
Differentiable Spatial Planning using Transformers

Devendra Singh Chaplot^{1,2} Deepak Pathak² Jitendra Malik^{1,3}

Project webpage: <https://devendrachaplot.github.io/projects/spatial-planning-transformers>

Abstract

We consider the problem of spatial path planning. In contrast to the classical solutions which optimize a new plan from scratch and assume access to the full map with ground truth obstacle locations, we learn a planner from the data in a differentiable manner that allows us to leverage statistical regularities from past data. We propose Spatial Planning Transformers (SPT), which given an obstacle map learns to generate actions by planning over long-range spatial dependencies, unlike prior data-driven planners that propagate information locally via convolutional structure in an iterative manner. In the setting where the ground truth map is not known to the agent, we leverage pre-trained SPTs in an end-to-end framework that has the structure of mapper and planner built into it which allows seamless generalization to out-of-distribution maps and goals. SPTs outperform prior state-of-the-art differentiable planners across all the setups for both manipulation and navigation tasks, leading to an absolute improvement of 7-19%.

1. Introduction

The problem of path planning has been a bedrock of robotics. Given an obstacle map of an environment and a goal location, the task is to output the shortest path to the goal location starting from any position in the map. We consider path planning with spatial maps. Building a top-down spatial map is common practice in robotic navigation as it provides a natural representation of physical space (Durrant-Whyte & Bailey, 2006). Robotic manipulation can also be naturally phrased via spatial map using the formalism of configuration spaces (Lozano-Perez, 1990), as shown in Fig 1. This problem has been studied in robotics for several decades, and

¹Facebook AI Research ²Carnegie Mellon University ³UC Berkeley. Correspondence to: Devendra Singh Chaplot <dchaplot@fb.com>.

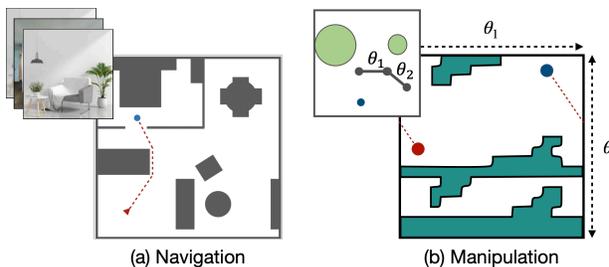


Figure 1. Spatial Path Planning: The raw observations (top left) and obstacles can be represented spatially via top-down map in navigation (left) and via configuration space in manipulation (right).

classic planning algorithms include Dijkstra et al. (1959), PRM (Kavraki et al., 1996), RRT (LaValle & Kuffner Jr, 2001), RRT* (Karaman & Frazzoli, 2011), etc.

Our objective is to develop methods that can *learn* to plan from data. However, a natural question is why do we need learning for a problem which has stable classical solutions? There are two key reasons. First, classical methods do not capture statistical regularities present in the natural world, (for e.g., walls are mostly parallel or perpendicular to each other), because they optimize a plan from scratch for each new setup. This also makes analytical planning methods to be often slow at inference time which is an issue in dynamic scenarios where a more reactive policy might be required for fast adaptation from failures. A learned planner represented via a neural network can not only capture regularities but is also efficient at inference as the plan is just a result of forward-pass through the network. Second, a critical assumption of classical algorithms is that a global ground-truth obstacle space must be known to the agent ahead of time. This is in stark contrast to biological agents where cognitive maps are not pixel-accurate ground truth location of agents, but built through actions in the environment, e.g., rats build an implicit map of the environment incrementally through trajectories enabling them to take shortcuts (Tolman, 1948). A learned solution could not only provide the ability to deal with partial, noisy maps and but also help build maps on the fly while acting in the environment by backpropagating through the generated long-range plans.

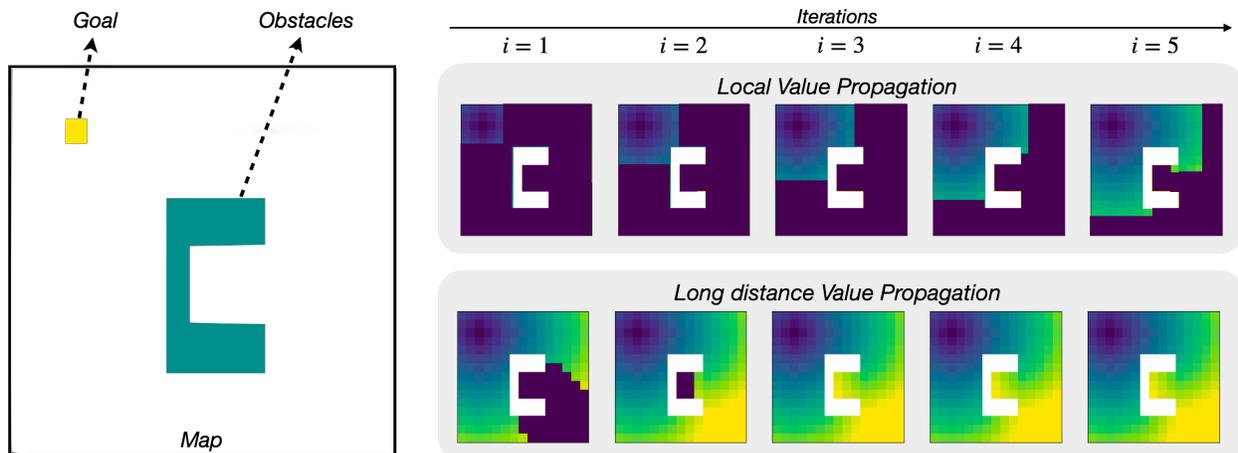


Figure 2. Local vs Long-distance value propagation. Figure showing an example of number of iterations required to propagate distance values over a map using local and long-distance value propagation. The obstacle map and goal location shown on the left and the distance value predictions over 5 iterations is shown on the right (distance values increase from blue to yellow). Prior methods based on convolutional networks use local value propagation and require many iterations to propagate values accurately over the whole map (top right). Our method is based on long-distance value propagation between points without any obstacle between them. This type of value propagation can cover the whole map in 3 iterations in this example (bottom right).

Several recent works have proposed data-driven path planning models (Tamar et al., 2016; Karkus et al., 2017; Nardelli et al., 2019; Lee et al., 2018). Similar to how classical algorithms, like Dijkstra et al. (1959), move outward from the goal one cell at a time to predict distances iteratively based on the obstacles in the map, current learning-based spatial planning models propagate distance values in only a *local* neighborhood using convolutional networks. This kind of local value propagation requires $\mathcal{O}(D)$ iterations, where D is the shortest-path distance between two cells. For two corner cells in a map of size $M \times M$, D can vary from M to M^2 . In theory, however, the optimal paths can be computed much more efficiently with total iterations that are on the order of number of obstacles rather than the map size. For instance, consider two points with no obstacle between them, an efficient planner could directly connect them with interpolated distance. This is possible only if the model can perform long-range reasoning in the obstacle space which is a challenge.

In this work, our goal is to capture this *long-range* spatial relationship. Transformers (Vaswani et al., 2017) are well suited for this kind of computation as they treat the inputs as sets and propagate information across all the points within the set. Building on this, we propose *Spatial Planning Transformers* (SPT) which consists of attention heads that can attend to any part of the input. The key idea behind the design of the proposed model is that value can be propagated between distant points if there are no obstacles between them. This would reduce the number of required iterations to $\mathcal{O}(n_O)$ where n_O is the number of obstacles in the map. Figure 2 shows a simple example where long-

distance value propagation can cover the entire map within 3 iterations while local value propagation takes more than 5 iterations – this difference grows with the complexity of the obstacle space and map size. We compare the performance of SPTs with prior state-of-the-art learned planning approaches, VIN (Tamar et al., 2016) and GPPN (Lee et al., 2018), across both navigation as well as manipulation setups. SPTs achieve significantly higher accuracy than these prior methods for the same inference time and show over 10% absolute improvement when the maps are large.

Next, we turn to the case when the map is not known apriori. This is a practical setting when the agent either has access to a partially known map or just know it through the trajectories. In psychology, this is known as going from *route* knowledge to *survey* knowledge (Golledge et al., 1995) where animals aggregate the knowledge from trajectories into a cognitive map. We operationalize this setup by formulating an end-to-end differentiable framework, which in contrast to having a generic parametric policy learning (Glas-machers, 2017), has the structure of mapper and planner built into it. We first pre-train the SPT planner to capture a generic data-driven prior and then backpropagate through it to learn a mapper that maps raw observations to an obstacle map. This allows us to learn without requiring map supervision or interaction. Learned mapper and planner not only allow us to plan for new goal locations at inference but also generalize to unseen maps. SPT outperforms both classical algorithms and prior learning-based planning methods on both manipulation and navigation tasks resulting in an absolute improvement of over 18%.

