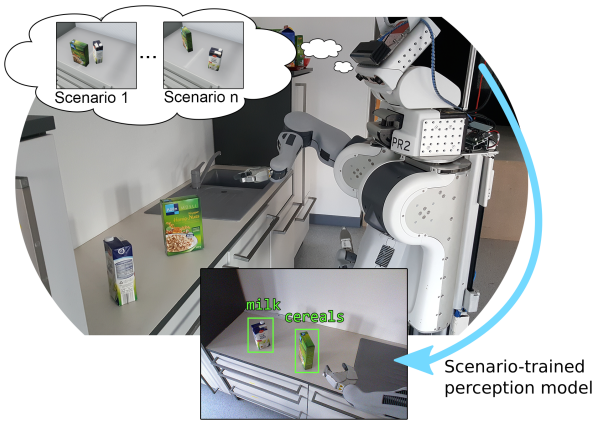

Fig. 1: With the proposed framework, it is possible to generate Scenario-based training data in game engines for learning perceptual agents.

A second recent development is the level of photorealism that can be achieved with modern game technology. For example, the game engine Unreal Engine 4(UE4) has powerful tools for realistic materials, including transparency, reflectance patterns and lighting, which make it possible to move a camera through a simulated environment and produce images[1] similar to those obtained in the real world([5],[6]).

Robot vision can make use of this powerful game engine technology to generate high quality training data for computer vision. The basic idea is that the game engine exactly knows where each object in the virtual environment is placed. Having this knowledge, one can calculate which parts of the incoming sensory data correspond to a given object, annotate it with additional information (for example: class labels, see Fig. 1) and store it in the set of training examples.

UnrealCV[6] provides a module that can be loaded into UE4 to generate RGB, depth and object mask images from cameras inside of a virtual environment. It has been used to generate synthetic data for different purposes, for example to learn an object detection model[7] or to generate optimal trajectories for UAVs in scanning tasks[8].

In this paper, we build on the UnrealCV infrastructure and propose a framework for self-training perceptual agents that

* The author is with the Collaborative Research Centre Everyday Activities Science and Engineering (EASE), University of Bremen, D-28359 Bremen, Germany, WWW: http://ease-crc.org/

[1]https://www.youtube.com/watch?v=E3LtFrMAvQ4

can generate distributions of training data that correspond to the distributions that robot control programs generate when performing realistic tasks in realistic environments. This method can be expected to boost the learning performance as many learning algorithms assume that the distribution of the images in the training data is the same as the one generated by the robot application later([9],[10]).

The framework consists of the following components that are at the same time the research contributions of this paper:

- A *Training Data Collector*: This system samples scenes in a virtual environment and executes fragments of the parameterized high-level robot plans simulating the respective perception-action loop of the robot control program. For each captured image it generates a tuple together with the ground truth data that are computed from the data structure of the environment as a training data entry for the learning task. In the experiment section, we will show that the generated training data can be used for an object recognition task on a real robot and compare it to real world training data.
- A *Scenario Description Language (SDL)*: We propose SDL, a semantic specification language that enables robot programmers to specify distributions of scenarios and behavior specifications for robotic agents in a transparent and modular way. A scenario consists of a normative scenario, which is then probabilistically modified by eliminating and adding objects, and changing the pose of objects. In addition, a parameterized high-level behavior for the robot can be specified where again the parameterization can be probabilistically modified.
- A *Semantic Scenario Sampler*: This sampler takes the probabilistic scenario description and converts it into a stochastic process that draws examples implied by the distribution defined by the SDL specification.

II. Motivation

Robotic agents perform perception tasks in order to accomplish their manipulation actions. Thus if a robotic agent executes a fetch and place task, it has to detect the object to be picked up and examine it in order to find the proper place to grasp and hold them. Depending on which task context the fetch and place program is called the same category of objects looks very different. Consider for example a household robot. The robot is to set the table, serving the meal, and cleaning the table afterwards. If the robot is to set the table or serve a meal, then it first has to find the plate in the cupboard. Plates will be stacked and therefore the task of the perception system is to find the topmost horizontal line in the cupboard. Typically, lighting conditions inside the cupboard are poor so the robot has to detect the plate in a dark setting. This is shown in Fig. 2 in the middle. When the robot is serving the meal then the plates typically appear as ovals in the image. In the case that the plates are textured the robot might want to detect the plate pattern in the image. The lighting conditions are typically bright and you might get reflections from sunlight (see Fig. 2 left). Finally, when the robot is to clean the table then the

