FloorGenT: Generative Vector Graphic Model of Floor Plans for Robotics

Ludvig Ericson, Patric Jensfelt

***Abstract*— Floor plans are the basis of reasoning in and communicating about indoor environments. In this paper, we show that by modelling floor plans as sequences of line segments seen from a particular point of view, recent advances in autoregressive sequence modelling can be leveraged to model and predict floor plans. The line segments are canonicalized and translated to sequence of tokens and an attention-based neural network is used to fit a one-step distribution over next tokens. We fit the network to sequences derived from a set of large-scale floor plans, and demonstrate the capabilities of the model in four scenarios: novel floor plan generation, completion of partially observed floor plans, generation of floor plans from simulated sensor data, and finally, the applicability of a floor plan model in predicting the shortest distance with partial knowledge of the environment.**

I. Introduction

Reasoning and planning in indoor environments is an important capability in the context of robotics. One approach to this problem is through modelling floor plans. Floor plans are a simplified and minimalist map of an indoor environment, and can be used for both reasoning and communicating about such environments. In this paper, the aim is to predict floor plans based on environmental cues in order to facilitate robotics tasks in unknown environments, such as autonomous exploration and search-and-rescue. To this end, we construct a generative model that encodes the implicit rules of floor plans without explicitly stating those rules, e.g., that walls join at right angles, rooms are symmetrical, and doorways join rooms.

Predicting floor plans from environmental cues is not a novel idea, and it has previously been cast as an image in-painting problem where an image model, typically a convolutional neural network, is used to fill in the missing regions of a rasterized floor plan. However, convolutions imply an inductive bias towards local dependencies in pixel space which is not necessarily a good fit for floor plans, which often exhibit non-local correlations. Furthermore, floor plan rasterizations are unlike natural images consisting exclusively of high-frequency content, making them particularly prone to artifacts such as bleeding and blurring. Care must be taken to prevent such effects as shown in [1].

By modelling floor plans as vector graphics, this issue is avoided. It also lets us leverage recent advances in language models, such as attention-based neural network architectures.

Paper accepted for the 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2022). All authors are with the Division of Robotics, Perception, and Learning at KTH Royal Institute of Technology, Stockholm, SE-11428, Sweden. This work was financed by the Swedish Research Council grant XPLORE3D. For e-mail correspondence, contact `ludv@kth.se`.
978-1-5386-5541-2/18/$31.00

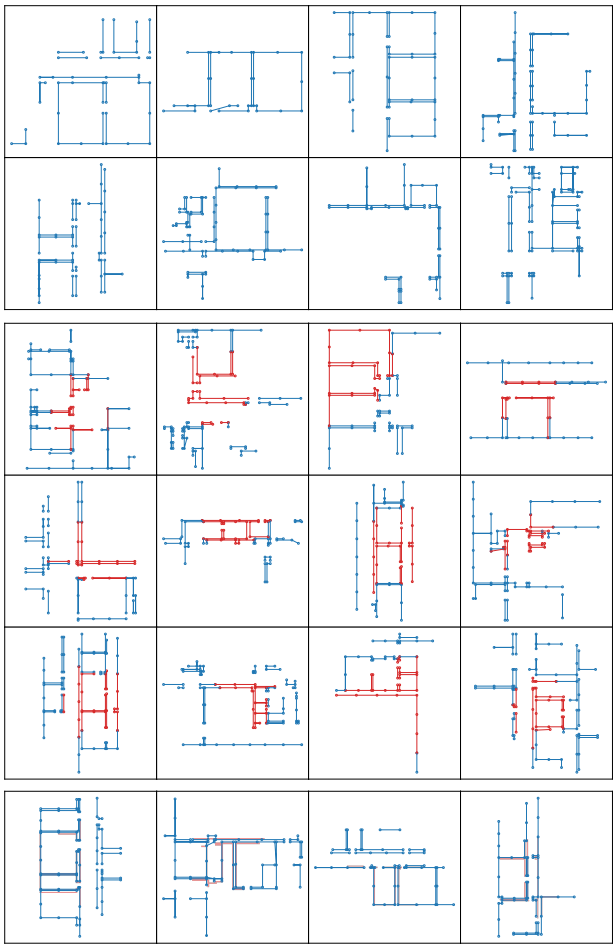

Fig. 1. Randomly picked samples of a FloorGenT network trained on the KTH floor plan dataset generated with nucleus sampling ($p = 90\,\%$). Blue is sampled model output. **Top**: unconditioned novel samples. **Middle**: partial sequence completion samples conditioned on the segments shown in red (first 25 segments of randomly selected test sequences, i.e., novel data to the network). **Bottom**: partial image conditioned samples with the input image shown in red (rasterization of the first 25 segments of randomly selected test sequences).

