2022 IEEE 2nd International Conference on Intelligent Reality (ICIR) | 978-1-6654-8755-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICIR55739.2022.00018

CityLifeSim: A High-Fidelity Pedestrian and Vehicle Simulation with Complex Behaviors

Cheng Yao Wang
Cornell University, USA

Oron Nir
Microsoft, Israel

Sai Vemprala, Ashish Kapoor
Microsoft, USA

Eyal Ofek, Daniel McDuff
Mar Gonzalez-Franco*
Microsoft Research, USA

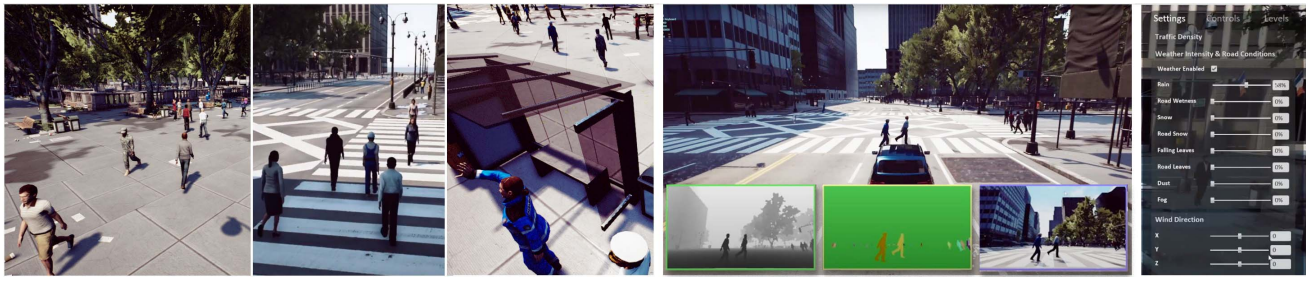

Fig. 1: Screenshots of the simulation of pedestrians inside CityLifeSim: walking by a park, crossing a street, hailing a bus. A view of the depth, segmentation and RGB maps as seen by a vehicle in the street. The weather parameters available.

***Abstract*—Simulations are a powerful tools particularly in the case of safety critical scenarios. However, simulating complex temporal events in multi-agent scenarios with vehicles and pedestrians, such as those that exist in urban environments, is challenging. We present CityLifeSim, a simulation for the research community that focuses on rich pedestrian behavior, such as the one that arises when different personalities, environmental events, and group goals are simulated. In our simulations we can see cases of people jay walking a red light, sitting on a bench, waiting for the bus, or calling on the phone, but also more complex creation and management of crowds that might even line up or just keep moving while observing interpersonal distances. CityLifeSim is configurable and can create unlimited scenarios with detailed logging capabilities. As a demonstration we have run CityLifeSim to create a demo dataset for training setups that includes 17 different cameras, views from a moving vehicle in the street under different weather conditions (rain, snow, sun), and from a drone with frontal and downward views. All content is released with the corresponding original configuration files, ground truth pedestrian segmentation, and RGB-D frames. We evaluate our dataset on a pedestrian detection and identification task with state of the art Multi-Object Tracker (MOT), showing the limitations and opportunities for synthetic data in this use case.**

***Index Terms*—pedestrian simulation, self-driving cars, causal ML, dataset**

I. Introduction

Simulations are a cornerstone of complex games and environments, and they will have a central role also for VR and AR setups. Furthermore, given the pivotal role that data plays in machine learning systems, simulations are also becoming a powerful part of the infrastructure for training models. Computer vision systems have long used simulations for evaluation [1]–[6], as they provide a way for systematically varying parameters and generating samples with "perfect" labels. More recently, as simulations have improved and data-hungry deep learning systems have become more common, simulations have been used for creating training data [7], [8]. Researchers have designed synthetic pipelines to generate samples that help to address problems such as a lack of representation [9] and bias [10]. Simulations have also proven useful in more nascent domains, such as causal and counterfactual reasoning, as the parameters of synthetic environments can be used to create causal relationships between elements and introduce confounders [11]–[13].

These have all surfaced as interesting spaces where simulations are thriving and growing, but the area where simulations are becoming essential is in safety-critical scenarios, where it may not be ethical or realistic to witness or "cause" specific sequences of events in the real world.

