
World Model Based Sim2Real Transfer for Visual Navigation

Chen Liu*, Kiran Lekkala* and Laurent Itti*[†]

Abstract

Sim2Real transfer has gained popularity because it helps transfer from inexpensive simulators to real world. This paper presents a novel system that fuses components in a traditional *World Model* into a robust system, trained entirely within a simulator, that *Zero-Shot* transfers to the real world. To facilitate transfer, we use an intermediary representation that are based on *Bird's Eye View (BEV)* images. Thus, our robot learns to navigate in a simulator by first learning to translate from complex *First-Person View (FPV)* based RGB images to BEV representations, then learning to navigate using those representations. Later, when tested in the real world, the robot uses the perception model that translates FPV-based RGB images to embeddings that are used by the downstream policy. The incorporation of state-checking modules using *Anchor images* and *Mixture Density LSTM* not only interpolates uncertain and missing observations but also enhances the robustness of the model when exposed to the real-world environment. We trained the models in a CARLA[2]-based *Differential-drive* robot simulator. Our methodology's effectiveness is shown through the deployment of trained models onto a *Real-world Differential-drive* robot. Lastly we release a comprehensive codebase, dataset and models for training and deployment that are available to the public.

1 Related Work

Traditional *Sim2Real* transfer for Visual Navigation comprise methods ranging from Fine-tuning[5], Meta-learning[1], Domain Randomization[9] to System Identification[6]. Some of the recent works, such as [11] and [12], have shown promising results in attempting to cope with the Sim2real gap using *Style Transfer* but might not be suitable for visual navigation on drones and delivery robots which have limited computational bandwidth.

Recurrent world-models [3] introduces a novel approach to RL, incorporating a Vision model for sensory data representation and a Memory model for capturing temporal dynamics, all of which collectively improve agent performance. Apart from the advantages of pertaining each module, some of the modules in this architecture can be frozen after learning the representation of the environment, paving the way for more efficient and capable RL agents.

In this paper, we formulate a new setting for *Zero-shot Sim2Real* transfer for *Visual Navigation without Maps*. The system, as outlined in 1, is trained entirely on a simulated dataset and is frozen and deployed on a real-world mobile robot.

*First two authors contributed equally

[†]The authors are with Thomas Lord Department of Computer Science, University of Southern California, 90089, USA Correspondence to klekkala@usc.edu

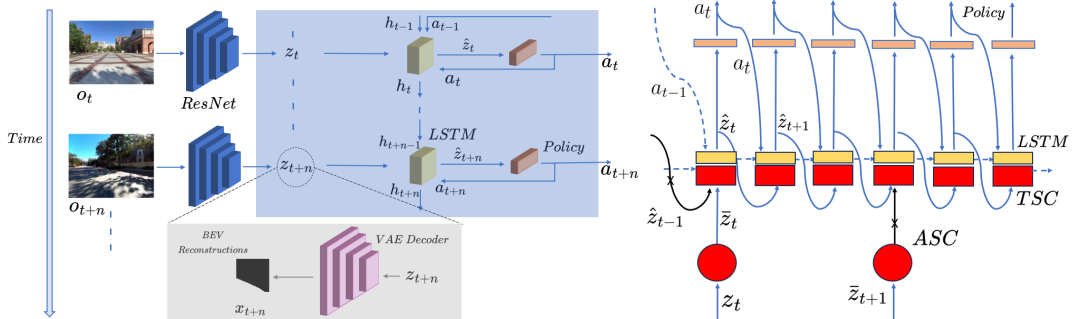


Figure 1: Left: **Working of the System.** RGB observation o_t at time step t is passed to the ResNet-50 and compressed into a Bird’s Eye View (BEV) based embedding z_t . The LSTM model takes the current latent representation and uses the historical context to refine the state into \hat{z}_t . The control policy takes the current state and gives a proper action command a_t . Blue box corresponds to the part of the system that sees the world only from a BEV based representation. Grey box depicts what would happen if we pass the output of the perception model into the VAE Decoder. Right: **Robustness Enhancement.** TSC (in Red) only takes input from the representation z_t when it comes with a high confidence score. Otherwise, it takes the previous prediction by the LSTM \hat{z}_{t-1} as interpolation. ASC improves the representation of the incoming observation by making it in-domain.