In an autoregressive sequence formulation such as [2], [3], the model takes its own previous output into account at each step as it generates an output sequence. This approach has shown remarkable results in modelling and generating various sequences, including but not limited to natural languages. By casting floor plans as sequences, similar performance could be obtained in the floor plan domain, and leveraged in a robotics context.

In this paper we present such an implicit model of indoor environments, and demonstrate its usefulness in a typical

robotics tasks. In summary, the contributions of this paper are:

1) A method of representing floor plans as a sequence of ordered line segments seen from a particular point of view within the floor plan, and a procedure to canonicalize the sequences.
2) A attention-based generative model tailored to dealing with such sequences of line segments.
3) An evaluation of the performance in generating diverse novel floor plan sequences, sequence completions from partial sequences, and sequence generation from partial birds-eye view raster images.
4) A demonstration of the model's abilities in solving a typical robotics task, namely path planning in a partially observed environment.

The network architecture and related code is accessible at `github.com/lericson/FloorGenT/`, and an online demonstration is available at `lericson.se/floorgent/demo/`.

II. Related Work

Floor plans as the basis of robotic planning and reasoning has been explored before. In the category of predicting maps, approaches range from explicit to implicit. A recent example of an explicit method is [4], in which an algorithm for predicting the layouts of partially observed rooms in indoor environments is proposed, by constructing an *explicit* set of rules and assumptions about those environments in the algorithm itself. A majority of recent work is based on deep learning, learning the rules from data rather than by design from an expert. The present work belongs to this latter *implicit* category, but differs in a key regard: previous work is largely based on the convolutional neural networks from computer vision, such as [5], [6]. We instead cast the problem as modelling a sequence of line drawing instructions.

Contrastingly, [7] use a *topological* rather than metric representation, e.g., "offices connect to kitchens through corridors", which is then used for symbolic reasoning and planning. In pursuit of this, the *KTH floor plan dataset* was constructed, consisting vector graphics of $38\,000$ rooms in 184 floor plans from 27 different university buildings. We use the same dataset, though not for its topological information. It is important to note that these floor plans are large, with hundreds of rooms per floor.

Separately from the robotics-aimed line of work is what is becoming known as *inverse CAD*, where the goal is mapping images or pointclouds to architectural blueprints. For example, [8] propose a method to produce a vector graphics floor plan from a set of RGBD sensor scans that fully cover an apartment. In [1], a method is proposed to classify room types (e.g., kitchen, bathroom, bedroom) from rasterized floor plan drawings. Both are based on convolutional neural networks. Inverse CAD is in general aimed at realtors and architects, rather than robotics.

Though not directed at solving robotics tasks, recent work has also been aimed at modelling vector graphics generally. For example, [9], [10] show that vector graphics can be embedded in a latent space of fixed dimensionality. The embeddings then have some spatial relationship to each other, such that similar drawings are nearby in that space. Neither work is aimed at prediction from partial inputs.

In terms of network architecture, we leverage recent developments in sequence modelling for natural languages in a similar style to attention-based models such as [3], [11]. In [12], a method is proposed to model 3D meshes as sets of polygons through the use of attention-based sequence models as proposed in [13]. Our model is built on that same foundation.

III. FloorGenT

We model the distribution over floor plans $\mathcal{F}$ by translating them to sequences of line drawing instructions, encoded as a *token sequence* $\boldsymbol{t}$ and factorizing in autoregressive manner:

$$p(\mathcal{F}) \triangleq p(\boldsymbol{t}) = \prod_{i=1}^{k} p(t_i | \boldsymbol{t}_{<i}) \tag{1}$$

where t_i is the ith token in the sequence, and $\boldsymbol{t}_{<i}$ are the tokens before i. The tokens combine into triplets (c, x, y) to form the line drawing instructions, consisting of the opcode $c \in \{\texttt{stop}, \texttt{move}, \texttt{line}\}$ meaning "end of sequence", "move to coordinate", and "draw line to coordinate" respectively, and the operands are the discrete coordinates $x, y \in \{q_1, q_2, \ldots, q_{N_Q}\}$. We denote the set of tokens

$$\mathcal{T} = \{\texttt{stop}, \texttt{move}, \texttt{line}, q_1, q_2, \ldots, q_{N_Q}\} \tag{2}$$

Each line segment is encoded as two triplets, a `move` followed by a `line`, illustrated in Fig. 3. This representation allows manipulating sequences at line segment level, with the drawback that it is redundant when a line segment is joined with the previous segment. This is not typical in our case, as about $12.4\,\%$ segment pairs are so joined.

A. Token Sequence Model

With the sequence representation defined, we turn to modelling the distribution over sequences defined in (1). We use the Transformer decoder architecture with *multi-head scaled dot product attention* as in [13]. These networks were originally proposed to model natural languages, and our token sequences can similarly be thought as a synthetic language with its semantics defined primarily by the resulting vector graphic, making it necessary to "read between the lines".