Autonomous systems that operate in environments with humans require certain fundamental components that fall in the realm of safety-critical applications. For example, any autonomous vehicle that needs to perform pedestrian detection, trajectory prediction, and is required to reason about their actions has to be trained to deal with varied, and perhaps rare, safety-critical scenarios related to pedestrians. However, it may not be possible or ethical to cause real world unsafe situations for experimentation and testing; hence simulations such as the presented system are essential for designing such machine learning systems. Unsurprisingly, synthetic data have been employed heavily in autonomous driving scenarios, e.g., GTACrash [14], VIENA^2 [15], CARLA [16], AirSim [17], and, more recently, CausalCity [18].

Despite the existence of multiple datasets and simulations,

*Corresponding author: margon@microsoft.com
Our simulation and dataset are available on - CityLifeSim.github.io

978-1-6654-8755-9/22/$31.00 ©2022 IEEE
DOI 10.1109/ICIR55739.2022.00018

Fig. 2: A city environment map showing locations of points of interest (POIs) and waypoints, used to create complex pedestrian and vehicle scenarios. Map's green regions highlight "low-risk walkable" areas for pedestrians. Pedestrians' spawning positions and waypoints are shown the right image. Both waypoints and POIs' interactions are defined in the configuration file.

there are still some unresolved weaknesses or gaps in the widely available tools. Firstly, public simulation environments are often low-fidelity, both visually and in terms of behaviours (e.g., CARNOVEL [19]). The second is the lack of configurable control to create specific and complex sequences of events involving multiple agents. In particular, the support for pedestrian behaviors is often limited. GTACrash and VIENA^2 [15] are created on top of the videogame Grand Theft Auto V (GTA), which provides high visual fidelity, but still not customizable enough with only a static dataset and with limited access tothe underlying simulation. CARLA, on the other hand, provides a similar visual fidelity to AirSim, and also has pedestrians that can navigate sidewalks, understand traffic lights and crossings etc. But they lack further support for crowd intelligence, personality of agents or otherwise more complex interactions with the environment. Which in CityLifeSim we make available through particular support for POIs, and individual agent configurations, which ultimately allows simulating pedestrians and vehicles with rich behavioral variety.

In the remainder of this paper we: 1) Present a novel, state-of-the-art, highly configurable simulation, released as an executable, 2) create a snapshot dataset and benchmark performance for pedestrian detection and identification, 3) define a set of tasks that demonstrate the potential of such a simulation for safety-critical tasks.

II. Simulation

Our goal in creating CityLifeSim is to provide a city simulation environment where vehicles and pedestrians have realistic and complex interactions. This system can be used both to create smarter agents in immersive environments, as well as be used to train machine learning algorithms.

CityLifeSim is built on the Unreal Engine, which is also the main engine behind the existing environment of the **AirSim** [17] vehicle simulator, which can then be configured externally. Unreal Engine has also been used in the past for creating data employed for interrogating computer vision systems [6]. AirSim, provides the backbone of our

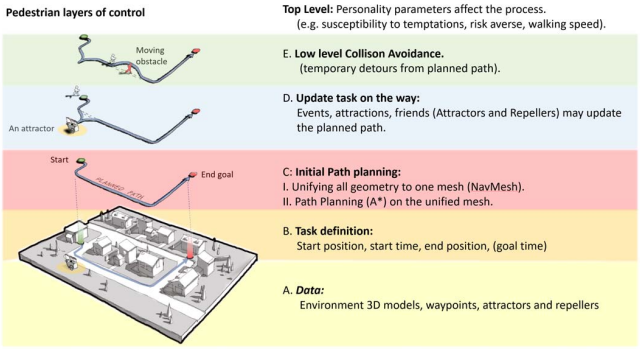

Fig. 3: Pedestrian automaton uses multiple simulation layers that will affect their paths, from the 3D model to the personality parameters in the configuration file that will affect all decision making, to the low level collision avoidance.

simulation and already contains the capability to render cars and drones that are controllable through a Python API. It enables environment variations such as weather, time of the day, and visibility parameters, to create complex, varied, and visual datasets. AirSim also allows recording a range of modalities (e.g. color, depth, segmentation maps) from multiple cameras for logging/visualization.