2 Proposed Method

2.1 Perception model

The perception model consists of a *ResNet-50* [4] that is tasked with processing the observation (o_t) obtained from an RGB camera, with the primary objective of comprehending the environmental context in which the robot operates. To learn consistent representations, we have opted for a design wherein the perception model compresses o_t into a consistent intermediate representation, z_t , which stays close to the corresponding BEV representation z'_t . Our choice for BEV representations is rooted in their capacity to convey the surrounding roadmaps with minimal information redundancy. To learn the representations from the binary BEV images, we train a *Variational Autoencoder (VAE)* to encode a binary BEV image x_t into $z'_t \in \mathbb{R}^{32}$. The loss function we optimize for training VAE is available in the supplementary material.

When training the perception model, we focus on 3 main principles. Firstly, irrespective of whether o_t originates from a simulator or the real-world environment, the output vector of the ResNet-50 should always be consistent by being close to the corresponding BEV vector. Secondly, BEV images must be represented in a continuous latent space that has smooth transitions to similar BEV images. Finally, the perception model (ResNet-50) must operate irrespective of any other decoder, i.e. the representations estimated by the encoder must be sufficient for performing the downstream task efficiently. Moreover, this would also allow for unsupervised training/fine-tuning of the ResNet-50 using real-world RGB sequences, which we leave for the future work.

We opt to train the ResNet-50 after initializing with ImageNet pre-trained weights on a large-scale dataset containing FPV-BEV image pairs captured in the simulator. To achieve the FPV-BEV translation, we align the output vector $z \in \mathbb{R}^{32}$ from the ResNet-50 with the corresponding BEV latent vector z' through *Contrastive Learning* (Fig 2). We optimize the model parameters with the cosine similarity-based contrastive loss and Mean Square Error (MSE) loss for image encoding. The detailed cost functions can be found in the supplementary material.

2.2 Temporal model with Robustness modules

To enhance the robustness of the perception model and transfer it to the real world setting, we implemented an additional model in the pipeline. Fig. 1 shows our proposed method of robustness enhancement. This involves the integration of an LSTM, functioning as a Memory model. The LSTM was trained on sequences $\{o_j, a_j\}_{j=0}^{j=T}$ gathered from sequences $\{\mathcal{T}_0, \mathcal{T}_1, \dots, \mathcal{T}_n\}$ in the simulator. The primary outcome of this Memory model is to effectively infuse historical context $\{\{z_j, a_j\}_{j=0}^{j=T}\}$

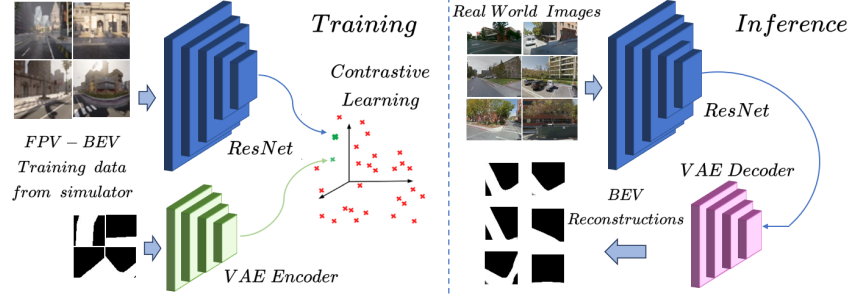


Figure 2: Training pipeline for the perception model. During the training phase, the Resnet model is trained using FPV (First person view) images from the simulator. During the test phase, its used for inferring the embeddings of real-world RGB images. These embedding are further passed through the VAE decoder to get BEV (Birds eye view) observations.