The architecture is presented in Appendix A. The output of the model is a categorical distribution $p(t_i | \boldsymbol{t}_{<i}; \theta)$, where θ denotes the parameters of the model. The loss $\mathcal{L}$ is the negative log likelihood of the ground truth token values $\hat{t}$ under the model's induced distribution,

$$\mathcal{L}(\hat{t}, \theta) = \sum_i \log p(t_i = \hat{t}_i \,|\, \boldsymbol{t}_{<i} = \hat{\boldsymbol{t}}_{<i}; \theta) \tag{3}$$

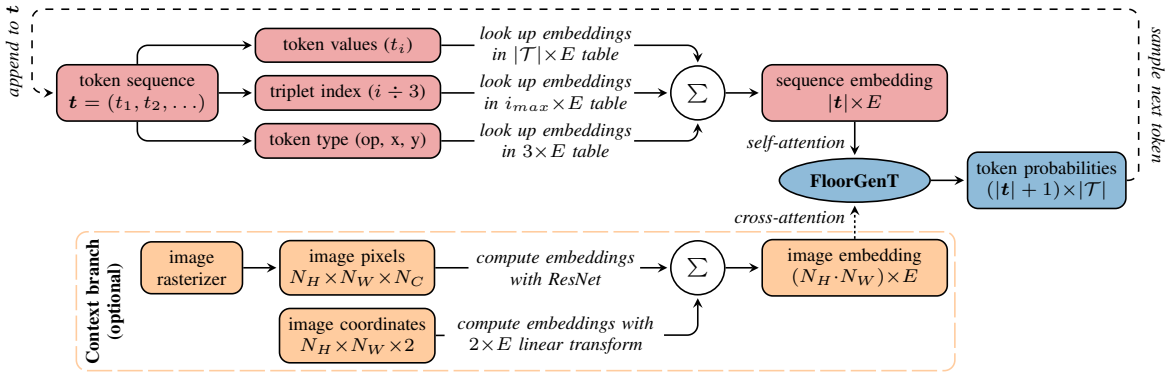

Fig. 2. Overview of data flow in FloorGenT. The input sequence is a possibly empty sequence of tokens $\boldsymbol{t}$. Each token is embedded as a sum of three discrete embedding vectors. The embedded tokens are the inputs of the first self-attention layer, and later layers take the previous layer's output as their input. In image models, the input image pixel values is embedded through a convolutional neural network, and are embedded along with their respective coordinates through a linear transformation. The per-pixel embeddings are then input to the cross-attention layers. When sampling, the next token is repeatedly drawn from the next token distribution, and fed back into the network at the end of the token sequence. E is the number of embedding dimensions.

$$\boldsymbol{t} = \begin{pmatrix} \texttt{move}, & q_1, & q_4, & \texttt{line}, & q_1, & q_1, \\ \texttt{move}, & q_1, & q_1, & \texttt{line}, & q_3, & q_1, \\ \texttt{move}, & q_3, & q_2, & \texttt{line}, & q_2, & q_2, \\ \texttt{move}, & q_2, & q_2, & \texttt{line}, & q_2, & q_4, \end{pmatrix} \rightarrow$$

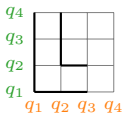

Fig. 3. An example of a token sequence and its corresponding drawing in the shape of an L. Note that in practice, the line segments would be sorted by their distance to some origin coordinate as described in Section III-D.

B. Embeddings

We use discrete embedding vectors for the token sequence as illustrated in Fig. 2. Though it is possible to embed each vertex (x, y) with a single token by summing or concatenating each coordinate's embedding, [12] reported that this reduced performance significantly, and we therefore keep them embedded as separate tokens. It is also possible to remove the opcode tokens, however, [14] suggest that separator tokens allow the network to better propagate information between attention heads.

Some variants of our model take images as inputs, and these are embedded by feeding the pixel values through an off-the-shelf convolutional neural network, together with pixel coordinate embeddings from a learned linear transformation of the pixel coordinates. This is illustrated as an optional context branch in Fig. 2. We evaluate two types of image embedding networks: first, as in [12], a pre-activation style ResNet [15], and second, the more recently proposed MLP Mixer [16]. Neither network is pretrained.

C. Turning Floor Plans into Token Sequences

In the KTH floor plan dataset, a floor plan $\mathcal{F}$ is defined as a set of *spaces*, with each space having a bounding polygon defined as a sequence of line segments. Each segment has a type, e.g., wall, window, or "portal", i.e., a connection to another space. For our purposes, we treat windows as walls, and portals are ignored.

We model an *in situ* agent partially observing an environment with some sensor, e.g., a LIDAR, at some location in the environment. This lets us target on-line estimation scenarios such as autonomous exploration or search-and-rescue. It also means the dataset is "over-sampled", in the sense that we generate many training samples from a single floor plan.