We simulate more complex vehicle behavior on top by leveraging the AI traffic modules from [18]. By changing the simulated pedestrian's and driver's objectives (targets) and preferences (such as detour susceptibility), we can obtain high-fidelity pedestrian and vehicle simulations that can be paired and scripted at a high-level using CSV or JSON configuration files.

At run-time, a pedestrian, defined through a configuration file, starts walking from its spawn point through zero or more waypoints toward its final goal position. The spawn or goal points are selected from a list of such feasible points in the environment (Figure 2). The spawn time is spread along the simulation time.

On top of the pre-configured pedestrian course, a built-in

function enables reactive obstacle avoidance behavior, and an additional dynamic layer that will make the pedestrian react in different ways to POIs in the environment, after pedestrians execute an update on their task they return to their original target.

A. Pedestrian Modeling

In CityLifeSim the pedestrian simulation is a complex multi-layer process (Figure 3). The final trajectory executed by a pedestrian takes several aspects into account: such as the 3D model, existing environment, and the pre-configured route, all the way to local avoidance, and finally the interactions with events it might decide to spend time on during the simulation.

CityLifeSim also enables the pedestrians to stray off the path to approach/respond to points of interest (POI). For example, POIs allow pedestrians to join a friend in the street, sit on a bench, stop at a traffic light, stop to watch a street event, or form a line to catch the bus. Pedestrians that engage with the environment will do so for a limited time and then return to their original path. Such behavior is governed by a pre-configured probability that controls how likely a pedestrian will be affected by particular POIs. Thus creating varied behaviors for different pedestrians that are important in particular for simulating rich street environment with unique situations.

Characters and Animations. Into the city environment we integrate the graphics and animation assets from the Microsoft Rocketbox [20] library of avatars that consists of 115 characters and avatars fully rigged for animations and with high definition. These avatars have been widely used for Virtual Reality and other libraries to animate them have been also released. The library also contains over 1000 animations that can be combined and used among the avatars. A column in the configuration file indicates which identity will be used for which person.

B. Points of Interests - POIs

As a pedestrian walks along its planned path, different resources or events in the environment that may lure them to modify their path, getting closer or further away from these points (Figure 4). In CityLifeSim we model such events using *Points-of-interests* (POIs) of three types: **Attractors** that lure pedestrians (Figure 5), **Repellers** which drive pedestrians away, and **Blockers** that represent impassable areas. Once a pedestrian steps into a POI, there is a possibility she will change her path, and behaviour depending on their pre-configured personality and the state of the POI.

All POIs follow a similar structure, they have an *impact area* around them, a *capacity* defining the number of concurrent served pedestrians, and the *service-time* that each person stays at the POI. Pedestrians follow the pre-configured probabilities together with rules such as proximity [21] to define if they are going to be affected by the POI. After a pedestrian has completed the interaction with the POI they will return to their path, leaving a vacancy for the next person.

a) Attractors.: Are defined with an area of attraction around them and a smaller area of interaction inside them. Pedestrians that enter the area of attraction will be directed to the interaction space, that might take more or less time. This might generate cues or crowds, depending on how the attractor is defined and how many people it can serve. In our system we have three examples of attractors as seen in Figure 5.

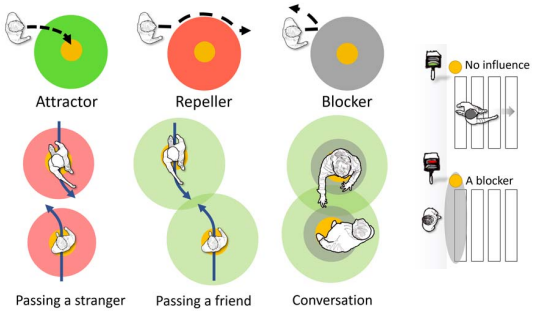

Fig. 4: a). Attractors (green) draw people toward them, Repellers (red) represent points that people avoid, and Blockers (gray) represent non-traversal areas. POIs might also have influence only in some pedestrians, or in certain times. As such blockers can also be used to halt pedestrians at traffic lights.