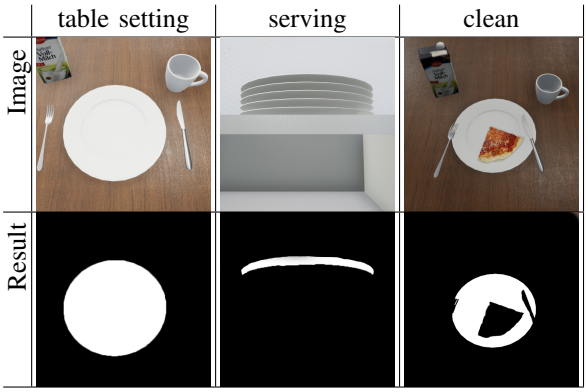

Fig. 2: The appearance of objects changes substantially, depending on the task context.

plates look like the ones for table setting but they might be dirty as shown in Fig. 2 on the right. Thus, Fig. 2 shows images captured by the robot in order to detect plates and how the appearance of plates changes depending on the task context and the desired output of the detection routine in the lower row. The figure suggests that if we learn different detection routines for each task context that these detection routines can be expected to be more robust, more accurate, and more efficient than a general plate detector. This process of optimizing perception by exploiting the regularities and constraints imposed by the task and environment context has been called environment and task specialization by Horswill and its potential of specialization was intensively discussed and investigated in [11].

III. System Architecture

Fig. 3 shows the software architecture of the proposed system and the essential data flow and computational processes. The system consists of three major components:

The *Scenario Runner* creates scenes according to a given scenario in the *Simulation Environment* and uses the *High-Level Agent Control* module to parametrize the agent that is to be improved, according to the Feedback of the *Critic*. The Simulation Environment is composed of a state of the art game engine and an *Interface Layer*, which implements efficient communication capabilities and the necessary logic to manipulate the virtual environment programmatically. After the scene and the agent have been instantiated in the Simulation Environment, the agent uses its High-Level control program and perception system to sense its environment, reason about its actions and solve the task at hand. Each percept can be stored in a *Percept Database* to log the experiences of the robot and provide a data basis to further evaluate and improve its given models.

Each situation that an agent should be confronted with will be modeled as a *Scenario* in our SDL. Each Scenario contains three major parts: The *Scenario Meta Data* is used to identify each Scenario and its different components uniquely. To decrease the synthetic nature of the sensor data, one can also define noise models in this section of the SDL. A description of the virtual environment where the agent

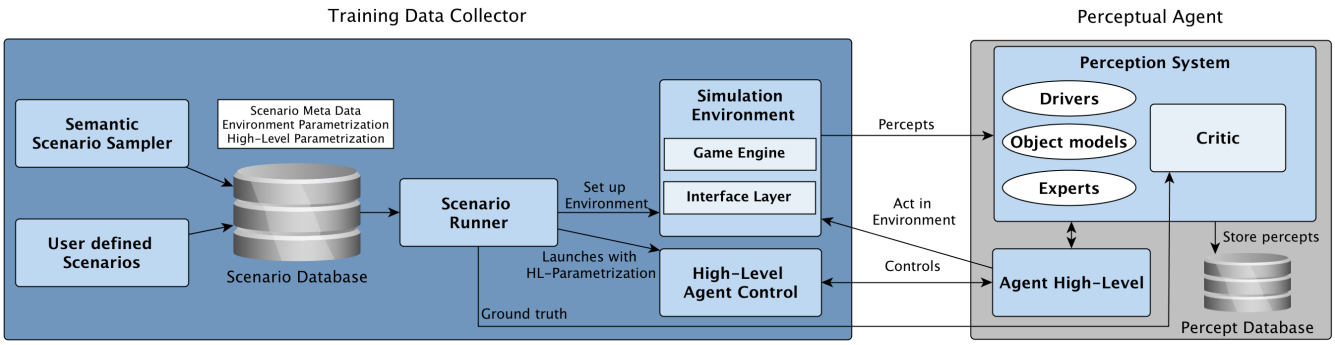

Fig. 3: Overview of the proposed system. Scenarios from different domains can be expressed in our SDL and stored in a specialized Database. These Scenarios can be set up in a simulation environment where a perceptual agent can act in. The agent perceives sensory data from the environment and uses its perception system to accomplish a task. A critic component uses ground truth from the Scenario description to feedback the current performance to the perception system.