into the prediction of \hat{z}_t , which forms a candidate of z_t , and enhancing the robustness of the perception module when confronted with the unseen real-world data. To model the uncertainty in future states, we add an *Mixture Density Network (MDN)* on the top of LSTM output layer. The above pipeline can be formulated as:

$$\hat{z}_t \sim P(\hat{z}_t | a_{t-1}, \hat{z}_{t-1}, h_{t-1}) \quad (1)$$

where $a_{t-1}, \hat{z}_{t-1}, h_{t-1}$ respectively denotes action, state prediction at the previous timestep, and historical hidden state at the time step $t - 1$. \hat{z}_t is the latent representation that is given as an input to the policy. The detailed cost functions for training the Memory model (M) can be found in the supplementary material.

Nonetheless, it is noteworthy that z_t , that is obtained from the ResNet-50 may slightly distract from the latent distribution of BEV images when the perception model is applied to real-world observations o_t , potentially impacting the performance of the LSTM and the policy. To mitigate this concern, we collected a dataset \mathcal{S} comprising of the latent vectors s of 1439 BEV images which we define as the *BEV anchors*. In practice, upon obtaining the output vector z_t from the ResNet-50, we measure its proximity to each $s \in \mathcal{S}$, subsequently identifying the closest match. We replace z_t with the identified anchor embedding \bar{z}_t , ensuring that both the LSTM and the policy consistently uses the pre-defined BEV data distribution. We pass \bar{z}_t as an input to the LSTM, along with the previous action a_{t-1} to get the output \hat{z}_{t+1} . Again, we find the closest match $\hat{s}_t \in \mathcal{S}$ for \hat{z}_t . We call this module *Anchor State Checking (ASC)*:

$$\bar{z} = \arg \min_{s \in \mathcal{S}} \|z - s\| \quad (2)$$

We also utilize the LSTM model for rejecting erroneous predictions by the ResNet-50, further enhancing the system’s robustness against noise. If the processed prediction \bar{z}_t from the perception model is estimated with confidence score τ_t , obtained from either cosine-similarity or MSE, below a predefined threshold ρ , we deliberately discard \bar{z}_t and opt for \hat{z}_t . In such instances, we resort to the output of the LSTM at the previous time-step. This module is known as *Temporal State Checking (TSC)*:

$$\hat{z}_t = \begin{cases} \bar{z}_t, & \tau_t \geq \rho, \\ \hat{z}_{t-1}, & \tau_t < \rho. \end{cases} \quad (3)$$

Apart from adding robustness to the system using TSC, the utilization of the Memory model also serves as the crucial purpose of performing interpolation for the robots state in instances where actual observations o_t are delayed. This ensures the continuity and reliability of the entire system. There is often a notable discrepancy in the update frequencies between control signals and camera frames. Typically, control signals exhibit a significantly higher update rate compared to the incoming stream of camera frames.

	Class#1	Class#2	Class#3	Class#4	Class#5	Class#6
<i>Simulation Dataset</i>						
Baseline	31	33	29	31	34	22
Ours (Cosine Similarity)	58.1	75.8	69.0	96.8	29.4	68.2
Ours (MSE)	96.8	84.9	75.9	96.8	76.5	77.3
	90.3	93.9	82.8	90.3	82.4	81.8
<i>Real-World Dataset</i>						
Baseline	37	103	32	37	39	37
Ours (Cosine Similarity)	47.0	43.8	54.8	91.4	9.3	31.8
Ours (MSE)	91.9	65.0	53.1	83.8	35.9	48.6
	89.2	69.9	59.4	91.9	35.9	56.8

Figure 3: **Evaluation on the Validation dataset.** Left: We constructed two 6-class validation datasets: one from the simulator (second row) and another from street-view data (third row). The first row depicts the Bird’s eye view (BEV) of the corresponding RGB images, which forms the basis for each class. Right: We compared our two contrastive learning approaches (MSE-based and Cosine similarity-based) against a 6-class CNN classifier. Our methods outperformed the baseline on both the unseen simulation dataset and the real-world validation dataset as shown above.