We sample *valid* locations inside a floor plan by rejection sampling N_P points $\mathcal{P}$ by sampling uniformly over the floor plan's bounding box until each sample are inside a space by some minimum clearance. We then maximize the distance between the *selected* points $\mathcal{S} \subset \mathcal{P}$ by iteratively constructing $\mathcal{S}_i = \mathcal{S}_{i-1} \cup \{\boldsymbol{s}_i\}$ from the selected point $\boldsymbol{s}_i$ at each iteration where

$$\boldsymbol{s}_i = \underset{\boldsymbol{p} \in \mathcal{P} \setminus \mathcal{S}_{i-1}}{\arg\max} \left(\min_{\boldsymbol{p}' \in \mathcal{S}_{i-1}} \|\boldsymbol{p} - \boldsymbol{p}'\|_2 \right) \tag{4}$$

until $|\boldsymbol{s}_i - \boldsymbol{s}_{i-1}| < d_{pmin}$. We initialize with $\mathcal{S}_1 = \{\boldsymbol{p}_1\}$ without loss of generality as all $\boldsymbol{p}_i$ are independent and identically distributed.

D. Canonicalization

To remove extraneous degrees of freedom, we canonicalize the line segments so the set of token sequences that produce identical visual result is as small as possible. First, let the canonical order of endpoints of a line segment from (x_0, y_0) to (x_1, y_1) be such that $x_0 < x_1$, and $y_0 < y_1$ in the case when $x_0 = x_1$. In other words, the segments are always oriented left-to-right, or bottom-to-top for vertical lines.

The line segments are centered and scaled by a global scale factor, determined by the dataset statistics, before being quantized. We extend the segments by considering any two overlapping parallel lines, then extending them to each other's furthest endpoints, resulting in one longer line. This yields a representation where each unbroken line segment is represented by only one actual segment. We then break each line segments at each intersection with another segment, so that no segments cross in their interior. Finally, we subdivide each segment so the longest segment length is bounded.

TABLE I
EVALUATION AND COMPARISON OF SELECTED MODELS

Model	NLL (bits)	Top-1	Top-5
Uniform	8.02	0.4 %	1.9 %
Nearest Neighbor	——	59.7 %	63.3 %
Equivalent MLP	1.87	68.2 %	84.3 %
FloorGenT	1.09	81.4 %	91.8 %
- Opcode Tokens	1.12	81.0 %	91.5 %
- Position Embed.	1.34	77.8 %	90.0 %
FloorGenT Images	0.83	84.3 %	95.0 %
- Position Embed.	0.99	81.7 %	93.5 %
+ All Segm., ResNet	0.50	89.5 %	98.2 %
+ All Segm., MLP Mixer	0.40	91.4 %	98.9 %

E. Partial Sequences

For each sample location s, the corresponding floor plan's line segments are ordered by their distance to s, and only the first (nearest) N_{segs} line segments are kept. The distance is computed as the Euclidean distance between the location and the closest point on the line segment. The segments are translated so that the point-of-view location is at the origin. Finally, only the N_{segs} nearest segments are kept and mapped to a sequence of tokens by inserting the appropriate opcodes.

F. Image Rasterization

A line drawing algorithm is used to rasterize the first $N_{raster} \ll N_{segs}$ segments into a black-and-white bitmap to approximate a partial birds-eye-view occupancy grid. We do not use antialiasing.

IV. EXPERIMENTS

In this section, we present both quantitative metrics and qualitative demonstrations of the model's output in different settings. In all experiments, the quantization level N_Q is 256, the sequence is at most N_{segs} = 100 segments, each segment is at most 2.5 m long, the nearest segment is at least $40\,\mathrm{cm}$ away, and the furthest segment is at most 7.5 m away.

A. Network Architecture and Training Details

We use six attention layers, E is 512, and N_{heads} is 8. See Appendix A. The models are trained with the Adam [17] optimizer on the KTH floor plan dataset, divided into a 90–10 split with a learning rate of $3 \cdot 10^{-4}$. The dropout rate was set to $60\,\%$ for all models. Each model was trained over 10^6 batches of size N_B = 8, after which the optimization has converged. All layers use ReZero [18], this accelerates convergence and allows deeper network architectures.

1) Data Augmentation: Each floor plans is sampled at high density with multiple overlapping sequences as a form of data augmentation, which also preserves invariants imposed by the canonicalization such as segment order. We then mirror and rotate by a multiple of $90°$ at random using a *signed permutation*. There are exactly eight such signed permutation matrices, generated by enumerating all combinations of three primitive operations: mirror X axis, mirror Y axis, and swap X and Y axes.