should act in is defined in the *Environment Parametrization*. This includes for example the definition of entities like primitives or 3D-Models or different parametrizations for lighting which may effect vision algorithms. To support the study of different agent behaviors within this framework, the High-Level Control for the agent can also be parametrized on a Scenario-level. All Scenarios are collected in a *Scenario Database* and can be defined in different ways. One possibility is the creation of Scenarios by hand, as one might do it under lab conditions. The desired environment will be set up in the simulation by the given editing tools like Move, Rotate, etc. After everything is set up, the structure of the environment can be saved as a Scenario and stored into the Scenario Database.

To generate an extensive amount of Scenarios for mobile robotic agents that generate typical image distributions of the objects to manipulate, we propose the usage of probabilistic models. By employing these models and an initial distribution of the task-specific objects in a given scene, new object constellations in a relevant task context can be generated and used as a training input for the perceptual agent. The Scenario Runner can setup arbitrary amounts of these sampled Scenarios in the Simulation Environment, place and start the perceptual agent in it, wait for its completion and observe the achieved performance. Within this perception-action-feedback loop, the agent can improve his contextual perception routines based upon the received percepts of the sampled scenes and the feedback of the Critic.

A. Scenario Description Language

Before describing the individual components of the system, the developed SDL will be discussed briefly. When running computer vision tasks in the context of task-specific learning, very often one will need a) an experimental setup where you want to validate or train your system on, b) the necessary ground truth for it, c) instructions for the agent to cope with the task at hand and d) some meta data to identify the individual results on the different Scenarios. These four parts are described in our SDL, which is roughly of the following form:

(a scenario
  (identified-by (a date) (an #id)
  (an environment
    (an object (type class_1) (pose p_1) (distribution dist_1))
    (a light (pose p_2) (with light-parameters(...))))
  (an agent
    (task-description (a task(...)))
    (with startup-parameters
      (param-type_1 val_1)
      ...
      (param-type_n val_n))))

The Environment Parametrization contains all the necessary information to describe an environment where the agent should act in. Object descriptions are supplied for each object which should be spawned in the virtual environment and includes the spatial parameters, distribution properties, ground truth data and parameters to define the appearance in the simulation. To get realistic object renderings in a simulation, properly setting up light sources is a crucial part of the virtual environment design process and can therefore also be specified.

Supporting the initialization and parametrization of different agents is accomplished by the *High-Level Parametrization* part of the SDL, which allows the definition of the starting process of the agent and an object-action task description. This description is used to start the agent that is run by the Scenario Runner, which coordinates the execution of the agent and the setup and destruction of the virtual environment. The last part of the SDL is the Scenario Meta Data, which is used to index individual Scenarios, hold a reference to the Environment Parametrization & High-Level Parametrization and user-defined meta data which is necessary for the task at hand (for example sensor noise models or textual descriptions of a Scenario etc.).

The process of generating new Scenarios according to typical distributions of entities is implemented by the Semantic Scenario Sampler. First, we distinguish Scenarios into two types: *Normative* and *Concrete*. Both Scenarios are declared in the same SDL, but are necessary for two

different tasks: *Normative Scenarios* are used as a prior to model the abstract structure of the scene. In a table-setting example, this would include the potential position and amount of all task-relevant objects. Using this basic structure in addition to the *distribution* properties for every object in the Environment Parametrization, we can sample an infinite amount of different Scenarios to improve on in a stochastic process. Inside of this process, the sampler will decide if objects from the Normative Scenarios will be removed from the scene, modified in their amount or be placed in other areas. These new, generated Scenarios will be described in our SDL and are called *Concrete Scenarios*. This kind of Scenario will be instantiated in the Simulation Environment and be used to improve the perceptual capabilities of the agent.