	Class#1		Class#2		Class#3		Class#4		Class#5		Class#6													
	#	ACC MSE CE	#	ACC MSE CE	#	ACC MSE CE	#	ACC MSE CE	#	ACC MSE CE	#	ACC MSE CE												
Baseline	860	47	0.17	2.12	1378	39	0.29	6.58	1506	82	0.2	3.45	1106	81	0.18	1.95	10	0	0.13	2.89	140	0	0.22	2.16
Ours (-LSTM)	860	22	0.16	0.71	1378	81	0.14	0.45	1506	64	0.12	0.72	1106	73	0.13	0.49	10	0	0.17	2.19	140	42	0.22	1.49
Baseline	657	33	0.23	1.38	1853	64	0.34	9.07	1214	85	0.22	4.84	326	79	0.39	10.47	640	19	0.46	11.25	310	21	0.4	9.06
Ours (-LSTM)	657	52	0.11	0.80	1853	75	0.13	1.37	1214	42	0.15	2.16	326	76	0.14	1.62	640	58	0.35	6.29	310	28	0.31	3.93
Baseline	1644	20	0.21	1.54	1287	49	0.31	6.36	1003	57	0.18	2.74	973	94	0.18	2.24	0	-	-	-	93	0	0.19	1.65
Ours (-LSTM)	1644	22	0.21	0.74	1287	71	0.13	0.44	1003	51	0.09	0.60	973	78	0.12	0.74	0	-	-	-	93	15	0.15	1.70
Baseline	679	33	0.21	0.13	2268	69	0.30	0.74	846	88	0.22	0.38	1087	90	0.19	0.24	63	20	0.30	0.91	57	10	0.34	0.63
Ours (-LSTM)	679	61	0.10	0.07	2268	94	0.08	0.07	846	58	0.07	0.08	1087	50	0.19	0.06	63	79	0.11	0.13	57	45	0.07	0.16
Ours (-LSTM)	35	3	0.30	0.50	1079	73	0.40	0.81	69	28	0.30	0.90	131	91	0.30	0.42	0	-	-	-	177	7	0.40	5.21
Ours (+LSTM)	35	77	0.18	0.32	1079	78	0.30	1.13	69	81	0.21	0.77	131	88	0.23	0.77	0	-	-	-	177	50	0.41	1.16
Ours (-LSTM)	50	44	0.17	3.24	1511	70	0.32	4.38	151	28	0.20	0.60	198	37	0.30	1.40	0	-	-	0	-	-	-	-
Ours (+LSTM)	50	90	0.14	3.89	1511	69	0.29	4.21	151	60	0.19	0.69	198	84	0.18	0.44	0	-	-	0	-	-	-	-
Ours (-LSTM)	253	81	0.10	0.70	1914	89	0.33	4.49	33	12	0.20	1.00	190	42.6	0.20	0.90	0	-	-	0	-	-	-	-
Ours (+LSTM)	253	97	0.13	0.50	1914	84	0.17	2.40	33	88	0.17	0.85	190	88	0.19	0.70	0	-	-	0	-	-	-	-
Ours (-LSTM)	0	-	-	-	974	83	0.32	4.42	17	29	0.26	5.22	0	-	-	-	88	90.9	0.24	4.95	78	26	0.37	9.83
Ours (+LSTM)	0	-	-	-	974	83	0.29	3.99	17	94	0.19	3.34	0	-	-	-	88	92.0	0.23	4.62	78	59	0.19	3.42

Figure 4: **Experiments on the Test Dataset** Each double-row corresponds to a data sequence. In the upper portion of the table, we assessed our method independently of the LSTM on an unseen temporal sequence from the simulator, contrasting it with the baseline CNN classifier. In the lower portion, we compared the performance of system with and without LSTM on a real-world data sequence. Note that dashes in the table indicate the absence of a class in the respective sequence.