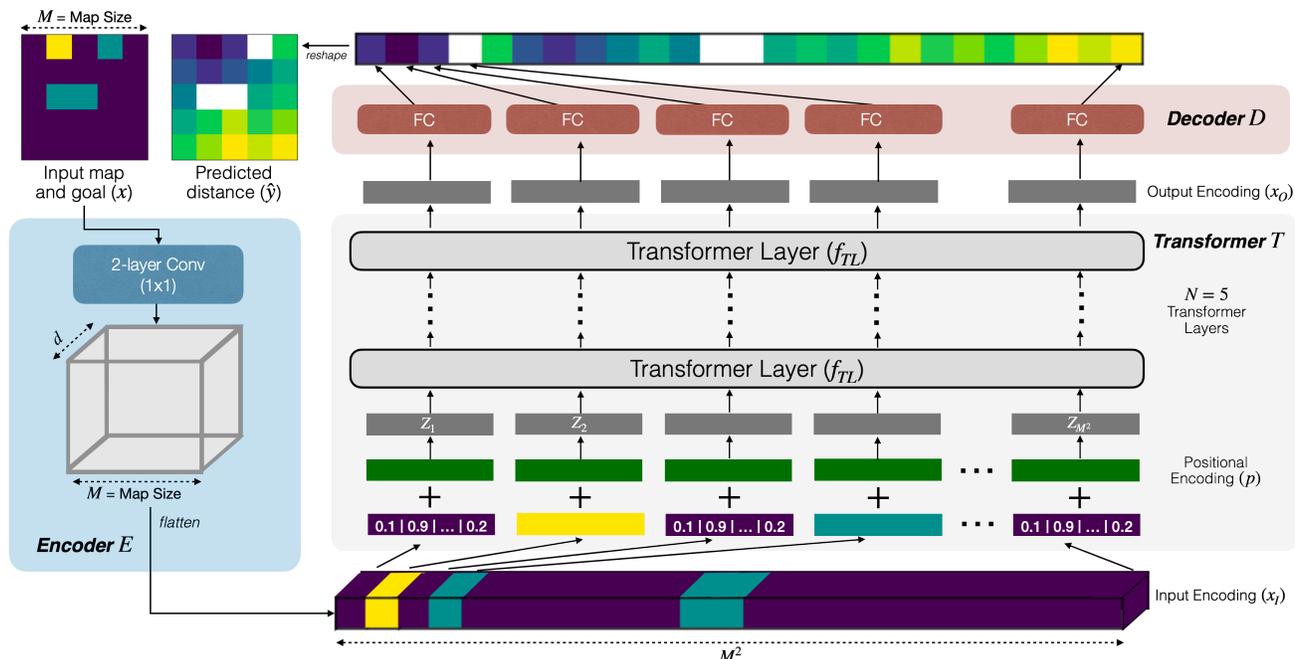


Figure 3. **Spatial Planning Transformer (SPT)**. Figure showing an overview of the proposed Spatial Planning Transformer model. It consists of 3 modules: an *Encoder E* to encode the input, a *Transformer* network *T* responsible for planning, and a *Decoder D* decoding the output of the Transformer into action distances.

2. Preliminaries and Problem Definition

We represent the input spatial map as a matrix, m , of size $M \times M$ with each element being 1, denoting obstacles, or 0, denoting free space. The goal location is also represented as a matrix, g , of size $M \times M$ with exactly one element being 1, denoting the goal location, and rest 0s. The input to the spatial planning model, x , consists of matrices m and g stacked, $x = [m, g]$, where x is of size $2 \times M \times M$. The objective of the planning model is to predict y which is of size $M \times M$, consisting of action distances of corresponding locations to the goal. Here, action distance is defined to be the minimum number of actions required to reach the goal.

For navigation, m is a top-down obstacle map, and g represents the goal position on this map. For manipulation, m represents the obstacles in the configuration space of 2-dof planar arm with joint angles denoted by θ_1 and θ_2 . Each element (i, j) in m indicate whether the configuration of the arm with joint angles $\theta_1 = i$ and $\theta_2 = j$, would lead to a collision. g represents the goal configuration of the arm. In the first set of experiments, we will assume that m is known and in the second set of experiments, m is not known and the agent receives observations, o , from its sensors instead.

3. Methods

We design a spatial planning model, called Spatial Planning Transformer (SPT), capable of long-distance information

propagation. We first describe the design of the SPT model, which takes in a map and a goal as input and predicts the distance to the goal from all locations. We then describe how the SPT model can be used as a planning module to train end-to-end learning models, which take in raw sensory observations and goal location as input and predict action distances without having access to the map.

3.1. SPT: Spatial Planning Transformers

To propagate information over distant points, we use the Transformer (Vaswani et al., 2017) architecture. The self-attention mechanism in a Transformer can learn to attend to any element of the input. This allows the model to learn spatial reasoning over the whole map concurrently. Figure 3 shows an overview of the SPT model, which consists of three modules, an *Encoder E* to encode the input, a *Transformer* network *T* responsible for spatial planning, and a *Decoder D* decoding the output of the Transformer into action distances.

Encoder. The Encoder *E* computes the encoding of the input x : $x_I = E(x)$. The input $x \in \{0, 1\}^{2 \times M \times M}$ consisting of the map and goal is first passed through a 2-layer convolutional network (LeCun et al., 1998) with ReLU activations to compute an embedding for each input element. Both layers have a kernel size of 1×1 , which ensures that the embedding of all the obstacles is identical to each other, and the same holds true for free space. The output of this

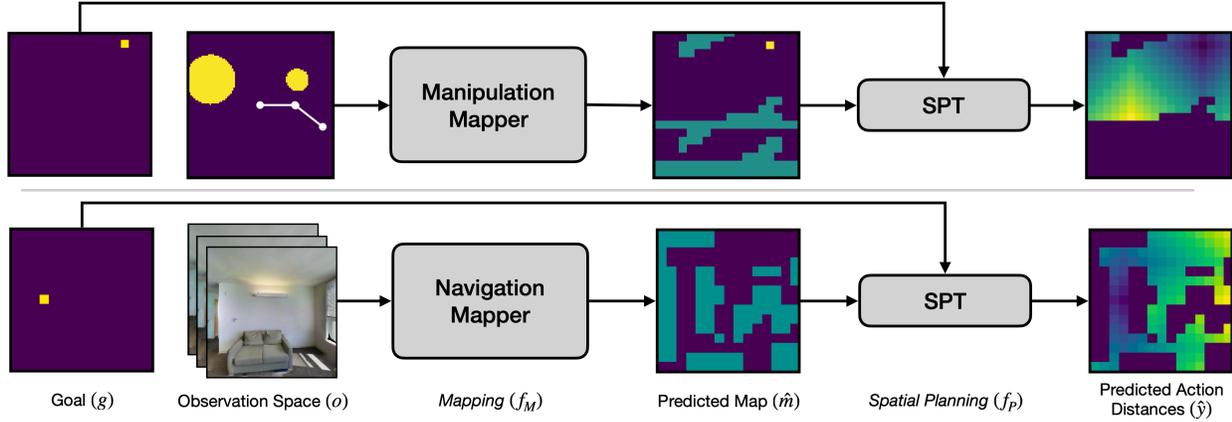


Figure 4. **End-to-end Mapping and Planning.** Figure showing an overview of end-to-end mapping and planning model for both the navigation and manipulation tasks.

convolutional network is of size $d \times M \times M$, where d is the embedding size. This output is then flattened to get x_I of size $d \times M^2$ and passed into the Transformer network.

Transformer. The Transformer network T converts the input encoding into the output encoding: $x_O = T(x_I)$. It first adds the positional encoding to the input encoding which enables the Transformer model to distinguish between the obstacles at different locations. We use a constant sinusoidal positional encoding (Vaswani et al., 2017):

$$p_{(2i,j)} = \sin(j/C^{2i/d}), \quad p_{(2i+1,j)} = \cos(j/C^{2i/d})$$

where $p \in \mathcal{R}^{d \times M^2}$ is the positional encoding, $j \in \{1, 2, \dots, M^2\}$ is the position of the input, $i \in \{1, 2, \dots, d/2\}$, and $C = M^2$ is a constant.

The positional encoding of each element is added to their corresponding input encoding to get $Z = x_I + p$. Z is then passed through $N = 5$ identical Transformer layers (f_{TL}) to get x_O (see Appendix A for a background on Transformers).

Decoder. The Decoder D computes the distance prediction \hat{y} from x_O using a position-wise fully connected layer:

$$\hat{y}_i = W_D^T x_{T,i} + b_D$$

where $x_{T,i} \in \mathcal{R}^{d \times 1}$ is the input at position $i \in \{1, 2, \dots, M^2\}$, $W_D \in \mathcal{R}^{d \times 1}$, $b_D \in \mathcal{R}$ are parameters of the Decoder shared across all positions i and $\hat{y}_i \in \mathcal{R}$ is the distance prediction at position i . The distance prediction at all position are reshaped into a matrix to get the final prediction $\hat{y} \in \mathcal{R}^{M,M}$. The entire model is trained using pairs of input x and output y datapoints with mean-squared error as the loss function.

3.2. Planning under unknown maps

The SPT model described above is designed to predict action distances given a map as input. However, in many applications, the map of the environment is often not known.

In such cases, an autonomous agent working in a realistic environment needs to predict the map from raw sensory observations. While it is possible to train a separate mapper model to predict maps from observations, this often requires map annotations which are expensive to obtain and often inaccurate. In contrast, demonstration trajectories consisting of observations and optimal actions are more readily available or easier to obtain in many robotics applications. One of the key benefits of learning-based differentiable spatial planning over classical planning algorithms is that it can be used to learn mapping just from action supervision in an end-to-end fashion without having access to ground-truth maps. To demonstrate this benefit, we train an end-to-end mapping and planning model to predict action distances from sensor observations for both navigation and manipulation tasks.

The end-to-end model consists of two modules, a Mapper (f_M) and a Planner (f_P), as illustrated in Figure 4. The Mapper is used to predict the map \hat{m} from sensor observations o and the Planner is a spatial planning model to predict action distances, \hat{y} , from the predicted map \hat{m} :

$$\hat{y} = f_P(\hat{m}) = f_P(f_M(o))$$

For navigation, o is the set of first-person RGB camera images each of size $3 \times H \times W$. We sample 4 images, one for each orientation, at each valid location in the map. For invalid locations, we pass an empty image of 0s for all orientations. Thus, for a map of size $M \times M$, observation o consists of $4M^2$ images for all locations and 4 orientations similar to the setup in Lee et al. (2018). For manipulation, o is a top-down view of the operational space with obstacles of size $P \times P$, where each element is 1 or 0 denoting obstacles or free space. We use different Mapper architectures for navigation and manipulation experiments.