B. Simulation Interfacing

The main component of the Simulation Environment in our proposed framework is Unreal Engine 4[12], which is a modern game engine capable of producing highly realistic renderings. UE4 uses a proprietary data format to describe a virtual environment in it. To set up the environment which has been defined in our SDL in this game engine, a communication layer was implemented that allows to modify the Simulation Environment during runtime. This allows our framework to setup different environments without needing time-consuming restarts of the Simulation Environment and therefore a higher throughput of Scenario executions. Our communication layer is based upon ROS (Robot Operating Systems), which is a widely used robotic framework in the scientific community. Each action like spawning or destructing objects or changing light settings can be accomplished using standard ROS communication methods. The central endpoint of this communication layer is the implemented Interface Layer inside of UE4. This layer handles the communication stack and provides methods to manipulate a given virtual environment. A central component is the SDL Spawning module, which processes the object creation requests. One can either spawn geometric primitives or 3D-Meshes. Primitives might be useful for robot belief state visualization or learning the outcome of physical processes, like in [13], while 3D-Meshes are indispensable when learning from image distributions in real life tasks with textured objects like household products. To record the spatial relation of objects during runtime, we implemented ROS tf2 [14] in UE4 to analyze the movement of objects during the execution of the agent or to record a given setup as a Scenario.

We are publishing[2] our interface between UE4 and ROS, which is necessary to manipulate virtual environments programmatically and allows other researchers to do their own experiments in this game engine.

C. Optimizing Perceptual Agents

This section explains the integration of perceptual agents in this framework. The most important source of data for perceptual agents, are the actual percepts that are generated from the Simulation Environment. These percepts must be captured by the perception component of the agent, which might drastically change depending on the task at hand. To cope with the variety of perception systems and interfaces, we integrated ROS in the Simulation Environment to provide the sensory data to the perception system. Because UE4 is available on the most common platforms, we have used rosbridge [15] to provide a platform-independent simulation interface.

The main sensory data of our agent, are the RGBD percepts of the virtual environment. Because these perception data streams are demanding a large amount of link bandwidth, it is necessary to transfer this binary data in a compact manner. Rosbridge uses JSON for communcation, which has no built-in type for binary data and must therefore be converted to base64, which is computationally expensive. To optimize the communication performance in our framework, we extended rosbridge to support BSON serialization[16]. This improvement has been merged into the official rosbridge codebase to help others with similar performance demands.

The generation of RGBD data is realized by a specialized virtual camera in the simulated environment that generates RGB, object mask and depth data. The rendering code of this camera is based on [6]. It has been extended to support the BSON encoding of the sensor data, the inclusion of sensor noise models and the necessary integration to use it within ROS including the underlying coordinate frames and camera parameters.

Each perceptual agent can use ROS Topics to get the percepts and camera parameters from the Simulation Environment. The High-Level Control of the agent must be instantiated by implementing an interface provided by the High-Level Agent Control in order to have control over the lifecycle of the agent running in the simulation.

As an exemplary perception system for mobile robots, we integrated RoboSherlock[17] into our framework. It is ROS-compatible and wraps a variety of vision algorithms in order to classify certain structures in sensor data. When receiving a percept, RoboSherlock can analyze the data and maintain an internal belief state of the environment. After analyzing the data, every percept can be saved in an internal database alongside with all the hypotheses of the individual perception algorithms inside RoboSherlock. The logged percepts and annotations can not only be used for learning problems, but are also suitable to debug already trained models on a percept- or scenario-level.

The Critic measures the performance of the current agent in the selected Scenario. This happens by observing the current state of the Scenario Runner and the High-Level Agent Control and gathering the meta data about the currently executed Scenario. Additionally, the Critic captures the computed belief state from the perception system and compares this with the ground truth of the current Scenario. The resulting feedback of this evaluation will be send back to the perception system which is able to improve its own models by incorporating suitable learning techniques.

[2] https://github.com/code-iai/ROSIntegration

Fig. 4: The platform for the experiments. The REFILLS platform has been developed at the IAI@Uni Bremen.