Fig. 5: Attractor examples with associated times and animations. The orange box is the attraction area, the cylinder indicates the direction and the dots highlight where a queue will form. Left: Street dancing gathering. Center: A bench, can serve only one person. Right: A bus stop with waiting pedestrians that form a line while the first pedestrian hails for the bus. Bottom: Attractors at run-time in the simulation.

b) Repellers.: The repel area is defined with a grid of points affected by the repeller. Each value has a probability of a pedestrian repelling from their path. A local path planning phase is used to generate a path that will minimize stray from the path, yet keep the pedestrian at a distance from the repeller (a function of the pedestrian *susceptibility to influence*. In CityLifeSim we implemented an example repellent: a loud angry person on the street. (Figure 6).

Fig. 6: A repeller example. Top) A loud person on the phone repels pedestrians from her near vicinity (POIRepeller1). The repel probability values are defined in a grid around the repeller. Bottom) At runtime, a local path finding is calculated in the repulsion area.

c) Blockers.: These are used to represent an impassable obstacle. A pedestrian whose path goes through a blocker will be stopped. Blockers are used to guide and protect pedestrians, a blocker in front of a car prevents pedestrians to step in front of the car, or to temporarily stop pedestrians when the crossing traffic signal turns red (POIBlocker1). We use only moving or temporary blockers, permanent blockers can be implemented in the geometry as normal obstacles.

C. Pedestrian configuration file

The whole Pedestrian simulation can be configured externally using a CSV configuration file as shown in Figure 7.

Most importantly a pedestrian's *susceptibility to influences* parameter (POIProbs), represented by a score between 0 and 1, is the probability the pedestrian may choose to stray from its main goal to interact with different POIs. These probabilities are the root of each pedestrian's behavior.

Pedestrian		Task definition			Behavior parameters			Rendering	
Key	Id	Spawn	Spawn Radius	Waypoints	Walk Speed	POITypes	POIProbs	Appearance Idx	Anim Type
Ped_1	Ped_1	B1_top	250	BD_topleft,BC_topright A1_top,C1_right	150	POIRepeller1,POIAttractor1, POIAttractor2,POIAttractorQueue1, POIAttractorQueue2,POIBlocker1	0.0,0.2, 1.0,1.0, 0.5,0.5	Business_Female_03	Female
Ped_2	Ped_2	A1_top	250	AD_topleft,AC_topright, C1_right, A1_top	100	POIRepeller1,POIAttractor1, POIAttractor2,POIAttractorQueue1, POIAttractorQueue2,POIBlocker1	1.0,0.8, 0.7,0.2, 1.0,0.0	Business_Male_06	Male
Ped_3	Ped_3	C1_right	250	A1_top,B1_top, BD_topleft,BC_topright,	120	POIRepeller1,POIAttractor1, POIAttractor2,POIAttractorQueue1, POIAttractorQueue2,POIBlocker1	0.8,1.0, 0.5,0.3, 0.2,1.0	Business_Female_04	Female

Fig. 7: An example configuration file, defining 3 pedestrians. Each pedestrian has a unique ID, task (spawn position and target waypoints), walking speed, probability of interaction with different POI, and rendering parameters (model, animations).

D. Path Finding

Path finding is the process of defining the valid, high level path executed by each pedestrian. It is executed first when a pedestrian receives their task of walking from a start position to the next way point, and again at each way point toward the next one. The path may also be executed locally, to adapt the pedestrian path accordingly once a decision is made to take a detour from the planned path toward an attractor or away from a repeller. The pathfinding procedure in CityLifeSim uses the 'Recast and Detour' algorithm to find the shortest collision-free path from the current pedestrian position to the next waypoint. Recast and Detour is available as a native Unreal Engine feature - it first calculates the *NavMesh* which merges all known objects in the environment to one unified mesh (see the Blue mesh in Fig. 8 b and d). Upon this mesh, an A* algorithm [22] is then used to find a feasible path for each pedestrian as required.