3 Evaluation and Results

We evaluated the performance of our ResNet-50 model using the Validation dataset and the results are shown in Figure 3. Our perception model identifies the closest matching class for the output embedding. The baseline is a ResNet-50 model trained on a 6-class training dataset comprising 140,213 labelled FPV images.

Following a similar approach, we used the Test dataset to evaluate the entire system. Apart from the accuracy also used Cross entropy (CE) and Mean Square error (MSE) to judge the quality of reconstructions by the LSTM model. These results are shown in Figure 4. The metrics presented in this table exhibit a slight decrease compared to Figure 3. This can be attributed to the increased presence of abnormal observations and higher ambiguity between classes within the time-series data obtained from the robot, as opposed to the manually collected and labelled dataset in the validation dataset. For additional details regarding the experiments, please refer to the supplementary material.

References

[1] Karol Arndt et al. “Meta Reinforcement Learning for Sim-to-real Domain Adaptation”. In: *CoRR* abs/1909.12906 (2019). arXiv: 1909. 12906. URL: <http://arxiv.org/abs/1909.12906>.

- [2] Alexey Dosovitskiy et al. “CARLA: An Open Urban Driving Simulator”. In: *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*. Vol. 78. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1–16. URL: <http://proceedings.mlr.press/v78/dosovitskiy17a.html>.
- [3] David Ha and Jürgen Schmidhuber. “World Models”. In: *CoRR* abs/1803.10122 (2018). arXiv: 1803.10122. URL: <http://arxiv.org/abs/1803.10122>.
- [4] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: <http://arxiv.org/abs/1512.03385>.
- [5] Josip Josifovski et al. “Analysis of Randomization Effects on Sim2Real Transfer in Reinforcement Learning for Robotic Manipulation Tasks”. In: *arXiv e-prints* (June 2022). DOI: 10.48550. arXiv: 2206.06282 [cs.R0].
- [6] Manuel Kaspar, Juan David Muñoz Osorio, and Jürgen Bock. “Sim2Real Transfer for Reinforcement Learning without Dynamics Randomization”. In: *arXiv e-prints*, arXiv:2002.11635 (Feb. 2020). DOI: 10.48550. arXiv: 2002.11635 [cs.AI].
- [7] Eric Liang et al. “Ray RLLib: A Composable and Scalable Reinforcement Learning Library”. In: *CoRR* abs/1712.09381 (2017). arXiv: 1712.09381. URL: <http://arxiv.org/abs/1712.09381>.
- [8] Lennart Luttkus, Peter Krönes, and Lars Mikelsons. “Scoomatic: Simulation and Validation of a Semi-Autonomous Individual Last-Mile Vehicle”. In: *Sechste IFToMM D-A-CH Konferenz 2020: 27./28. Februar 2020, Campus Technik Lienz*. Vol. 2020. Feb. 21, 2020. DOI: 10.17185/dupublico/71204. URL: <https://nbn-resolving.org/urn:nbn:de:hbz:464-20200221-092453-2>.
- [9] Fereshteh Sadeghi and Sergey Levine. “CAD2RL: Real Single-Image Flight Without a Single Real Image”. In: *Robotics: Science and Systems XIII, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, July 12-16, 2017*. Ed. by Nancy M. Amato et al. 2017. DOI: 10.15607/RSS.2017.XIII.034. URL: <http://www.roboticsproceedings.org/rss13/p34.html>.
- [10] John Schulman et al. “Proximal Policy Optimization Algorithms”. In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: <http://arxiv.org/abs/1707.06347>.
- [11] Gregory J. Stein and Nicholas Roy. “GeneSIS-RT: Generating Synthetic Images for training Secondary Real-world Tasks”. In: *CoRR* abs/1710.04280 (2017). arXiv: 1710.04280. URL: <http://arxiv.org/abs/1710.04280>.
- [12] Jingwei Zhang et al. “VR-Goggles for Robots: Real-to-Sim Domain Adaptation for Visual Control”. In: *IEEE Robotics Autom. Lett.* 4.2 (2019), pp. 1148–1155. DOI: 10.1109/LRA.2019.2894216. URL: <https://doi.org/10.1109/LRA.2019.2894216>.