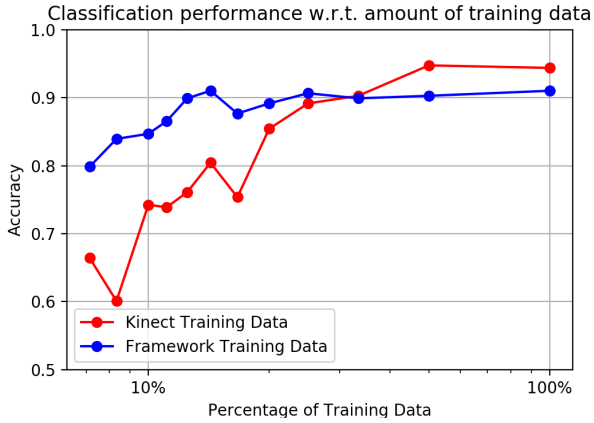

Fig. 5: Accuracy results for the example object recognition system s_1. Training samples are a) captured from real world kinect and b) generated by our framework. The plot shows that there is only a small gap between real world training data and data generated by our framework.

Even though the proposed system is capable of providing data for supervised learning problems, it could also be used to systematically evaluate vision algorithms, like in [6] and [5]. Different setups of object constellations or orientation changes can be modeled in our SDL and then systematically tested with our proposed software system.

IV. EXPERIMENTS

The framework has been created to support the development of robust perception routines that can benefit from percepts that are following a task-specific distribution. To fully evaluate this approach, an extensive amount of annotated real world data is required that has been acquired by a mobile robot in long-term tasks. Since it is not feasible yet to create such a comprehensive data set for different manipulation tasks, we will evaluate this framework in a different manner and show the plausibility of the framework first.

In the following section, we will discuss the usage of our framework to train an object recognition system s_1 for a real robot and compare this against s_1 trained with images from a Microsoft Kinect 2.0 to provide a baseline. The focus of the experiments is not to propose a certain object recognition approach, but rather show that the training data generated by our framework can be used with a comparable performance on a real robot.

A. Approach

The experiments have been conducted in a super market mockup, which features multiple shelves and products of everyday use. To generate learning inputs for the robot, a super market environment has been modeled in the Simulation Environment of our framework. The fixtures in the virtual environment are composed of 3D-Models, while the products have been created by using a Go!Scan50 Handscanner. To evaluate the approach of learning different product appearances, we have selected a set of 22 products for the experiments. A Normative Scenario models the potential position of these products in the virtual super market. It also features two uniform distributions d_1 and d_2. d_1 models the selection of a product p that is to be trained, while d_2 models a change in rotation of the product's vertical axis in the interval of $[0, 2\pi)$ radian. By sampling the Normative Scenario 7700 times, we achieve 7700 Concrete Scenarios which feature different products and changed poses on which the learning agent can be trained. The High-Level of the learning agent calculates the viewpoint relative to p and sets the virtual RGBD camera in the Simulation Environment accordingly. RGBD percepts are collected by RoboSherlock to segment p from the sensor data and generate a ROI in the RGB image that only features p. The percept and segmented data are stored in the Percept Database.

To generate real world training data a Kinect 2.0 sensor has been used. Capturing such training data is very labor-intensive and therefore naturally limited. For each of the 22 products, 72 images have been taken by rotating the sensor around each product in steps of $\frac{\pi}{36}$ radian. This results in a total of 1584 training images. For each of the training images, the shown product is segmented and stored.

To train the object recognition classifiers $c_{framework}$ and c_{kinect}, the framework and kinect training images are fed seperately into the pre-trained ImageNet [18] in the ILSVRC-2012 variant and features from the final hidden layer are extracted. These „DeCAF7" features [19] are used to train k-NN classifiers with the Chi-Square distance:

$$d_{x,y} = \sqrt{\sum_{i=1}^{j} \frac{(x_i - y_i)^2}{x_i + y_i}} \quad (1)$$

During classification, a ROI of a product candidate is fed through the ImageNet and DeCAF7 features are extracted. After performing a k-NN search with $k = 3$, the predicted class is equal to the class most common among the k neighbors. An evaluation dataset has been recorded with the Intel RealSense 435 camera of the robot. Each product is recorded in up to 14 different configurations, including partial occlusions, varying distances and views of the generally less feature-rich backside of products.