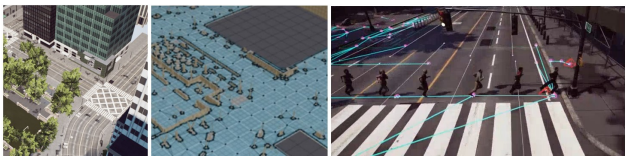

Fig. 8: Left). Part of the urban environment model. Middle). Navmesh generates a unified mesh that incorporate all obstacles. Right). A* is used to generate a path (light blue lines) that pedestrians then follow.

a) Collision avoidance.: CityLifeSim deals with moving obstacles, not included in the NavMesh, using the Reciprocal Velocity Obstacles (RVO) system inside the Unreal Engine. It looks at the moving agents in the vicinity of the pedestrian, assuming temporally constant velocity, and temporarily changes the pedestrian velocity, then tries to return to the original planned path (again, using collision avoidance if needed).

b) POI interactions: When a pedestrian enters the influence zone of a POI, it may modify its path, creating a new path towards or away from the POI. Once the pedestrian finishes its interaction, CityLifeSim recalculates the A* path to the original waypoint destination.

E. Vehicles

The backbone of our vehicle simulation is based on AirSim [17]. Additional vehicles are driven via an AI traffic module.[1] This module handles low-level navigation and the vehicles traverse the scene along predefined splines. In the configuration file of the vehicles, each vehicle is given a *spawn* point and a list of *actions* to execute. Actions are slightly different from the notion of pedestrians' waypoints but the rules are similar. Merging actions happen along lane splines and turning actions happen at intersections. More details about the vehicle control simulator can be found in **CausalCity** [18]. Vehicles in CityLifeSim have an invisible collider in the front so that the cars will stop if an obstacle such as a pedestrian enters that space, until the object is gone. This cone area is also used to stop the car when it detects red light ahead. This behavior has exceptions, as it only makes the risk of collisions lower, similar to the real-world, if a car is driving fast towards an intersection and brakes late, it can still pass the light and risks colliding with objects or pedestrians.

F. Traffic Signals

Our simulated environment contains traffic lights, and both vehicles and pedestrians respond to them. To do so, we use POI blockers that are coordinated with the light signals. Besides keeping the traffic flowing in a realistic manner, traffic signals, also introduce causal connections at the intersections. The sequence and timing of these lights can be controlled during the scenario run-time. Both pedestrians and vehicles can be "forced" to continue progress during red light, to simulate dangerous events. To do so, the blocker's probability can be reduced in the configuration file.

G. Logging

CityLifeSim provides rich logging capabilities, ranging from recording visual modalities such as color, depth and segmentation frames, to logs of pedestrians and vehicles locations and velocities, their visibility in all sensors, traffic lights state, weather conditions and more. Besides logging, CityLifeSim also enables queries at run-time on the state of different objects and agents on the scene that are visible through an API built on top of Airsim. We provide examples of logging on our project page and in our snapshot dataset.

[1]https://www.unrealengine.com/marketplace/en-US/product/arch-vis-ai-traffic-system

III. Snapshot Dataset

To further facilitate the tasks of those who want to use this tool for machine learning applications, we created, using CityLifeSim, a snapshot dataset, containing videos, depth images and segmentation maps. The pedestrian IDs are indexed consistently over time and between cameras. There a total of 128 pedestrians in each video and we provide the bounding box ground truth with individual IDs for each pedestrian.

The color depth and segmentation videos include: 17 simulated cameras covering different waypoints and POIs from Figure 2, each camera runs for about 13 seconds. 3 videos of a car driving view, under different weather conditions: sunny (137 seconds), raining (141 seconds) and snowing (134 seconds). 2 videos of drone views, with a drone looking up front (94 seconds) and a drone looking down (94 seconds) from a zenithal perspective. The simulations can be re-run by replaying the same configuration files that are provided with the dataset. These videos have a sampling rate of 10 frames/second, and a resolution of 1024 x 640 pixels. For examples of the videos see our project page.

Example Pedestrian Detection Task. We use the task of pedestrian detection to exemplify what the simulation might be used for. Pedestrian detection is also the backbone for many other tasks, such as trajectory calculations or causal reasoning or Representation Learning [23]. Which might also enable many other tasks on the dataset. The Multi-object Tracking (MOT) task for pedestrian detection is tested using the state-of-the-art tracker - ByteTrack [24] and evaluated with standard metrics according to MOTChallenge [25], namely MOTA, MOTP, and IDF1. As opposed to real-life footage, ground-truth labels are derived from segmentation maps instead of manual annotations. The use of segmentation maps can help improve ML models, as they can get pixel-accurate ground truth and outperform models trained with human annotators, who would not recognize small segments as pedestrians. In fact in our own dataset, we saw a significant drop on performance if we accounted for bounding boxes smaller than 10x10 pixels using a priory human labeled dataset ($t(16)=2.11$, $p<0.001$, from $19\pm2\%$ to $36\pm4\%$ MOTA). Our results with the filtered bounding boxes are shown in Table I.