The Navigation Mapper module predicts a single value between 0 and 1 for each image in o indicating whether the cell in the front of the image is an obstacle or not. The

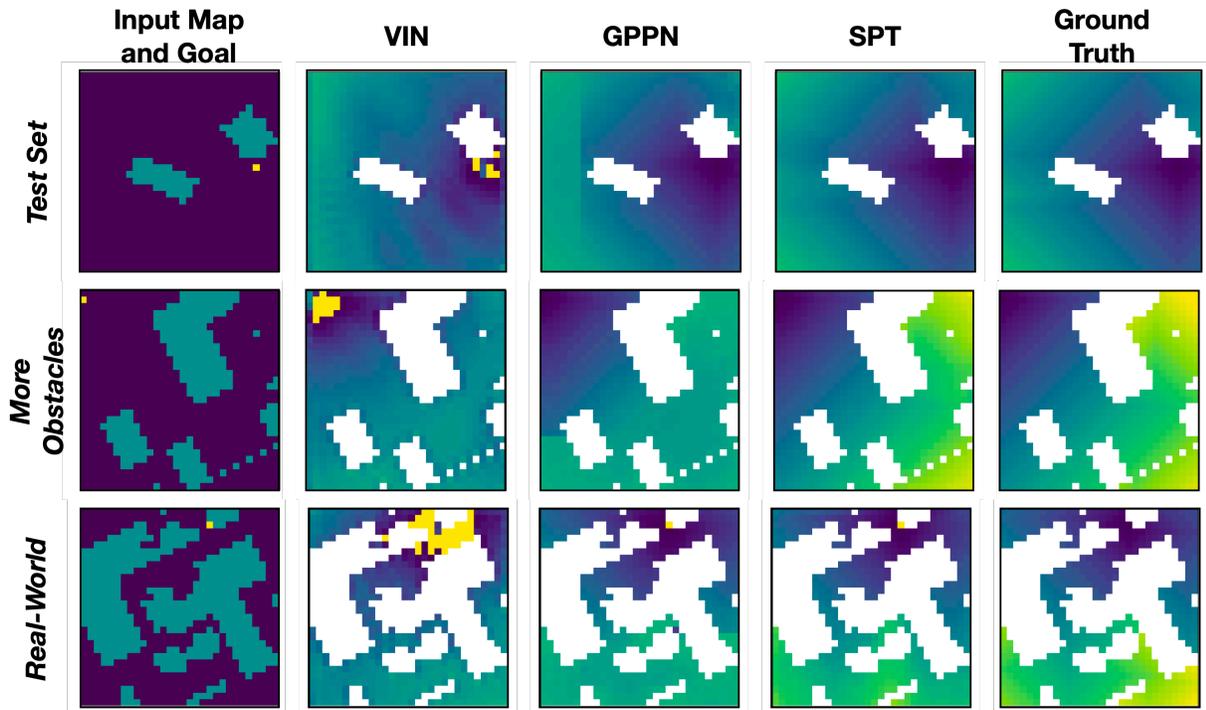


Figure 5. **Spatial Planning Examples.** Figure showing 3 examples of the input, the predictions using the proposed SPT model and the baselines, and the ground truth for map size $M = 30$. The obstacles are shown in blue, free space in purple and goal in yellow in the leftmost input column. The predictions and ground truth in the rest of the column are color-coded from blue to yellow to represent increasing action distance.

architecture of the Navigation mapper consists of ResNet18 convolutional layers followed by fully-connected layers (see Appendix C for details). Each cell can have up to 4 predictions (from images corresponding to the four neighboring cells facing the current cell), which are aggregated using max-pooling to get a single prediction. Predictions for all the cells are arranged in a matrix to get the whole map prediction which is then passed to the Planner module.

The Manipulation Mapper module needs to predict which configurations of the arm would lead to a collision. For each configuration (θ_1, θ_2) , the mapper module needs to check whether any point in this configuration consists of an obstacle. A Transformer-based model is well suited to learn this function as well as it can attend to arbitrary locations in the operational space to predict the obstacles in the configuration space. We use the same architecture of the SPT model as the Manipulation Mapper as well, with the only difference being the encoder consisting of 3×3 kernel size convolutional layers instead of 1×1 to encode the $P \times P$ observation space to a $M \times M$ representation.

The Planner module is the SPT model with encoder, transformer, and decoder units as described in the previous subsection. It is pretrained on synthetic maps and its weights are frozen during end-to-end training. We train the entire

end-to-end mapping and planning model with pairs of input observations o and output action distances y using standard supervised learning with the mean-squared error loss function. Since the planning module is pretrained it expects a structured map input, the mapper module learns to predict the map accurately such that the predicted map, when passed through the planner, minimizes the action level loss.

4. Experiments & Results

We conduct experiments to test the effectiveness of the proposed SPT model as compared to prior differentiable planning methods. We first conduct experiments when the map is known in Section 4.1. We then conduct experiments when the map is not known in Section 4.2. In this setting, the map needs to be predicted from sensory observations without having access to map-level supervision using end-to-end mapping and planning. We compare the SPT model with prior differentiable planning models keeping the mapping model identical across all learning-based methods.

4.1. Known maps

Datasets. We generate synthetic datasets for training the spatial planning models for both navigation and manipulation settings. For the navigation setting, we perform exper-

Method	Navigation			Manipulation		Overall
	M=15	M=30	M=50	M=18	M=36	
VIN	86.19	83.62	80.84	75.06	74.27	80.00
GPPN	97.10	96.17	91.97	89.06	87.23	92.31
SPT	99.07	99.56	99.42	99.24	99.78	99.41

Table 1. **Generalization to in-distribution maps.** Average planning accuracy of the proposed model Spatial Planning Transformer (SPT) as compared to the baselines on in-distribution test sets for both the navigation and manipulation experiments.

Method	Navigation						Manipulation		Overall
	More Obstacles			Real-World			More Obstacles		
	M=15	M=30	M=50	M=15	M=30	M=50	M=18	M=36	
VIN	49.05	62.05	70.64	49.91	56.67	71.16	65.27	59.81	60.57
GPPN	90.68	89.93	84.86	90.11	91.07	88.32	79.86	80.79	86.95
SPT	93.34	92.71	92.03	95.96	94.70	95.39	98.16	99.18	95.18

Table 2. **Generalization to out-of-distribution maps.** Average planning accuracy of the proposed model Spatial Planning Transformer (SPT) as compared to the baselines on out-of-distribution test sets for both the navigation and manipulation experiments.

iments with $M \times M$ maps with three different map sizes, $M \in \{15, 30, 50\}$. For manipulation, we experiment with two map sizes, $M \in \{18, 36\}$, corresponding to 20° and 10° bins for each link. In each map, we randomly generate $o_{min} = 0$ to $o_{max} = 5$ obstacles. Dataset generation details are provided in the Appendix B.

For both the settings, we generate training, validation, and test sets of size $100K/5K/5K$ maps. For each map, we choose a random free space cell as the goal location. The action space consists of 4 actions: north, south, east, west. For the navigation task, the map boundaries are considered as obstacles, while for the manipulation task the cells on the left and right boundaries and top and bottom boundaries are connected to each other since angles are circular. The ground truth action distances are calculated using the Dijkstra algorithm (Dijkstra et al., 1959). Unreachable locations and obstacles are denoted by -1 in the ground truth.

In addition to testing on unseen maps with the same distribution, we also test on two types of out-of-distribution datasets: (1) **More Obstacles**, where we generate $o_{min} = 15$ to $o_{max} = 20$ obstacles per map, and (2) **Real-world**, where the top-down maps are generated from reconstructions of real-world scenes in the Gibson dataset (Xia et al., 2018).

Hyperparameters and Training. For training the SPT model, we use Stochastic Gradient Descent (Bottou, 2010) for optimization with a starting learning rate of 1.0 and a learning rate decay of 0.9 per epoch. We train the model for 40 epochs with a batch size of 20. We use $N = 5$ Transformer layers each with $h = 8$ attention heads and an embedding size of $d = 64$. The inner dimension of the fully connected layers in the transformer is $d_{fc} = 512$. We use the same architecture with the same hyperparameters for training the SPT model for both navigation and manipulation for all map sizes.

Baselines. Our baselines are prior spatial planning models, Value Iteration Networks (VIN) (Tamar et al., 2016) and Gated Path-Planning Networks (GPPN) (Lee et al., 2018). For tuning the hyperparameter (K) for the number of iterations in both the baselines, we consider all values of K in multiples of 10 such that the inference time of the baseline is comparable to the inference time of the SPT model (≤ 1.1 times). For each setting, we tune K and the learning rate to maximize performance on the validation set.

Metrics. We use average action prediction accuracy as the metric. Distance prediction is converted to actions by finding the minimum distance cell among the 4 neighboring cells for each location. When multiple actions are optimal, predicting any optimal action is considered to be a correct prediction. The accuracy is averaged over all free space locations over all maps in the test set.

Results. The planning accuracy of all the methods for both the navigation and manipulation tasks on the in-distribution test sets are shown in Table 1 and on the out-of-distribution test sets are shown in Table 2. The proposed SPT model outperforms both the baselines across all settings achieving an overall accuracy of 99.41% vs 92.31% (in-distribution) and 95.18% vs 86.95% (out-of-distribution) as compared to the best baseline. The performance of the SPT model is stable as the map size increases while the performance of the baselines drops considerably. We believe this is because both the baselines need to use a larger number of iterations to cover a larger map ($K = 60$ iterations for GPPN and $K = 90$ iterations for VIN for $M = 50$) since the information propagation is local in VIN and GPPN. The optimization becomes difficult for such deep models. In contrast, the SPT model uses a constant $N = 5$ layers for all map sizes.