B. Results

Both classifiers c_{kinect} and $c_{framework}$ have been evaluated on the described dataset. To study the performance effect of the number of used training images, both classifiers have been evaluated multiple times while reducing the set of training data. The results of the experiments are shown in Fig. 5. Both classifiers show a similar performance when presented with high percentages of the training data sets. While c_{kinect} outperforms $c_{framework}$, it takes the classifier a considerable amount of training data to achieve this performance. Misclassifications by $c_{framework}$ are mostly due to a confusion of two objects which are equal in size, shape and very similar in their texture. We suppose that c_{kinect} performs better for these objects because the training data of the kinect data is generally slightly sharper than the texture of the 3D-Models, which is a result of our 3D Handscanner.

We showed that the accuracy gap between an object recognition system with training data from a kinect and our framework is small. We expect that with developments in 3D scanners in the near future the gap between simulated and real world training data is even closer. Generating training data in the real world is usually very labor-intensive and is therefore naturally limited. By using our framework, we have the possibility to generate arbitrary amounts of high quality training data. Having this possibility allows us to generate useful training samples of many different configurations of complex scenarios that could not be efficiently reproduced, captured and annotated in the real world. This may lead to robust robotic systems, that can generate the necessary training data for vision systems in complex everyday activities.

V. RELATED WORK

Qiu et al.[6] introduced UnrealCV, which is a framework to use UE4 for vision tasks. In their work, they generated a synthetic image dataset with ground truth annotations and tested a neural network model against images from varying viewpoints of a scene. Zhong[20] used UnrealCV in a reinforcement learning setting where moving agents should learn to avoid collisions. Our approach allows a similar set of features, but also includes tools to manipulate the virtual environment during runtime and handle the execution of batches of scenarios. Skinner et al.[5] studied the usage of game engines in vision. They showed a relation between the performance of vision algorithms on real world and synthetic data in a SLAM experiment. The authors also stated that „Integrating the game engine into the ROS environment would enable us to run a robotic vision algorithm in a manner analogous to a hardware-in-the-loop simulator, with its output commanding the camera pose in the simulator and the rendered image serving as input.". With our published ROS communication interface, this integration is made possible.

Shah et al.[21] introduced a UE4-based simulator for multicopters and associated learning tasks. Our work extends this approach by utilizing a Scenario Description Language to bootstrap different learning environments, controlling an arbitrary complex agent in it and providing a more generalized way of accessing the simulator by integrating the well-known ROS framework instead of a domain-specific RPC API. Ganoni et al.[22] also created a UE4-based simulation for multicopters and measured the effect of wind or light changes on the performance of tracking algorithms in a fixed natural environment. Related work on scenario descriptions varies between domains. The Scene Description Format[23] is often used with the Gazebo Robotic Simulator to describe virtual environments. However, it does not include the necessary Scenario Meta Data and High-Level Parametrization to handle a batch of experiments to run from different scenario descriptions. The STISIM vehicle simulator[24] was used in research to study the effects of different in-vehicle scenarios. The subject will face pre-programmed scenarios which are defined in a „Scenario Definition Language". Possible definitions in this language are focused on placing pre-defined traffic-related objects into a scene or changing the conditions of the road environment. The fidelity of the renderings are subpar to UE4, which makes it impossible to use STISIM for real world vision tasks.

VI. CONCLUSION

We presented a novel framework with a specialized scenario description language to automatically create training data that follows a task-specific distribution. Respecting this distribution for training data is important, because many learning algorithms assume that the training data follows the same distribution as the input data of the target domain. The gathered data is valuable for robots, because modern game engines allow us to create training data from a photorealistic simulation environment. This has the potential to increase the robustness of future vision algorithms, because they can be trained on more realistic data while accounting the actual distribution of the target domain.

In our experiments, we showed the plausibility of the framework by demonstrating an example object recognition system on a real robot. The system showed similar performance results when trained with real world data versus the same system trained with data generated by our framework.

In future work, we strive to use this framework in the context of mobile household robotics. A robot with a basic perception system will be confronted with multiple household tasks. It shall then improve its perception performance by using our framework to benefit from training data that follows a task-specific distribution.