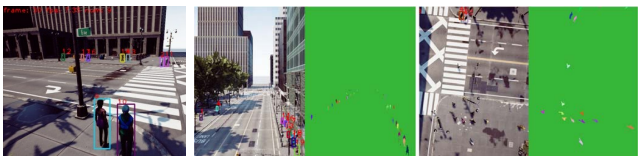

Fig. 9: Pedestrian detection performance. The colored boxes reflect the bounding boxes of the tracker, the green segmentation map is the ground truth. Examples of high and low accuracy detection. [24].

Factors like camera position and the pedestrian size distributions may yield either poor tracking results or state-of-the-art performance. While cameras like “Cam 11” had high accuracy, mostly missing pedestrians only as they were hidden behind some obstacle (Figure 9); results on videos from the drone top view were significantly worse than those from street level cameras, due to the fact that the detection model was not optimized for zenithal views of pedestrians. Pedestrians very far from the camera were too tiny to be detected. Other cameras like “Cam 1” had very few pedestrians and when they entered the frame, they were half cropped. Particular illumination conditions as well as weather conditions also dropped the accuracy, making it worse for snow conditions. It should be noted that the frame rate of our videos was one third (10Hz) of that used in the MOT20 challenge (30Hz), some performance reduction is expected to be due to this too.

Scenario	Conditions	FN[%]	FP[%]	IDSW[%]	MOTA[%]	MOTP[%]	IDF1[%]
Cam 0	Sunny	17.0	7.9	0.0	75.1	26.0	87.0
Cam 1	Sunny	87.9	6.8	0.0	5.3	27.2	20.3
Cam 2	Sunny	32.3	41.3	1.4	25.1	31.6	58.4
Cam 3	Sunny	44.9	37.2	0.7	17.1	29.4	47.5
Cam 4	Sunny	25.7	58.0	1.3	15.0	32.7	54.1
Cam 5	Sunny	43.2	29.0	0.8	27.0	33.6	50.6
Cam 6	Sunny	20.9	15.4	0.1	63.5	34.5	76.6
Cam 7	Sunny	24.5	44.3	0.7	30.5	32.8	61.3
Cam 8	Sunny	30.4	31.2	0.1	38.3	33.9	68.0
Cam 9	Sunny	29.9	26.2	0.4	43.5	30.5	68.9
Cam 10	Sunny	39.9	17.4	0.8	41.9	32.2	62.8
Cam 11	Sunny	22.5	9.8	0.8	66.8	28.6	69.3
Cam 12	Sunny	56.2	19.3	0.8	23.7	31.7	39.9
Cam 13	Sunny	63.6	15.8	0.6	20.1	31.6	45.7
Cam 14	Sunny	48.2	22.4	0.5	28.9	38.7	57.2
Cam 15	Sunny	22.5	12.3	1.6	63.6	30.5	65.3
Cam 16	Sunny	55.5	12.3	0.7	31.5	27.8	52.1
Car	Sunny	35.5	10.1	2.1	52.3	27.2	17.6
Car	Rainy	42.0	7.9	1.9	48.2	26.6	19.1
Car	Snowy	46.8	10.5	1.9	40.8	26.6	17.5
Drone	Top view	99.2	0.7	0.0	0.1	40.6	1.5
Drone	Frontal	87.4	38.8	0.1	26.2	38.0	16.5

TABLE I: MOT evaluation of CityLifeSim dataset. Statistics: False Negative detection rate (FN), False Positive detection rate (FP), Identification swaps (IDSW), standard MOT metrics [25] MOT Accuracy (MOTA), MOT Precision (MOTP), and track identification preservation F1 score (IDF1).

IV. Discussion, Limitations and Broader Impact

During the creation and testing of CityLifeSim and our Dataset, we have identified some key contributions that make our work of particular relevance, but also some limitations.