2) Train-Test Split: To avoid test data leaking into the training set, we split training and test before shuffling the set of sampled sequences. This ensures that a given floor plan is only in one of the splits. Since a building's floors can be similar or even identical, care must also be taken to ensure that a given *building* is only in one of the splits. In our case, the floor plans are ordered by the building they belong to, so such leakage is prevented by the same mechanism.

B. Predictive Performance on Test Set

The KTH floor plan dataset test set contains 2.46 million tokens with an average sequence length of 363 tokens. Likelihood and accuracy metrics of the models and variants described below are presented in Table I. Negative log likelihood (NLL) is reported on the test set as mean bits per token. Top-k accuracy is computed as the frequency with which the ground truth token value is in the k most likely tokens predicted by each model, with sequences from the test set. Top-1 is equivalent to the maximum likelihood estimate.

1) Uniform: We report a worst case reference point with uniform probabilities. The NLL is $\log|\mathcal{T}|$, and the top-k accuracy follows a hypergeometric distribution.

2) Nearest Neighbors: Predictions are formed by finding the N_k most similar subsequences in the training set, using the sliding window approach outlined in Appendix B to construct subsequences of N_w tokens. The most common following token is then the prediction. Sequence similarity is measured by Hamming distance, i.e., the number of unequal tokens. We report performance with $N_w = 10$ and $N_k = 32$, tuned by hyperparameter search and cross-validated. NLL is not reported as the model is deterministic.

3) Equivalent MLP: We ablate the attention mechanism entirely, turning the network into a multi-layer perceptron (MLP), refer to Appendix B for architecture details. Notably, this uses nearly four times more network parameters. The MLP had the best performance of the baselines we measured, though not as good as its attention-based counterparts.

4) FloorGenT: This is the standard formulation of the model as described in Section III.

5) Opcode Tokens: We evaluate a model that is only given coordinate tokens and the `stop` opcode token in its input. We let $p(t_i = \hat{t}_i) = 1$ for the removed opcode tokens, since the parity of the triplet index is sufficient to determine where `move` and `line` opcodes should be inserted. We find that performance is largely unchanged from the standard formulation, though inference time is shorter since a third of the tokens have been removed.

6) Position Embeddings: Line segments are the same regardless of their position in the token sequence, so their representation should arguably also be position invariant. We follow [19] and remove the position embeddings, so the model is unable to rely on statistics of the sequence position. We find that though this delayed overfitting somewhat, dropout was still necessary. The accuracy and NLL was unsurprisingly worse, as it is not possible to know which token was last (or anywhere) in the sequence without position embeddings, and it is therefore more difficult to predict what

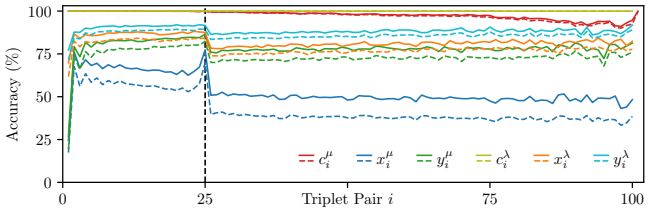

Fig. 4. Predicted probability (dashed lines) versus empirical accuracy (solid lines) per token position on the test set. The six plots correspond to token types when grouped into `move-line` triplet pairs denoted μ and λ. Each point on the horizontal axis thus corresponds to a line segment. These results are from the partial image model, and the dashed vertical line separates segments visible in the input image from those that are not. The plot for c_i^μ extends one index longer since it includes the final `stop` token.

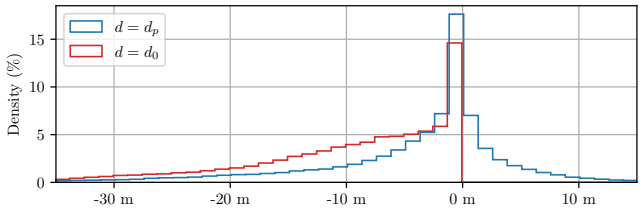

Fig. 5. Histogram of the distance error $\epsilon = d - \hat{d}$ with distances estimated in a completed floor plans from the first 25 line segments of the test data (d_p), compared to distances estimated using just the 25 first line segments themselves (d_0). $\hat{d}$ is the distance using the true floor plan. The histogram bins are eight cell sides wide. The two plots are shifted slightly horizontally to improve legibility.

the next token will be — though not impossible since the segments are ordered by distance from the origin.

C. Novel and Partial Sequence Completion

Generative language models are often used to complete sentences with contextually relevant completions. In the same way, we can complete partial floor plan sequences, generating plausible continuations of partially observed floor plans. We present a set of such samples from our model in Fig. 1.