Appendices

In this supplementary material, we offer further context and elaborate on details that were not included in the paper *World Model Based Sim2Real Transfer for Visual Navigation*.

A Loss Function

To learn the representations from the binary BEV images, we train a *Variational Autoencoder (VAE)* to encode a binary BEV image x_t into $z'_t \in \mathbb{R}^{32}$. and optimize the following loss function:

$$\mathcal{L}_{VAE} = \sum_{i=0}^B -(y_i \cdot \log(x_i) + (1 - y_i) \cdot \log(1 - x_i)) + \beta \cdot KL(\mathcal{N}(\mu_i, \sigma_i^2) || \mathcal{N}(0, 1)) \quad (1)$$

In the above loss function KL divergence is the *Kullback Leibler divergence*, y is the ground truth for reconstruction and B is the batch size. Using this loss function, the VAE Encoder will learn to embed the BEV observations x_i into a smooth Gaussian manifold that allows 2 BEV observations that are very similar, for example 2 straight roads, but a have slight variation in the angle to be closer, than a straight road and an intersection. These are the BEV embeddings the corresponding RGB images needs to be close with.

We optimize the parameters of the perception model V with the cosine similarity-based contrastive loss for image encoding.

$$\begin{aligned} \mathcal{L}_{BEV}^i &= -\log \frac{\exp(z_i \cdot z'_i / \tau)}{\sum_{j=0}^N \exp(z_i \cdot z'_j / \tau)} \\ \mathcal{L}_{FPV}^i &= -\log \frac{\exp(z'_i \cdot z_i / \tau)}{\sum_{j=0}^N \exp(z'_i \cdot z_j / \tau)} \\ \mathcal{L}_{contrastive} &= \sum_{i=0}^N (\mathcal{L}_{BEV}^i + \mathcal{L}_{FPV}^i) / 2 \end{aligned} \quad (2)$$

where τ is a temperature hyper-parameter, and N denotes the training batch size. Apart from the cosine similarity-based contrastive loss, we also attempt to use the Mean Square Error (MSE) loss function:

$$\mathcal{L}_{contrastive} = \|z - z'\|_2 \quad (3)$$

We optimize M with the below loss function:

$$\mathcal{L}_M = -\frac{1}{T} \sum_{t=1}^T \log \left(\sum_{j=1}^K \theta_j \cdot \mathcal{N}(z_t | \mu_j, \sigma_j) \right) \quad (4)$$

where $\{T, K, \theta_j, \mathcal{N}(z_t | \mu_j, \sigma_j)\}$ is, respectively, the training batch size, number of Gaussian models, Gaussian mixture weights with the constraint $\sum_{j=1}^K \theta_j = 1$, and the probability of ground truth at time step t conditioned on predicted mean μ_j and standard variance σ_j for Gaussian model j .

B Control model

To speedup RL for the control model, we intentionally allocate the majority of challenges to other components in the pipeline. Consequently, the training of our policy becomes a relatively straightforward endeavour. We accomplished this by training a policy employing the PPO algorithm [10]. The design of the reward function is rooted in proportionality to the number of waypoints the robot achieves to the designated goal point. In each timestep, the policy receives the current embedding of the observation z_t concatenated with the directional vector pointing towards the waypoint tasked with producing a pair of (Throttle, Steer) values.