The improvement in the performance of SPT over the baselines is larger in the manipulation task because the baselines

Method	Navigation		Manipulation		Overall	
	Map Acc	Plan Acc	Map Acc	Plan Acc	Map Acc	Plan Acc
Classical	64.43	45.20	-	-	-	-
VIN	60.92	47.77	81.25	66.45	71.08	57.11
GPPN	69.06	45.70	85.57	82.13	77.31	63.91
SPT	82.58	66.16	98.96	98.42	90.77	82.29

Table 3. **End-to-End Mapping and Planning Results.** Average mapping and planning accuracy of the proposed model Spatial Planning Transformer (SPT) as compared to the baselines for end-to-end mapping and planning experiments.

based on convolution operations are not well suited for propagating information looping over the edges of the map. In contrast, the SPT model can use self-attention to attend to any part of the map and learn to propagate information over the map edges.

Visualizations. In Figure 5, we show examples of predictions of the SPT model as compared to the baselines for 3 different input maps and goals from 3 different test sets. The examples show that the baselines are not able to predict the distances of distant cells accurately. This is because they propagate information in a local neighborhood that can not reach distant cells in the limited inference time budget ($K = 30$ for VIN and $K = 20$ for GPPN). In contrast, the SPT model is able to predict distances of distant cells more accurately with $N = 5$ layers indicating that it learns long-range information propagation. Additional examples are provided in Appendix E.

4.2. Unknown maps

In the above experiments, we compared the planning performance of different methods under perfect knowledge of the map m . In this section, we test the efficacy of spatial planning methods when map m is unknown and needs to be predicted from sensor observations o .

Datasets. For manipulation, we generate synthetic datasets of size $M = 18$ using the same process as described in Section 4. We discretize the operation space into a $P \times P$ image with $P = 90$ which is used as the observation o . The train/test sets are of size 100K/5K.

For navigation, we use the Gibson dataset (Xia et al., 2018) to sample maps of size $M = 15$ where each cell is $0.25m^2$ area. We get the camera images at the navigable locations in all 4 orientations using the Habitat simulator (Savva et al., 2019). The set of camera images each of size $3 \times H \times W$ act as the observation o for the navigation task, where $H = W = 128$. The train and test sets consist of 72 and 14 distinct scenes identical to the standard train and val splits in the Habitat simulator. We sample 500 maps in each scene creating training/test sets of size 36K/7K. Each sampled map is rotated to a random orientation.

Training. We load the weights of different models trained on synthetic data from the previous section. We then train the end-to-end model using the same action distance prediction loss while keeping the planner weights frozen. The architecture of the mapper module is identical across different planning methods. **Metrics.** We report both map accuracy and planning accuracy for both the tasks.

Baselines. In addition to using VIN and GPPN as baselines, we also use a classical mapping and planning baseline for navigation. Since there is no depth input available, we used the Monocular depth estimation model from Hu et al. (2019) for predicting the map which is then used for planning using Dijkstra as suggested by Mishkin et al. (2019).

Results. Table 3 shows the end-to-end mapping and planning results. SPT outperforms both GPPN and VIN by a large margin across both the tasks achieving an overall plan accuracy of 82.29% vs 63.91%. Table 3 also shows that the mapper learnt using end-to-end training with a pretrained SPT model is able to achieve an accuracy of 98.96% for manipulation and 82.58% for navigation, without receiving any map-level supervision. SPT also outperforms the classical mapping and planning baseline. These results demonstrate a key benefit of learning-based differentiable planners as compared to classical analytical planning algorithms. As SPT outperforms VIN and GPPN at spatial planning, it also leads to a better map accuracy (90.77% vs 77.31%).

5. Analysis

Runtime Comparison. To demonstrate one of the benefits of learning-based planners over classical planning algorithms, we compare the runtime of SPT to Dijkstra (Dijkstra et al., 1959) and A* (Hart et al., 1968) algorithms in Table 4. The results indicate that SPT is $1.24 \times$ to $20.22 \times$ faster than classical planning algorithms with the runtime benefit improving with the increase in map size.

Long-range value propagation. The SPT model is designed to capture long-range spatial relationships in planning and propagate value over distant points. In Fig 6, we plot mean-squared error in planning vs action distance for different methods on the navigation task (known map) with

Method	Runtime per map in ms		
	M=15	M=30	M=50
Dijkstra	4.17	43.82	371.05
A*	3.02	35.38	294.70
SPT	2.44	4.72	18.35

Table 4. **Runtime comparison.** Comparison of average runtime per map in milli seconds for different methods. All values are averaged over 10000 maps.

different map sizes. The figure shows that the difference between the planning error of SPT and the baselines increases with action distances. This result indicates that SPT can propagate values over longer distances more effectively as compared to the baselines.

Sparse and Noisy supervision. In Section 4.2, we assumed access to perfect and dense action-distance supervision. In practice, if we were to get supervision from human trajectories, the supervision could be *sparse*, as we might not have access to the optimal distance from all locations in the map, and *noisy* as humans might not take the optimal actions always and computing distances from human trajectories might be noisy. To analyze the effect of not having dense and perfect supervision for training the end-to-end mapping and planning model, we consider three settings:

Noisy supervision: We add zero-mean Gaussian noise to all ground-truth distance values with std. deviation, $\sigma = 1$.

Sparse supervision: Instead of providing ground-truth distances from all navigable locations, we provide distances for only 5 trajectories to the same goal in the training maps.

Noisy and Sparse supervision: We provide noisy distances for only 5 trajectories as supervision.

Figure 7 shows an example of noisy and sparse supervision. The results are shown in Table 5. The SPT model maintains performance benefits over the baselines under all the settings. Interestingly, under sparse supervision, the map prediction accuracy drops, but the planning accuracy does not drop as much. This is because the model learns to predict the minimum map required to predict the action distances of all valid locations accurately as seen in examples shown in Figure ?? in the Appendix.

Scalability. We chose an action space of 4 axis-aligned actions in our experiments to replicate the evaluation setting of our baselines. We believe higher dimensional action spaces favor SPT as it does not rely on local convolutional operations. To test whether SPT maintains performance benefits in higher dimensional state and action spaces, we conducted some experiments for the navigation task. We relaxed the action space from 4 actions to 100 actions by just allowing the agent to take any action in a 10x10 grid around it (and using a low-level controller to go to any cell).

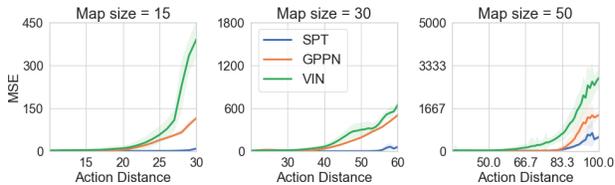


Figure 6. **MSE vs Action Distance.** Figure showing plots of mean-square error (MSE) in planning vs action distance for different methods on the navigation task with different map sizes.

The state space used for planning is discretized but the agent moves in a continuous state space in the Habitat simulator. We compute the continuous ground truth distance using the Fast Marching Method (instead of Dijkstra) for training with a larger action space, which allows us to accurately compute the distance for all locations and not be constrained by axis-aligned actions and distances.

SPT and the baselines are trained only on synthetic navigation mazes for this experiment. During evaluation in the Habitat simulator, we assume a perfect partial map based on part of the environment seen in the observations so far for planning. If the overall map size at this level of discretization is higher than planning map size, we simply use greedy planning in a window around the agent resulting in an “any-time” variant similar to the classical planning algorithms. This setup results in much finer-grained action space. SPT achieves a navigation success rate of 78.0% as compared to 47.2% for GPPN and 43.5% for VIN baselines.

6. Related Work

Path planning in known or inferred maps, also known as motion planning in robotics, is a well-explored problem led by the seminal papers (Canny, 1988; Kavraki et al., 1996; LaValle & Kuffner Jr, 2001; Karaman & Frazzoli, 2011). Although there are learned variants of motion planners proposed in the literature using gaussian processes (Ijspeert et al., 2013; Ratliff et al., 2018), data-driven motion planners using neural networks is a recent direction (Qureshi et al., 2019; Bhardwaj et al., 2020; Qureshi et al., 2020). Prior work has also studied the use of neural networks to learn the heuristics and sampling strategies in classical planners Ichter et al. (2018); Guez et al. (2018); Satorras & Welling (2021); Khan et al. (2020). Learning for planning is more common in Markov Decision Process (MDPs) for computing value function via dynamic programming based value iterations (Bellman, 1966; Bertsekas et al., 1995). Planning and learning in neural networks has been explored (Ilin et al., 2007) with a successful general formulation provided by value iteration networks (VIN) (Tamar et al., 2016) with follow-ups to improve scalability and efficiency (Lee et al., 2018; Karkus et al., 2017; Nardelli et al., 2019; Schleich et al., 2019; Khan et al., 2018; Chen et al., 2020). However,

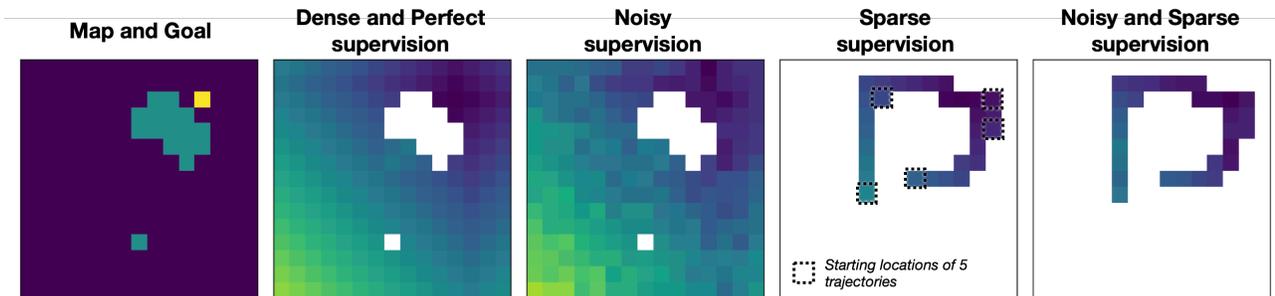


Figure 7. **Sparse and Noisy Supervision.** Figure showing examples of a map and goal with different levels of supervision. Noisy supervision adds gaussian noise to the ground truth distance values, and sparse supervision samples 5 trajectories for random starting locations to the goal location.