Sampling is performed iteratively in an autoregressive manner. Given a partial or empty sequence, the network returns a distribution over possible next tokens, a sample is drawn from this distribution, and then added to end of the sequence. This iteration continues until the sampled token is the `stop` token, or the maximum iteration count i_{max} is reached. This process is illustrated in Fig. 2. Nucleus sampling [20] is applied with top-p at $90\,\%$, which works by moving probability mass from the "unreliable tail" of the token distribution to its *nucleus*.

D. Partial Image Conditioning

We condition the model on binary 128×128 pixel images by rasterizing the first 25 line segments. The image is intentionally made ambiguous: there a quarter as many pixels as there are quantized coordinates, and only a subset of the line segments are drawn into the image. The network must therefore both learn to reproduce line segments from a rendering of the 25 nearest, and generate up to 75 novel line segments that fit the first 25.

To see the effect of the image embedding network, we report the performance of two models where *all* line segments are rasterized, with two different networks, ResNets [15] and MLP Mixer [16]. The MLP Mixer variant is $40\,\%$ smaller, yet it has $20\,\%$ lower NLL. The results are reported together with the non-image results in Table I.

The predicted token probability by the model, and the empirically estimated accuracy grouped by triplet pairs is presented in Fig. 4, where $c^\mu, x^\mu, y^\mu, c^\lambda, x^\lambda, y^\lambda$ refer to a pair of `move` and `line` triplets (denoted μ and λ respectively). We find the predicted probability to always be slightly less than the accuracy, and the difference is more pronounced when the probability is lower. Both drop sharply after the 25th triplet pair, which corresponds to the last visible line segment. We find that x^μ more or less determines the other three coordinates, suggesting that polar coordinates (ϕ, r) may be easier to model since the line segments are mostly ordered by r.

E. Predicting the Shortest Path

Finally, we evaluate the applicability of FloorGenT to a typical robotics task, namely, predicting the distance (and path) to an unknown point in space given environmental cues. The scenario is as follows: a robot observes its immediate surroundings, precisely the 25 nearest line segments at a location in the test set. FloorGenT is used to estimate the expected distance under the model given the observation via a Monte Carlo approximation. To that end, we generate 8 completions from the observation, and the completions are rasterized to create the occupancy grids $M_{p1}, \ldots, M_{p8}$. A shortest-path search from the origin to each cell in each occupancy grid is performed, with obstacles inflated by four cells. The predicted distance d_p for each cell is then defined to be the median distance of the 8 completions.

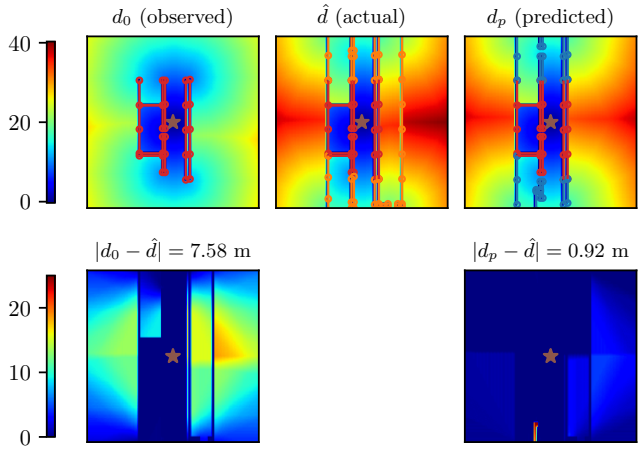

Fig. 6. An example of the three distance grids $d_0, \hat{d}, d_p$, and the absolute error $|\epsilon|$ per cell. All quantities are in meters. The corresponding line segments are drawn on top of each grid. Red line segments are observed, blue predicted, and orange is the true floor plan. The star denotes the origin, i.e., from where the distance is computed.

The same process is repeated for the "null hypothesis" occupancy grid M_0, created by rasterizing only the observed 25 line segments, and also repeated for the true floor plan rasterization $\hat{M}$. The occupancy grids are 256×256 cells over an area of $40\,\mathrm{m} \times 40\,\mathrm{m}$. The search is 8-connected and uses Euclidean distance cost. We denote the distance under the null hypothesis d_0, and the distance in the true floor plan $\hat{d}$. A cell is considered trivial if $d_p = d_0 = \hat{d}$ and is left out of the following statistical evaluation. An example calculation is presented in Fig. 6.

The mean absolute error $|\epsilon|$ is $6.21\,\mathrm{m}$ using predictions and $9.65\,\mathrm{m}$ ($+3.44\,\mathrm{m}$) under the null hypothesis, i.e., assuming that only the observed obstacles exist. A histogram of the errors is shown in Fig. 5. Note that the error ϵ is better centered around zero with predictions, while the null hypothesis by definition results in only underestimating, i.e., $\epsilon \leq 0$ (mean ϵ is $-4.03\,\mathrm{m}$ and $-9.65\,\mathrm{m}$ respectively).