First, CityLifeSim enables the generation of data that can be hard to obtain in other ways, either because it happens very rarely or because it creates safety concerns. With our level of configuration we can create such data much faster. The level of control provided and the rich behaviors that it generates are also well beyond other simulation tools available. This makes the system interesting for the creation of synthetic data that can later be use to train new models. Data that relies on manual annotation is limited to human perception, thus ground truth is only as good as the human. Whereas using a simulation we can achieve perfect pixel level labels for each object category regardless of the size, without introducing errors due to human judgement in the labeling process. This can help in interrogating current ML tools, and pushing performance beyond manual tagging. And while this is true for all simulations, again, the realism of the behaviours we are able to produce can

have significant impact on this area. Our benchmark task on our dataset shows some practical example of how the use of simulated ground truth data might affect existing models.

Third, CityLifeSim also allows us to generate observations of events from any view point (e.g., cameras), times of day, weather conditions, etc. This means we can create datasets with systematic variations in parameters and specific distributions and then observe the effects of changing the parameters on model performance. This, combined with the rich behaviors that can be created, opens the door to a complete new set of tasks such as causal discovery of relationships between agents. Additionally the real-time nature of the simulation also makes it a good candidate to be used in VR setups that require crowds or multiple agents.

Of course, there are also limitations to our work. Any simulation is only an approximation to the real-world. While CityLifeSim has a considerable amount of functionality and flexibility we do not argue that it creates perfectly realistic behaviors or that the visual appearance of the scenes is equivalent to a camera. Furthermore, it is a limited context simulation, reflecting a relatively small geographic region modeled on a city block. The environment, vehicles, and pedestrians have limited diversity. As such, it is designed for experimentation and exploration of machine learning techniques and not for training autonomous systems that will be deployed in the real-world. Nevertheless, it is still a capable tool and we hope the research community will benefit from it.

And despite, we show the performance of a benchmark pedestrian detection task on our dataset, it is also relevant to note that this simulation would not be suitable without further validation and research for tasks including: person/face recognition, emotion or affect detection, identifying suspicious behaviors.

V. Conclusion

We present CityLifeSim, a high-fidelity city simulation with vehicles and pedestrians that allow for complex behaviors and high-level scenario definitions. This simulation brings together a large library of avatars and animations, a rich simulation environment with comprehensive configuration and logging capabilities, and the notion of agency that means that vehicles and pedestrians can detect and avoid each other without requiring very low-level control from the user.

References

[1] Robert M Haralick. Performance characterization in computer vision. In *BMVC92*, pages 1–8. Springer, 1992.

[2] VSR Veeravasarapu, Rudra Narayan Hota, Constantin Rothkopf, and Ramesh Visvanathan. Model validation for vision systems via graphics simulation. *arXiv preprint arXiv:1512.01401*, 2015.

[3] VSR Veeravasarapu, Rudra Narayan Hota, Constantin Rothkopf, and Ramesh Visvanathan. Simulations for validation of vision systems. *arXiv preprint arXiv:1512.01030*, 2015.

[4] VSR Veeravasarapu, Constantin Rothkopf, and Visvanathan Ramesh. Model-driven simulations for deep convolutional neural networks. *arXiv preprint arXiv:1605.09582*, 2016.

[5] David Vazquez, Antonio M Lopez, Javier Marin, Daniel Ponsa, and David Geronimo. Virtual and real world adaptation for pedestrian detection. *IEEE transactions on pattern analysis and machine intelligence*, 36(4):797–809, 2013.

[6] Weichao Qiu and Alan Yuille. Unrealcv: Connecting computer vision to unreal engine. In *European Conference on Computer Vision*, pages 909–916. Springer, 2016.

[7] Erroll Wood, Tadas Baltrušaitis, Charlie Hewitt, Sebastian Dziadzio, Thomas J Cashman, and Jamie Shotton. Fake it till you make it: Face analysis in the wild using synthetic data alone. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3681–3691, 2021.

[8] Daniel McDuff, Javier Hernandez, Erroll Wood, Xin Liu, and Tadas Baltrusaitis. Advancing non-contact vital sign measurement using synthetic avatars. *arXiv preprint arXiv:2010.12949*, 2020.