Method	Dense and Perfect supervision		Noisy supervision		Sparse supervision		Noisy and Sparse supervision	
	Map Acc	Plan Acc	Map Acc	Plan Acc	Map Acc	Plan Acc	Map Acc	Plan Acc
VIN	81.25	66.45	75.68	60.78	70.16	60.23	70.22	58.97
GPPN	85.57	82.13	80.13	76.11	72.73	75.13	70.08	72.85
SPT	98.96	98.42	96.35	95.83	80.15	97.18	77.17	94.34

Table 5. **Sparse and Noisy Supervision Results.** Table showing the average mapping and planning accuracy of the proposed model Spatial Planning Transformer (SPT) as compared to the baselines for end-to-end mapping and planning experiments under noisy and sparse supervision settings for the manipulation task.

these models only capture local value propagation using CNNs and are mostly applied in navigation setups. In contrast, proposed SPTs capture long-range spatial dependency and easily scale to both navigation and manipulation.

Differentiable planning structure has also been explored in reinforcement learning with model-free methods (Silver et al., 2017; Oh et al., 2017; Zhu et al., 2017; Farquhar et al., 2018) as well as off-policy RL (Eysenbach et al., 2019; Laskin et al., 2020). Recent works also backpropagate through learned planners to train the policy (Pathak et al., 2018; Srinivas et al., 2018; Amos et al., 2018) and use imagined rollouts of a learned world model for long-term plans (Racanière et al., 2017; Hafner et al., 2019; Sekar et al., 2019). Unlike our work, these works lack the structure of a spatial planner.

Decomposing learning a controller into mapping and planning is common in robot navigation (Khatib, 1986; Elfes, 1987). Some works have explored joint mapping and planning (Elfes, 1989; Fraundorfer et al., 2012). Maps can also be built from vision (Konolige et al., 2010; Fuentes-Pacheco et al., 2015) with a learned mapper (Parisotto & Salakhutdinov, 2018; Karkus et al., 2020). There has been some work on learning maps without using map annotations as well (Gregor et al., 2019). For navigation specific applications, recent works proposed joint mapping and planning for navigation (Gupta et al., 2017; Zhang et al., 2017; Savinov et al., 2018; Chaplot et al., 2020b;a;c). However, most of these works either require access to ground truth map or

assume interaction. Hence, they will first need to be trained in simulation. In contrast, we show results when the map is not known to the agent by learning just from trajectories and can be directly learned from data collected in the real-world.

7. Discussion

The SPT model is designed to learn long-range spatial planning and it outperforms the baselines consistently across multiple experimental settings on both navigation and manipulation tasks. End-to-end learning experiments demonstrate that the SPT model can deal with unknown maps by learning mapping without any map-supervision, highlighting one key benefit over classical planning algorithms. SPT also offers runtime performance benefits over classical planners. We showed that the SPT model scales much better with increasing map sizes as compared to the baselines, however larger map sizes lead to higher memory requirements. In the future, the recent advances in Transformer architectures can be used to improve the memory efficiency of SPTs, for example via hashing of keys and query (Kitaev et al., 2020), or using a low-rank approximation to linearize attention (Shen et al., 2021). Another way of tackling larger maps is increasing the discretization to larger cells and using low-level controllers to navigate between cells. SPT model can also be used as a learned value function for sampling in classical planning algorithms instead of heuristics.

License for Gibson dataset: http://svl.stanford.edu/gibson2/assets/GDS_agreement.pdf

References

- Amos, B., Jimenez, I., Sacks, J., Boots, B., and Kolter, J. Z. Differentiable mpc for end-to-end planning and control. In *NeurIPS*, 2018.
- Bellman, R. Dynamic programming. *Science*, 1966.
- Bertsekas, D. P., Bertsekas, D. P., Bertsekas, D. P., and Bertsekas, D. P. *Dynamic programming and optimal control*. Athena scientific Belmont, MA, 1995.
- Bhardwaj, M., Boots, B., and Mukadam, M. Differentiable gaussian process motion planning. In *International Conference on Robotics and Automation (ICRA)*, 2020.
- Bottou, L. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*. Springer, 2010.
- Canny, J. *The complexity of robot motion planning*. MIT press, 1988.
- Chaplot, D. S., Gandhi, D., Gupta, A., and Salakhutdinov, R. Object goal navigation using goal-oriented semantic exploration. In *In Neural Information Processing Systems*, 2020a.
- Chaplot, D. S., Gandhi, D., Gupta, S., Gupta, A., and Salakhutdinov, R. Learning to Explore using Active Neural SLAM. In *ICLR*, 2020b.
- Chaplot, D. S., Salakhutdinov, R., Gupta, A., and Gupta, S. Neural topological slam for visual navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12875–12884, 2020c.
- Chen, B., Dai, B., Lin, Q., Ye, G., Liu, H., and Song, L. Learning to plan in high dimensions via neural exploration-exploitation trees. In *ICLR*, 2020.
- Dijkstra, E. W. et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1959.
- Durrant-Whyte, H. and Bailey, T. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 2006.
- Elfes, A. Sonar-based real-world mapping and navigation. *IEEE Journal on Robotics and Automation*, 1987.
- Elfes, A. Using occupancy grids for mobile robot perception and navigation. *Computer*, 1989.
- Eysenbach, B., Salakhutdinov, R. R., and Levine, S. Search on the replay buffer: Bridging planning and reinforcement learning. In *NeurIPS*, 2019.
- Farquhar, G., Rocktäschel, T., Igl, M., and Whiteson, S. Treeqn and atrec: Differentiable tree-structured models for deep reinforcement learning. In *ICLR*, 2018.
- Fraundorfer, F., Heng, L., Honegger, D., Lee, G. H., Meier, L., Tanskanen, P., and Pollefeys, M. Vision-based autonomous mapping and exploration using a quadrotor mav. In *International Conference on Intelligent Robots and Systems*, 2012.
- Fuentes-Pacheco, J., Ruiz-Ascencio, J., and Rendón-Mancha, J. M. Visual simultaneous localization and mapping: a survey. *Artificial intelligence review*, 2015.
- Glasmachers, T. Limits of end-to-end learning. In *Asian Conference on Machine Learning*, 2017.
- Golledge, R. G., Dougherty, V., and Bell, S. Acquiring spatial knowledge: Survey versus route-based knowledge in unfamiliar environments. *Annals of the association of American geographers*, 1995.
- Gregor, K., Jimenez Rezende, D., Besse, F., Wu, Y., Merzic, H., and van den Oord, A. Shaping belief states with generative environment models for rl. *Advances in Neural Information Processing Systems*, 32:13475–13487, 2019.
- Guez, A., Weber, T., Antonoglou, I., Simonyan, K., Vinyals, O., Wierstra, D., Munos, R., and Silver, D. Learning to search with mctsnets. In *International Conference on Machine Learning*, pp. 1822–1831, 2018.
- Gupta, S., Davidson, J., Levine, S., Sukthankar, R., and Malik, J. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
- Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. In *ICML*, 2019.
- Hart, P. E., Nilsson, N. J., and Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.
- Hu, J., Ozay, M., Zhang, Y., and Okatani, T. Revisiting single image depth estimation: Toward higher resolution maps with accurate object boundaries. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 1043–1051. IEEE, 2019.
- Ichter, B., Harrison, J., and Pavone, M. Learning sampling distributions for robot motion planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7087–7094. IEEE, 2018.
- Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P., and Schaal, S. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 2013.

- Ilin, R., Kozma, R., and Werbos, P. J. Efficient learning in cellular simultaneous recurrent neural networks—the case of maze navigation problem. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, 2007.
- Karaman, S. and Frazzoli, E. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 2011.
- Karkus, P., Hsu, D., and Lee, W. S. Qmdp-net: Deep learning for planning under partial observability. In *Advances in Neural Information Processing Systems*, 2017.
- Karkus, P., Angelova, A., Vanhoucke, V., and Jonschkowski, R. Differentiable mapping networks: Learning structured map representations for sparse visual localization. In *International Conference on Robotics and Automation (ICRA)*, 2020.
- Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 1996.
- Khan, A., Zhang, C., Atanasov, N., Karydis, K., Kumar, V., and Lee, D. D. Memory augmented control networks. In *ICLR*, 2018.
- Khan, A., Ribeiro, A., Kumar, V., and Francis, A. G. Graph neural networks for motion planning. *arXiv preprint arXiv:2006.06248*, 2020.
- Khatib, O. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*. Springer, 1986.
- Kitaev, N., Kaiser, Ł., and Levskaya, A. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020.
- Konolige, K., Bowman, J., Chen, J., Mihelich, P., Calonder, M., Lepetit, V., and Fua, P. View-based maps. *The International Journal of Robotics Research*, 2010.
- Laskin, M., Emmons, S., Jain, A., Kurutach, T., Abbeel, P., and Pathak, D. Sparse graphical memory for robust planning. In *NeurIPS*, 2020.
- LaValle, S. M. and Kuffner Jr, J. J. Randomized kinodynamic planning. *The international journal of robotics research*, 2001.
- LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.
- Lee, L., Parisotto, E., Chafflot, D. S., Xing, E., and Salakhutdinov, R. Gated path planning networks. In *ICML*, 2018.
- Lozano-Perez, T. Spatial planning: A configuration space approach. In *Autonomous robot vehicles*. Springer, 1990.
- Mishkin, D., Dosovitskiy, A., and Koltun, V. Benchmarking classic and learned navigation in complex 3d environments. *arXiv preprint arXiv:1901.10915*, 2019.
- Nardelli, N., Synnaeve, G., Lin, Z., Kohli, P., Torr, P. H., and Usunier, N. Value propagation networks. 2019.
- Oh, J., Singh, S., and Lee, H. Value prediction network. In *Advances in Neural Information Processing Systems*, 2017.
- Parisotto, E. and Salakhutdinov, R. Neural map: Structured memory for deep reinforcement learning. In *ICLR*, 2018.
- Pathak, D., Mahmoudieh, P., Luo, G., Agrawal, P., Chen, D., Shentu, Y., Shelhamer, E., Malik, J., Efros, A. A., and Darrell, T. Zero-shot visual imitation. In *ICLR*, 2018.
- Qureshi, A. H., Simeonov, A., Bency, M. J., and Yip, M. C. Motion planning networks. In *International Conference on Robotics and Automation (ICRA)*, 2019.
- Qureshi, A. H., Dong, J., Choe, A., and Yip, M. C. Neural manipulation planning on constraint manifolds. *IEEE Robotics and Automation Letters*, 5(4):6089–6096, 2020.
- Racanière, S., Weber, T., Reichert, D., Buesing, L., Guez, A., Rezende, D. J., Badia, A. P., Vinyals, O., Heess, N., Li, Y., et al. Imagination-augmented agents for deep reinforcement learning. In *NIPS*, 2017.
- Ratliff, N. D., Issac, J., Kappler, D., Birchfield, S., and Fox, D. Riemannian motion policies. *arXiv preprint arXiv:1801.02854*, 2018.
- Satorras, V. G. and Welling, M. Neural enhanced belief propagation on factor graphs. In *International Conference on Artificial Intelligence and Statistics*, 2021.
- Savinov, N., Dosovitskiy, A., and Koltun, V. Semi-parametric topological memory for navigation. In *ICLR*, 2018.
- Savva, M., Kadian, A., Maksymets, O., Zhao, Y., Wijmans, E., Jain, B., Straub, J., Liu, J., Koltun, V., Malik, J., et al. Habitat: A platform for embodied ai research. In *ICCV*, 2019.
- Schleich, D., Klamt, T., and Behnke, S. Value iteration networks on multiple levels of abstraction. In *Robotics: Science and Systems*, 2019.
- Sekar, R., Rybkin, O., Daniilidis, K., Abbeel, P., Hafner, D., and Pathak, D. Planning to explore via self-supervised world models. In *ICML*, 2019.