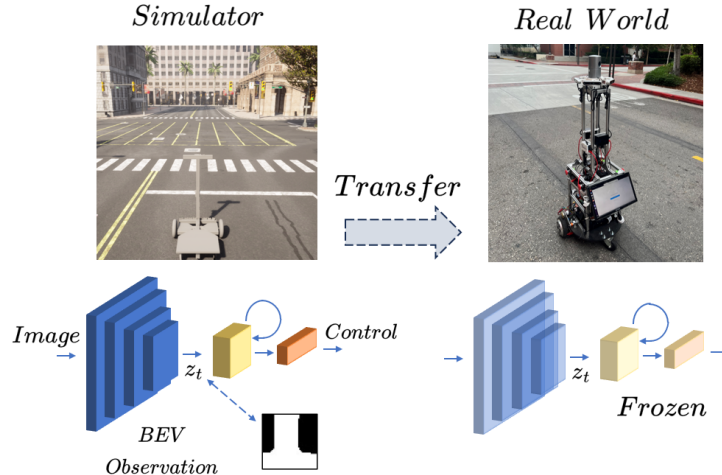


Figure 1: **Overview of our system** We first train the visual navigation system on a large-scale dataset collected in the simulator and deploy the frozen model in an unseen real-world environment.

C Experimental platform and setup

We collected the train dataset from the CARLA simulator to train both the Perception and the Memory model. Along with that, we also collected the validation and the test datasets from 2 different real-world sources. Following are the details on the collected datasets.

C.1 Data collection

C.1.1 Train dataset from CARLA simulator

	# of FPV-BEV Pairs	# of Sequences
Town01	201247	609
Town02	100206	301
Town03	202242	633
Town04	201784	385
Town05	100404	202
Town06	201333	400
Town07	201960	478
Town10HD	201332	317
Total	1410508	3325

Table 1: Statistics of the training dataset.

Within the CARLA simulator, we have access to the global waypoints various trajectories. We added multi-camera system on the robot to obtain the data. To allow some uncertainty, we randomly sampled a range of different orientations and locations. Leveraging this setup, we facilitated the generation of a large dataset of FPV-BEV images. We augmented the simulator’s realism by introducing weather randomization and non-player traffic into the simulated environment. C.1.1 presents the statistics of trajectories sampled across eight distinct CARLA simulator maps. These trajectories are initiated from randomized start points and conclude upon either the successful completion of a task or failure. Along these trajectories, corresponding FPV and BEV images are concurrently captured. The FPV images are captured in a size of $84 \times 84 \times 3$, while the BEV images are acquired at a size of $64 \times 64 \times 1$.

C.1.2 Validation dataset from Google Street View

Using the Google Street View API, we obtained all the panoramic images from various locations on the USC campus. The panoramic images were segmented with a Horizontal Field of View (FoV) of 90 degrees and are manually segregated into different 6 different classes. The validation dataset does not have any temporal sequencing and is primarily focused on having a broader data distribution. Due to these reasons, this dataset becomes an optimal choice for evaluating the perception model.

C.1.3 Test dataset from Beobotv3

To evaluate the quality of representations estimated by the entire system, we record a video sequence using a mobile robot. More precisely, we recorded a set of 5 *ROS Bag* sequences at different locations of the USC campus. Later, we labelled all the frames in a *ROS Bag* sequence, similar to the above paragraph. However, unlike the validation set, the test dataset has temporal continuity, which would help us to judge the system better.

C.2 Experimental platform

C.2.1 Scoomatic in CARLA

The CARLA simulator had been primarily tailored to self-driving applications, with a specific focus on synchronous-drive vehicles. Since all the vehicles in CARLA uses *Ackermann steering*, we further developed an existing differential drive setup using *Schoomatic* [8] and upgraded the CARLA simulator. We find this necessary because our real-world hardware system is based on differential-drive and to enable seamless transfer without any fine-tuning, both the control systems need to have similar dynamics. Consequently, it exhibited a conspicuous deficiency in accommodating differential-drive agents. In response to this limitation, Luttkus [8] designed a scheme for the integration of a differential-drive robot into the CARLA environment. Building upon their work, we undertook the development of a dedicated simulator catering to differential-drive robots, subsequently migrating it into the newly introduced CARLA *v0.9.13*.