[9] Elad Richardson, Matan Sela, and Ron Kimmel. 3d face reconstruction by learning from synthetic data. In *2016 fourth international conference on 3D vision (3DV)*, pages 460–469. IEEE, 2016.

[10] Nikita Jaipuria, Xianling Zhang, Rohan Bhasin, Mayar Arafa, Punarjay Chakravarty, Shubham Shrivastava, Sagar Manglani, and Vidya N Murali. Deflating dataset bias using synthetic data augmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 772–773, 2020.

[11] Yunzhu Li, Antonio Torralba, Anima Anandkumar, Dieter Fox, and Animesh Garg. Causal discovery in physical systems from videos. *Advances in Neural Information Processing Systems*, 33:9180–9192, 2020.

[12] Kexin Yi, Chuang Gan, Yunzhu Li, Pushmeet Kohli, Jiajun Wu, Antonio Torralba, and Joshua B Tenenbaum. Clevrer: Collision events for video representation and reasoning. In *International Conference on Learning Representations*, 2019.

[13] Ossama Ahmed, Frederik Träuble, Anirudh Goyal, Alexander Neitz, Yoshua Bengio, Bernhard Schölkopf, Manuel Wüthrich, and Stefan Bauer. Causalworld: A robotic manipulation benchmark for causal structure and transfer learning. *arXiv preprint arXiv:2010.04296*, 2020.

[14] Hoon Kim, Kangwook Lee, Gyeongjo Hwang, and Changho Suh. Crash to not crash: Learn to identify dangerous vehicles using a simulator. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 978–985, 2019.

[15] Mohammad Sadegh Aliakbarian, Fatemeh Sadat Saleh, Mathieu Salzmann, Basura Fernando, Lars Petersson, and Lars Andersson. Viena: A driving anticipation dataset. In *Asian Conference on Computer Vision*, pages 449–466. Springer, 2018.

[16] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

[17] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pages 621–635. Springer, 2018.

[18] Daniel McDuff, Yale Song, Jiyoung Lee, Vibhav Vineet, Sai Vemprala, Nicholas Gyde, Hadi Salman, Shuang Ma, Kwanghoon Sohn, and Ashish Kapoor. Causalcity: Complex simulations with agency for causal discovery and reasoning. *arXiv preprint arXiv:2106.13364*, 2021.

[19] Angelos Filos, Panagiotis Tigkas, Rowan McAllister, Nicholas Rhinehart, Sergey Levine, and Yarin Gal. Can autonomous vehicles identify, recover from, and adapt to distribution shifts? In *International Conference on Machine Learning*, pages 3145–3153. PMLR, 2020.

[20] Mar Gonzalez-Franco, Eyal Ofek, Ye Pan, Angus Antley, Anthony Steed, Bernhard Spanlang, Antonella Maselli, Domna Banakou, Nuria Pelechano, Sergio Orts-Escolano, et al. The rocketbox library and the utility of freely available rigged avatars. *Frontiers in virtual reality*, page 20, 2020.

[21] Edward T Hall, Ray L Birdwhistell, Bernhard Bock, Paul Bohannan, A Richard Diebold Jr, Marshall Durbin, Munro S Edmonson, JL Fischer, Dell Hymes, Solon T Kimball, et al. Proxemics [and comments and replies]. *Current anthropology*, 9(2/3):83–108, 1968.

[22] Xiao Cui and Hao Shi. A*-based pathfinding in modern computer games. *International Journal of Computer Science and Network Security*, 11(1):125–130, 2011.

[23] Oron Nir, Gal Rapoport, and Ariel Shamir. Cast: Character labeling in animation using self-supervision by tracking. In *Computer Graphics Forum*, volume 41, pages 135–145. Wiley Online Library, 2022.

[24] Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Zehuan Yuan, Ping Luo, Wenyu Liu, and Xinggang Wang. Bytetrack: Multi-object tracking by associating every detection box. *arXiv preprint arXiv:2110.06864*, 2021.

[25] Patrick Dendorfer, Aljosa Osep, Anton Milan, Konrad Schindler, Daniel Cremers, Ian Reid, Stefan Roth, and Laura Leal-Taixé. Motchallenge: A benchmark for single-camera multiple target tracking. *International Journal of Computer Vision*, 129(4):845–881, 2021.