Though we have only looked at the median distance, the same approach could for example be used to gauge certainty by computing the variance, or to assert the satisfaction of safety parameters, e.g., "cell can be reached without running out of battery".

V. Conclusions

We have presented FloorGenT, a generative model for floor plans, and showed its potency in modelling and predicting large-scale floor plans like the KTH floor plan dataset. We demonstrated the model's ability to incorporate other data modalities, in particular, rasterized images similar to occupancy grids which is a common representation in robotics applications, and showed its ability to reasoning about unknown space in a typical robotics task.

We believe there are many interesting real-world use cases for predictive floor plan modelling, and particularly in conjunction with sensor data. Concretely, we plan to employ map predictions in an autonomous exploration scenario as proposed in our previous work [21]. This could be accomplished with pretrained networks for the context embedding, and potentially combined with semi-supervised learning since there are relatively few datasets of indoor floor plans with sensory data, but abundant data from indoor scenes.

References

[1] Z. Zeng, X. Li, Y. K. Yu, and C.-W. Fu, "Deep floor plan recognition using a multi-task network with room-boundary-guided attention," in *International Conference on Computer Vision*. IEEE, 2019.

[2] A. van den Oord, N. Kalchbrenner, L. Espeholt, k. kavukcuoglu, O. Vinyals, and A. Graves, "Conditional image generation with pixelcnn decoders," in *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2016.

[3] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, 2019. [Online]. Available: https://openai.com/blog/better-language-models/

[4] M. Luperto, V. Arcerito, and F. Amigoni, "Predicting the layout of partially observed rooms from grid maps," in *International Conference on Robotics and Automation*. IEEE, 2019.

[5] M. Saroya, G. Best, and G. A. Hollinger, "Online exploration of tunnel networks leveraging topological cnn-based world predictions," in *International Conference on Intelligent Robots and Systems*. IEEE, 2020.

[6] L. Wang, H. Ye, Q. Wang, Y. Gao, C. Xu, and F. Gao, "Learning-based 3d occupancy prediction for autonomous navigation in occluded environments," in *International Conference on Intelligent Robots and Systems*. IEEE, 2021.

[7] A. Aydemir, P. Jensfelt, and J. Folkesson, "What can we learn from 38 000 rooms? reasoning about unexplored space in indoor environments," in *International Conference on Intelligent Robots and Systems*. IEEE, 2012.

[8] J. Chen, C. Liu, J. Wu, and Y. Furukawa, "Floor-SP: Inverse CAD for floorplans by sequential room-wise shortest path," in *International Conference on Computer Vision*. IEEE, 2019.

[9] A. Carlier, M. Danelljan, A. Alahi, and R. Timofte, "Deepsvg: A hierarchical generative network for vector graphics animation," *Advances in Neural Information Processing Systems*, 2020.

[10] P. Reddy, M. Gharbi, M. Lukac, and N. J. Mitra, "Im2Vec: Synthesizing vector graphics without vector supervision," in *Conference on Computer Vision and Pattern Recognition*. IEEE, 2021.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[12] C. Nash, Y. Ganin, S. A. Eslami, and P. Battaglia, "Polygen: An autoregressive generative model of 3D meshes," in *International Conference on Machine Learning*. PMLR, 2020.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017.

[14] G. Weiss, Y. Goldberg, and E. Yahav, "Thinking like transformers," in *International Conference on Machine Learning*. PMLR, 2021.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European Conference on Computer Vision*. Springer, 2016.

[16] I. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. P. Steiner, D. Keysers, J. Uszkoreit, M. Lucic, and A. Dosovitskiy, "MLP-mixer: An all-MLP architecture for vision," in *Advances in Neural Information Processing Systems Pre-proceedings*. Curran Associates, Inc., 2021.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.

[18] T. Bachlechner, B. P. Majumder, H. Mao, G. Cottrell, and J. McAuley, "ReZero is all you need: Fast convergence at large depth," in *Uncertainty in Artificial Intelligence*. PMLR, 2021.

[19] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh, "Set transformer: A framework for attention-based permutation-invariant neural networks," in *International Conference on Machine Learning*. PMLR, 2019.

[20] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi, "The curious case of neural text degeneration," in *International Conference on Learning Representations*. Curran Associates, Inc., 2019.

[21] L. Ericson, D. Duberg, and P. Jensfelt, "Understanding greediness in map-predictive exploration planning," in *European Conference on Mobile Robots*. IEEE, 2021.