- Shen, Z., Zhang, M., Zhao, H., Yi, S., and Li, H. Efficient attention: Attention with linear complexities. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 3531–3539, 2021.
- Silver, D., Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., et al. The predictron: End-to-end learning and planning. In *International Conference on Machine Learning*. PMLR, 2017.
- Srinivas, A., Jabri, A., Abbeel, P., Levine, S., and Finn, C. Universal planning networks. *ICML*, 2018.
- Tamar, A., Wu, Y., Thomas, G., Levine, S., and Abbeel, P. Value iteration networks. In *NIPS*, 2016.
- Tolman, E. C. Cognitive maps in rats and men. *Psychological review*, 1948.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *NIPS*, 2017.
- Xia, F., R. Zamir, A., He, Z.-Y., Sax, A., Malik, J., and Savarese, S. Gibson Env: real-world perception for embodied agents. In *CVPR*, 2018.
- Zhang, J., Tai, L., Boedecker, J., Burgard, W., and Liu, M. Neural slam: Learning to explore with external memory. *arXiv preprint arXiv:1706.09520*, 2017.
- Zhu, Y., Gordon, D., Kolve, E., Fox, D., Fei-Fei, L., Gupta, A., Mottaghi, R., and Farhadi, A. Visual semantic planning using deep successor representations. In *ICCV*, 2017.

A. Background: Transformers

The proposed spatial planning method is based on the Transformer model (Vaswani et al., 2017). A Transformer layer, denoted by f_{TL} , takes a tensor $X \in \mathcal{R}^{d \times S}$ as input, where d is the embedding size and S is the size of the input. It consists of two sublayers, a multi-head self-attention layer (f_{SA}) and a position-wise fully connected layer (f_{FC}). There is a residual connection around each sublayer, followed by layer normalization (Ba et al., 2016) (LN):

$$R = \text{LN}(f_{\text{SA}}(X) + X), Y = f_{\text{TL}}(X) = \text{LN}(f_{\text{FC}}(R) + R)$$

where $R, Y \in \mathcal{R}^{d \times S}$ are the intermediate and final representations, respectively.

The multi-head self-attention (f_{SA}) layer has h attention heads, each computes a scaled dot-product attention over queries Q , keys K and values V , which are all different projections of the input X :

$$Q_i = W_{Q,i}^T X, \quad K_i = W_{K,i}^T X, \quad V = W_{V,i}^T X$$

$$Z_i = \text{Attention}(Q_i, K_i, V_i) = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i$$

where $Q, K \in \mathcal{R}^{d_k \times S}$, $V \in \mathcal{R}^{d_v \times S}$, $i \in 1, 2, \dots, h$ d_k and d_v are hyper-parameters and all W s are parameters. The output of all attention heads, Z_i s, are concatenated and projected to the same dimension as the input. Finally, the position-wise fully connected (f_{FC}) layer applies two linear transformations to each position with a ReLU activation to the output of the multi-head attention.

B. Dataset Details

We generate synthetic datasets for training the spatial planning models for both navigation and manipulation settings. For the navigation setting, we perform experiments with $M \times M$ maps with two different map sizes, $M \in \{15, 30\}$. We randomly generate $o_{\min} = 0$ to $o_{\max} = 5$ obstacles in each map, where each obstacle is a rectangle at a random location with each side being a random length from 1 to $M/2$. All the rectangular obstacles are rotated in two random orientations.

For the manipulation setting, we consider a reacher task using a planar arm with 2 degrees of freedom. We use an operational space of size $P \times P$. Each link of the arm is of size $P/4$. The arm is centered at the center of the operational space. Let the orientation of two links be denoted by θ_1 and θ_2 . We assume both the links can freely rotate in a plane, $\theta_1, \theta_2 \in [0, 2\pi)$. For each environment, we generate $o_{\min} = 0$ to $o_{\max} = 5$ circular obstacles centered at a random location $0.25P$ to $0.75P$ distance away from the center, with a random radius between $0.05P$ and $D - 0.15P$ where D is the distance of the center of the obstacle from the center

of the operational space. We convert each environment to a configuration space map of size $M \times M$, where each cell (i, j) denotes whether the arm will collide with an obstacle when $\theta_1 = 2\pi i/M$ and $\theta_2 = 2\pi j/M$. We experiment with two map sizes, $M \in \{18, 36\}$, corresponding to 20° and 10° bins for each link. The choice of P does not affect the map as the collision check for each cell in the configuration space is performed in the continuous operational space where all distances are relative to P .

C. Navigation Mapper Architecture Details

The Navigation Mapper module predicts a single value between 0 and 1 for each image in o indicating whether the cell in the front of the image is an obstacle or not. The architecture of the Navigation mapper consists of ResNet18 convolutional layers followed by 3 fully-connected layers of size 256, 128, and 1 as shown in Figure 8. Each cell can have up to 4 predictions (from images corresponding to the four neighboring cells facing the current cell), which are aggregated using max-pooling to get a single prediction.

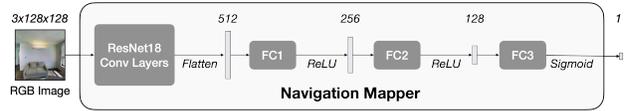


Figure 8. **Navigation Mapper Architecture.** Figure showing the architecture of the Navigation Mapper.

D. Attention Visualization

We show the visualization of attention maps corresponding to two different locations in Figure 9. Interestingly, we noticed three consistent patterns: a) at least one of the attention head out of eight captures obstacles (left), b) one of the attention heads focuses on goal location (middle), and c) some attention maps focus on nearby obstacles to get accurate planning distance (right).

E. Examples

We show additional examples for navigation task for in-distribution test set (in Figure 10), out-of-distribution More Obstacles test set (in Figure 11) and Real-World test set (in Figure 12) each with map size $M = 30$. Additional examples for manipulation task are shown for in-distribution test set (in Figure 13) and for out-of-distribution More Obstacles test set (in Figure 14).

We also visualize examples for the end-to-end mapping and planning experiments for the manipulation task. We show examples of map and action distance predictions using the SPT model trained with dense and perfect supervision in Figure 15 and with noisy and sparse supervision in Figure 16.

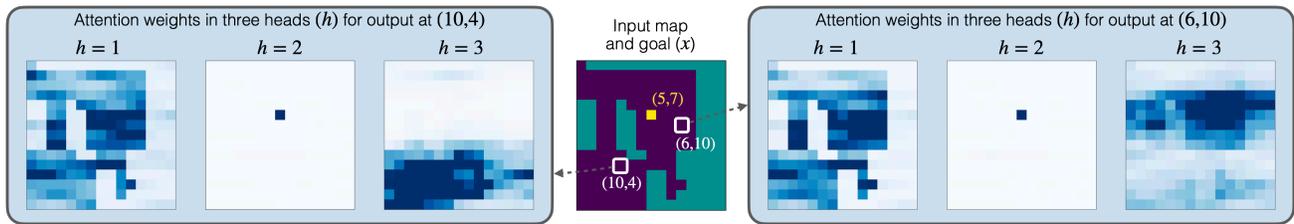


Figure 9. **Attention Visualization.** Visualization of the attention heads learned by Spatial Planning Transformers. SPTs learn an attention for each location in the map with respect to every other location.