Our enhancements extended beyond mere implementation, including improvements to the stability of motion through the fine-tuning of its physical configuration. Additionally, we implemented a sensor designed to respond to collisions with curbs, in order to keep the robot running on roads. To ensure accessibility and ease of utilization, we have thoughtfully encapsulated the simulator within a readily deployable Docker image, which is shared for public use.

C.2.2 Beobotv3

For evaluating Zero-shot Sim2Real transfer, we built a hardware apparatus which is a *Non-Holonomic, Differential-drive* robot (*Beobotv3*) for the task of visual navigation. Our system is implemented using the *ROS (Robotic Operating System)* middleware and uses a *Coral EdgeTPU*, which is an *ASIC* chip designed to run CNN models for edge computing for all the compute. We used this particularly to run the forward inference of the ResNet-50 [4] through a *ROS* nodes. The models are trained using the *PyTorch* framework and are integrated with the *ROS* API for inference. We integrated the *RLLib* [7] framework with *Pytorch* and CARLA to train the policy using RL.

C.3 Codebase for the system

As part of our open-source framework, we are realising 3 repositories, that allow users to integrate our models and system into their framework. Following are the brief outlines for each of them:

C.3.1 Model pretraining

The code pertaining to this repository allows the users to train CNN and LSTM parts of our system on offline trajectory sequences using our method.

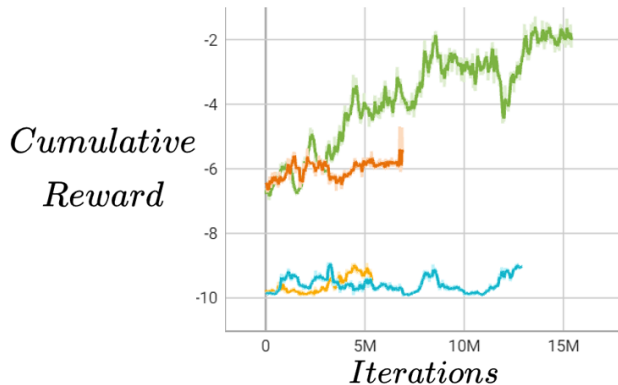


Figure 2: **RL experiments using BEV representations.** We notice that the BEV representations enable the agent to learn faster when the waypoint threshold is higher. Green and orange correspond to waypoint threshold of 5m, resolution of 1. Blue and Yellow corresponds to waypoint threshold of 1m, resolution of 1.

C.3.2 RL training

With our efficient implementation of CARLA API with RLlib integration, we were able to achieve simulator training at *1000 FPS*. With the objective of enabling distributed policy training, we integrated our compiled *Scoomatic* docker-based server.

C.3.3 System deployment

We also release ROS based code that allows users to integrate the above modules on their real-world robot. This specifically consists of ROS based APIs to load and infer the Pytorch models using the subscribed topics.

Apart from the above repositories, we also release a large dataset of FPV-BEV trajectory sequences collected by the *Schoomatic* robot in the CARLA simulator along with the pre-trained models.

D Additional Experiment

We performed RL experiments by deploying the policy trained in the CARLA simulator, which can be found here 2. We used the BEV representations to learn to navigate to a goal location with different waypoint density and waypoint threshold (threshold from the waypoint to be for obtaining the reward). We noticed that BEV representations tend to learn faster when the threshold is higher.

E Discussion and Conclusion

In this paper we proposed a robust navigation system that is trained entirely in a simulator and frozen when deployed. We learn compact embeddings of an RGB image for Visual Navigation that are aligned with those of corresponding BEV images. By decoupling the perception model from the control model, we get an added advantage of being able to deploy the model onto a future robot with unknown dynamics. This approach also allows us to pretrain the perception and the memory model using offline datasets. Lastly, we release software and models for efficient training and deployment of the model onto a real world robot.