[22] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

Appendix

A. Network Architecture Details

An attention layer is defined as $\boldsymbol{y} = \mathrm{AttnLayer}(\boldsymbol{x}_0)$ by

$$\begin{aligned}
\boldsymbol{x}_1 &= \boldsymbol{x}_0 + \alpha_1 \,\mathrm{D}_r(\mathrm{MHA}(\overline{\boldsymbol{x}_0}, \overline{\boldsymbol{x}_0}; N_{heads}, E)) \\
\boldsymbol{x}_2 &= \boldsymbol{x}_1 + \alpha_2 \,\mathrm{D}_r(\mathrm{MHA}(\overline{\boldsymbol{x}_1}, \boldsymbol{v}_{context}; N_{heads}, E)) \\
\boldsymbol{y} &= \boldsymbol{x}_2 + \alpha_3 \,\mathrm{D}_r(\mathrm{Linear}(\boldsymbol{x}_2))
\end{aligned}$$

where $\mathrm{Linear}(\boldsymbol{z}_0) = \boldsymbol{z}_2$ with

$$\begin{aligned}
\boldsymbol{z}_1 &= \mathrm{Dense}(\overline{\boldsymbol{z}_0}; N_{fc}) \\
\boldsymbol{z}_2 &= \mathrm{Dense}(\mathrm{ReLU}(\boldsymbol{z}_1); E)
\end{aligned}$$

The network is largely as proposed by [13]. D_r is dropout at rate r. $\overline{\boldsymbol{x}}$ is layer normalization as in [22]. $\mathrm{MHA}(\boldsymbol{q}, \boldsymbol{m})$ is

multiple heads of scaled dot-product attention as in [13], with queries $\boldsymbol{q}$, keys and values $\boldsymbol{m}$. Dense is a fully-connected layer with given number of output units. α_i are the ReZero coefficients. N_{heads} is the number of attention heads. E is the embedding dimension. $\boldsymbol{v}_{context}$ is the context embedding, Cf. image embedding in Fig. 2. For non-image models, we let $\boldsymbol{x}_2 = \boldsymbol{x}_1$. Each successive layer operates on the output of the previous, so the output $\boldsymbol{y}_{net}$ of an L-layer network is defined

$$\begin{aligned}\boldsymbol{y}_0 &= \textit{sequence embedding (Cf. Fig. 2)}\\ \boldsymbol{y}_i &= \mathrm{AttnLayer}(\boldsymbol{y}_{i-1})\\ \boldsymbol{y}_{net} &= \overline{\boldsymbol{y}_L}\end{aligned}$$

Finally, $\boldsymbol{y}_{net}$ is projected to $|\mathcal{T}|$ dimensions to obtain the logits of the predictive distribution $p(t_i|\boldsymbol{t}_{<i};\theta)$.

B. Equivalent MLP

In the "Equivalent MLP" model, we take N_w subsequences of each input sequence in a sliding window fashion, e.g., the sequence *abcdef* would under a sliding window with $N_w = 3$ yield four subsequences: *abc, bce, cde, def*. Padding items are prepended so that we obtain as many subsequences as there are tokens in the input, e.g., *ssabcdef* with a start token *s*. The network input $\boldsymbol{y}_0$ and output $\boldsymbol{y}_{net}$ is as before, but with the preprocessing step $\boldsymbol{y}_1$ and a different layer definition $\boldsymbol{y}_i$

$$\begin{aligned}\boldsymbol{y}_1 &= \mathrm{Join}(\mathrm{SlidingWindow}(\boldsymbol{y}_0); N_w{\cdot}E)\\ \boldsymbol{y}_i &= \mathrm{MLP\,Layer}(\boldsymbol{y}_{i-1})\end{aligned}$$

where $\mathrm{SlidingWindow}$ is said sliding window operation and yields a $S{\times}N_w{\times}E$ tensor for input length S. Join flattens the last two dimensions as indicated, i.e., a concatenation of the embeddings in each window. A single layer is defined $\boldsymbol{y} = \mathrm{MLP\,Layer}(\boldsymbol{x}_0)$ with

$$\begin{aligned}\boldsymbol{x}_1 &= \mathrm{Dense}(\overline{\boldsymbol{x}_0}; N_w{\cdot}N_{fc})\\ \boldsymbol{x}_2 &= \mathrm{Split}(\boldsymbol{x}_1; N_w{\times}N_{fc})\\ \boldsymbol{x}_3 &= \mathrm{Dense}(\mathrm{ReLU}(\boldsymbol{x}_1); E)\\ \boldsymbol{x}_4 &= \mathrm{Join}(\boldsymbol{x}_3; N_w{\cdot}E)\\ \boldsymbol{y} &= \boldsymbol{x}_0 + \alpha\,\mathrm{D}_r(\boldsymbol{x}_4)\end{aligned}$$

Split is the inverse of Join and unflattens the two last dimensions as indicated. The above setup mimics the attention layers: the first dense layer can "attend" to anything in the given window, and the second dense layer projects each position individually to E dimensional embeddings. In our experiments, we have six MLP layers, $E = N_{fc} = 12$, and $r = 5\,\%$.