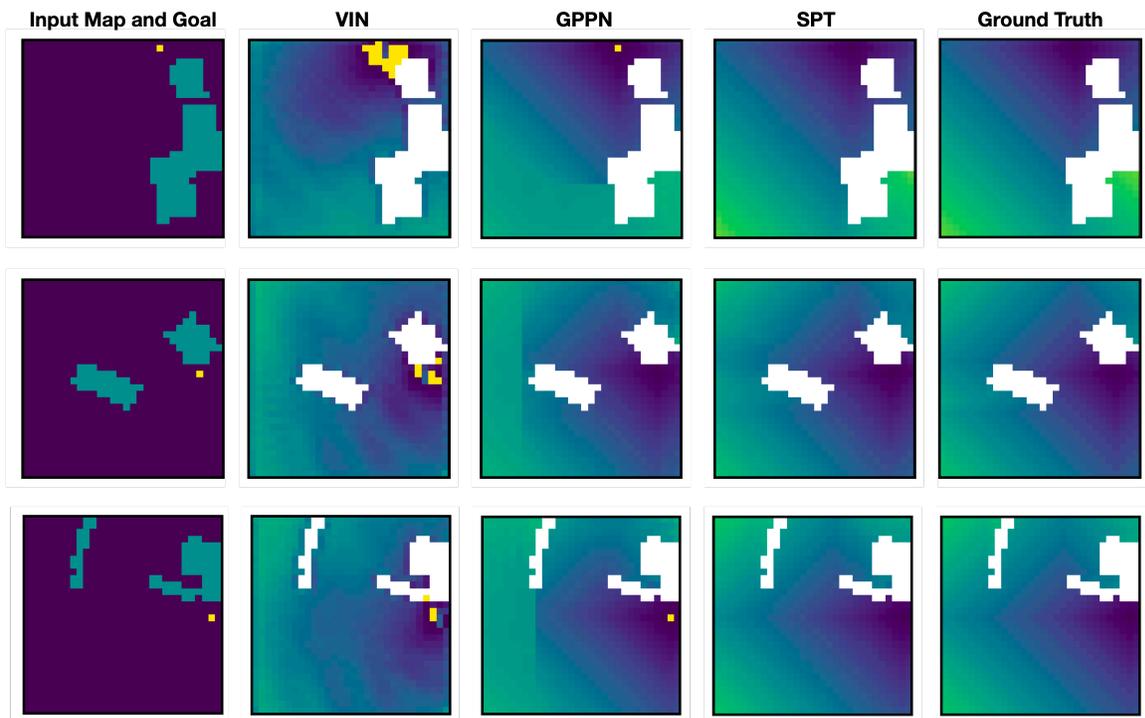


Figure 10. **Navigation in-distribution test set examples.** Figure showing 3 examples of the input, the predictions using the proposed SPT model and the baselines, and the ground truth for the Navigation in-distribution test set for map size $M = 30$.

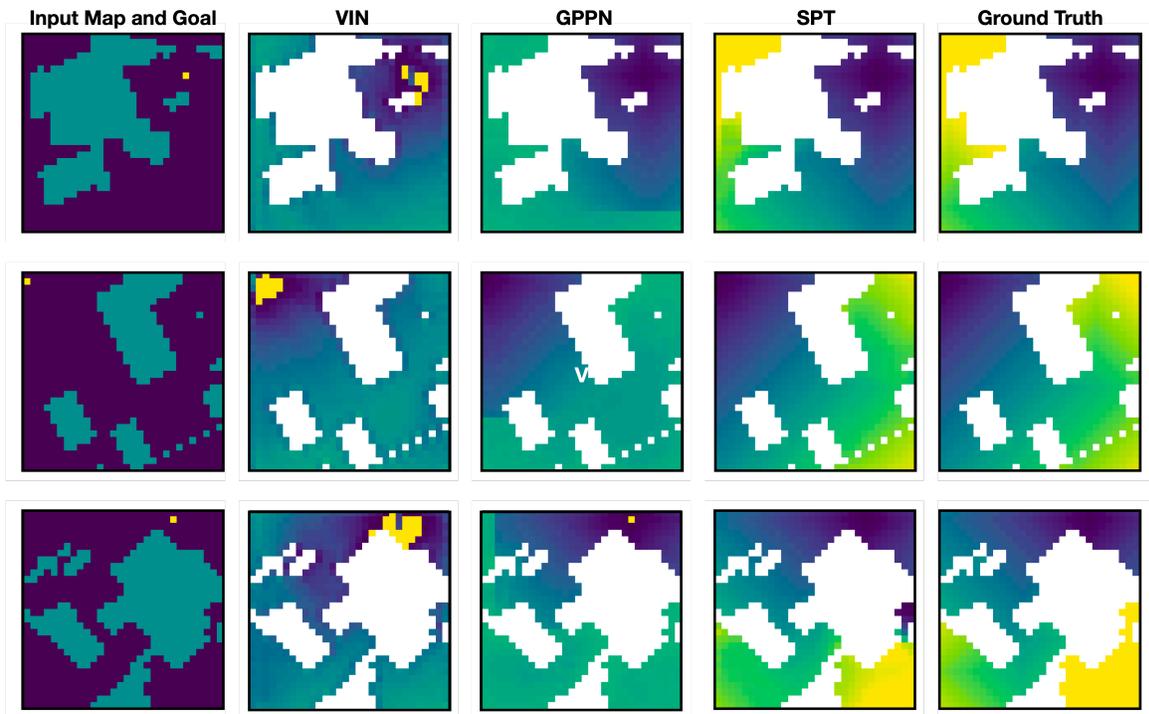


Figure 11. **Navigation out-of-distribution More Obstacles test set examples.** Figure showing 3 examples of the input, the predictions using the proposed SPT model and the baselines, and the ground truth for the Navigation out-of-distribution More Obstacles test set for map size $M = 30$.

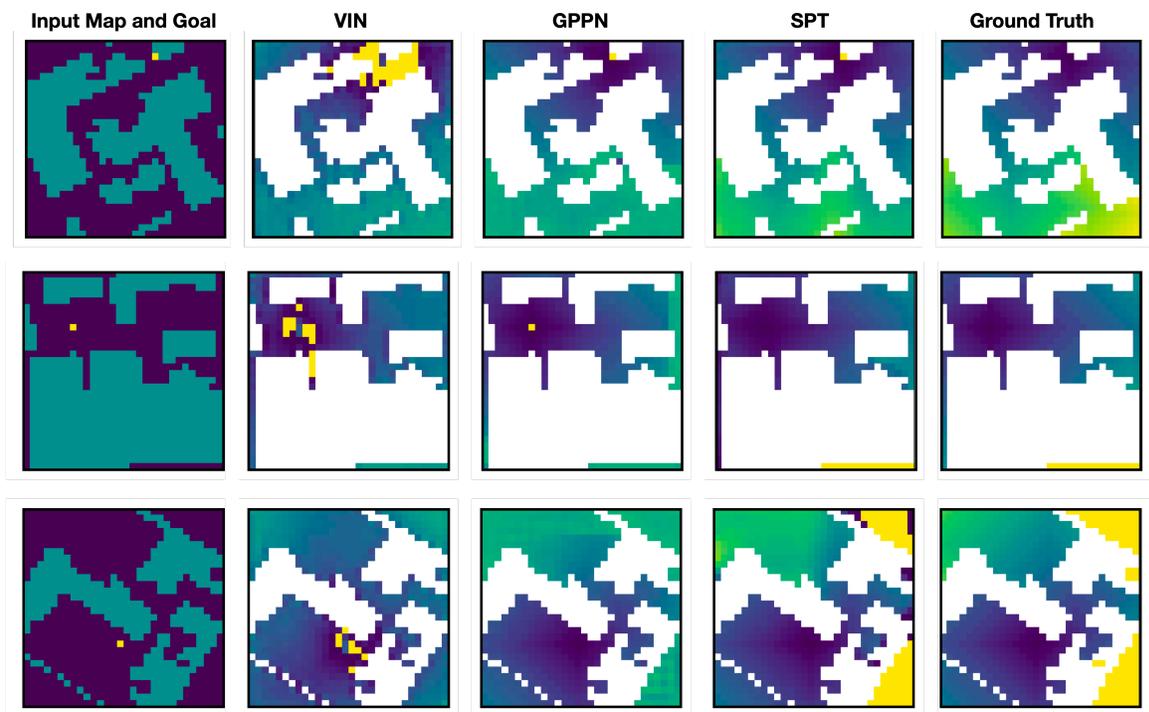


Figure 12. **Navigation out-of-distribution Real-World test set examples.** Figure showing 3 examples of the input, the predictions using the proposed SPT model and the baselines, and the ground truth for the Navigation out-of-distribution Real-World test set for map size $M = 30$.

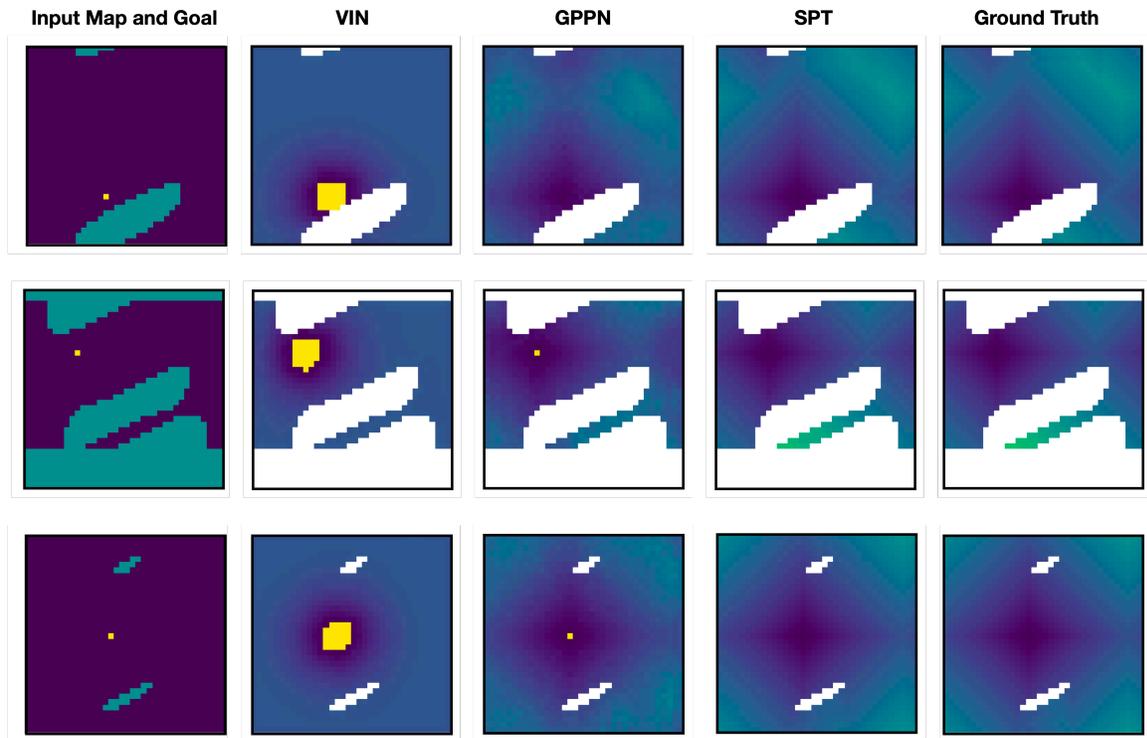


Figure 13. **Manipulation in-distribution test set examples.** Figure showing 3 examples of the input, the predictions using the proposed SPT model and the baselines, and the ground truth for the Manipulation in-distribution test set for map size $M = 36$.