We will also investigate the usage of Virtual Reality(VR) in conjunction with our framework. To learn real world distributions of objects in specific tasks, subjects could be asked to fulfill this task in a virtual environment with VR devices. VR allows the subject to almost naturally interact with the virtual world and therefore to manipulate the environment in a similar way to how it would have been done in the same task in the real world, getting more realistic data.

ACKNOWLEDGMENTS

This work is partially funded by Deutsche Forschungsgemeinschaft (DFG) through the CRC 1320, *EASE* and by the EU H2020 project *REFILLS* (Project ID: 731590).

REFERENCES

[1] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[2] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 3234–3243.

[3] X. Peng, B. Sun, K. Ali, and K. Saenko, "Learning deep object detectors from 3d models," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1278–1286.

[4] A. Teichman and S. Thrun, "Tracking-based semi-supervised learning," in *Robotics: Science and Systems*, Los Angeles, CA, USA, 2011.

[5] J. Skinner, S. Garg, N. Sünderhauf, P. Corke, B. Upcroft, and M. Milford, "High-fidelity simulation for evaluating robotic vision performance," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2016, pp. 2737–2744.

[6] W. Qiu and A. Yuille, "Unrealcv: Connecting computer vision to unreal engine," *arXiv preprint arXiv:1609.01326*, 2016.

[7] S. Qiao, W. Shen, W. Qiu, C. Liu, and A. Yuille, "Scalenet: Guiding object proposal generation in supermarkets and beyond," *arXiv preprint arXiv:1704.06752*, 2017.

[8] M. Roberts, D. Dey, A. Truong, S. Sinha, S. Shah, A. Kapoor, P. Hanrahan, and N. Joshi, "Submodular trajectory optimization for aerial 3d scanning," in *International Conference on Computer Vision (ICCV) 2017*, 2017.

[9] A. Farhadi, I. Endres, D. Hoiem, and D. Forsyth, "Describing objects by their attributes," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 1778–1785.

[10] C. H. Lampert, H. Nickisch, and S. Harmeling, "Attribute-based classification for zero-shot visual object categorization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 3, pp. 453–465, 2014.

[11] I. D. Horswill, "Specialization of perceptual processes," Ph.D. dissertation, Massachusetts Institute of Technology, 1993.

[12] Epic Games, "What is Unreal Engine 4," http://www.unrealengine.com, 2018, [Online; accessed 10-09-2018].

[13] A. Lerer, S. Gross, and R. Fergus, "Learning physical intuition of block towers by example," *CoRR*, vol. abs/1603.01312, 2016. [Online]. Available: http://arxiv.org/abs/1603.01312

[14] Open Source Robotics Foundation, "tf2," http://wiki.ros.org/tf2, 2018, [Online; accessed 10-09-2018].

[15] C. Crick, G. Jay, S. Osentoski, B. Pitzer, and O. C. Jenkins, "Rosbridge: Ros for non-ros users," in *Robotics Research*. Springer, 2017, pp. 493–504.

[16] MongoDB, Inc., "BSON Specification," http://bsonspec.org, 2018, [Online; accessed 10-09-2018].

[17] M. Beetz, F. Bálint-Benczédi, N. Blodow, D. Nyga, T. Wiedemeyer, and Z.-C. Márton, "Robosherlock: Unstructured information processing for robot perception," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1549–1556.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[19] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *International conference on machine learning*, 2014, pp. 647–655.

[20] F. Zhong, "Integrate unreal engine with openai gym for reinforcement learning based on unrealcv," https://github.com/zfw1226/gym-unrealcv, 2016, [Online; accessed 10-09-2018].

[21] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017. [Online]. Available: https://arxiv.org/abs/1705.05065

[22] O. Ganoni and R. Mukundan, "A framework for visually realistic multi-robot simulation in natural environment," *arXiv preprint arXiv:1708.01938*, 2017, wSCG 2017 proceedings.

[23] Open Source Robotics Foundation, "SDFormat," http://sdformat.org/, 2018, [Online; accessed 10-09-2018].

[24] Y. I. Noy, T. L. Lemoine, C. Klachan, and P. C. Burns, "Task interruptability and duration as measures of visual distraction," *Applied Ergonomics*, vol. 35, no. 3, pp. 207–213, 2004.