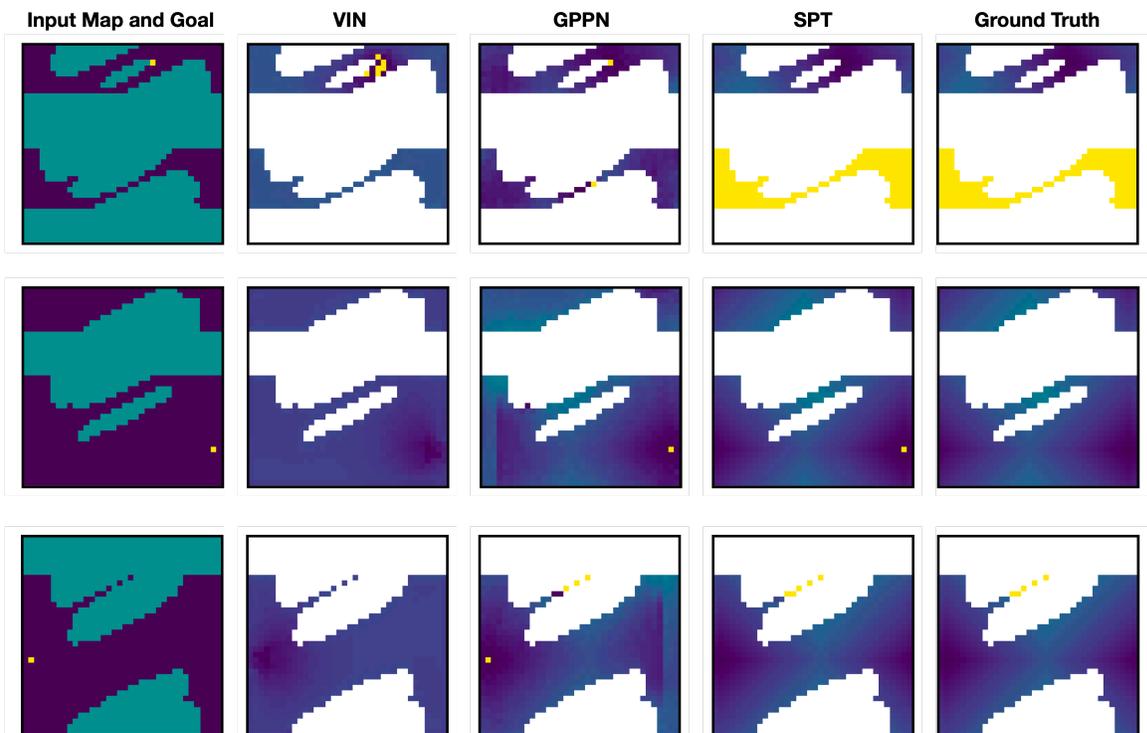


Figure 14. **Manipulation out-of-distribution More Obstacles test set examples.** Figure showing 3 examples of the input, the predictions using the proposed SPT model and the baselines, and the ground truth for the Manipulation out-of-distribution More Obstacles test set for map size $M = 36$.

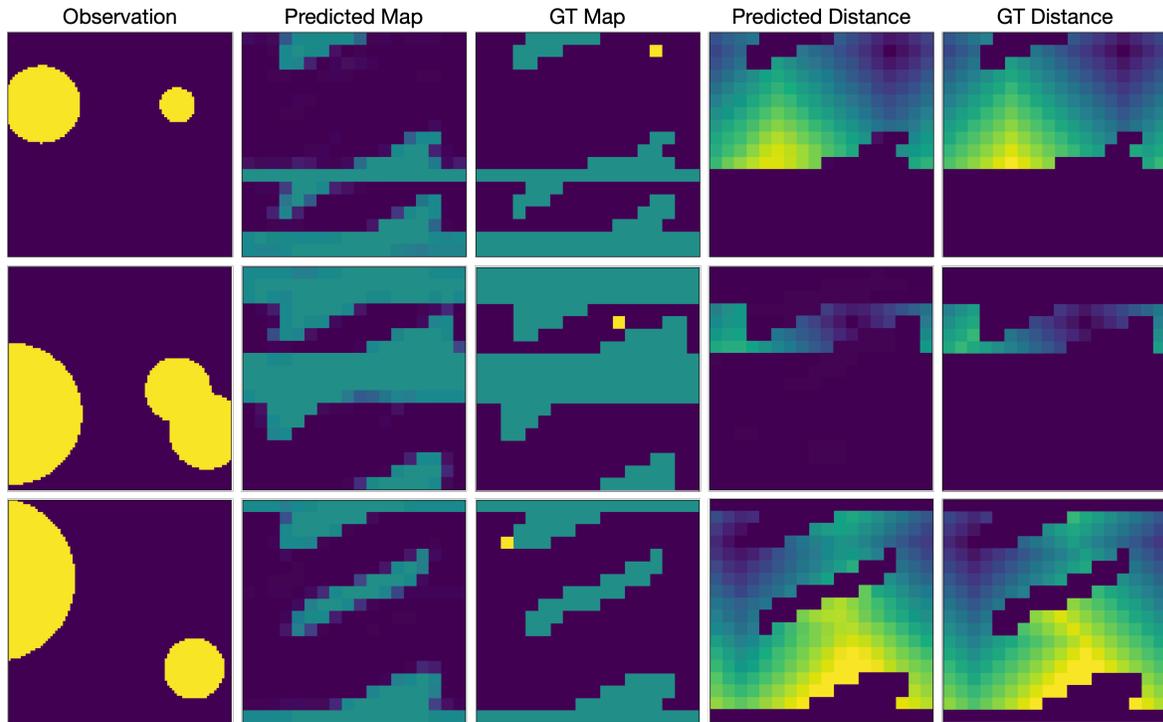


Figure 15. **Dense and Perfect Supervision.** Figure showing examples of map and distance predictions using the SPT model trained with dense and perfect action-level supervision.

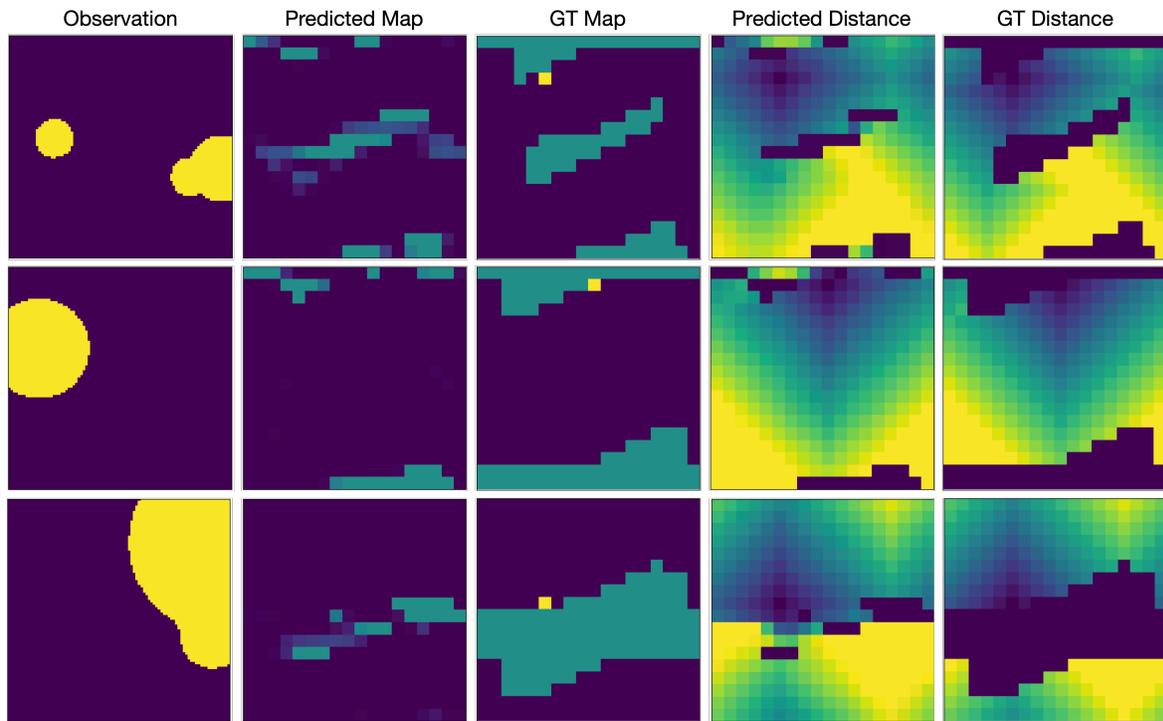


Figure 16. **Sparse and Noisy Supervision.** Figure showing examples of map and distance predictions using the SPT model trained with sparse and noisy action-level supervision.