Congestion Prediction for Large Fleets of Mobile Robots

Ge Yu[1] and Michael T. Wolf[2]

***Abstract*— This paper introduces a deep learning (DL) approach to predicting congestion delays in large multi-robot systems. The problem is motivated by real-world problems in modern logistics automation, such as a warehouse with hundreds to thousands of coordinated mobile robots. Here, the large scale, the complexity of the control software, and the uncertainties of the robots' dynamics make direct (simulated) prediction of future robot states impractical. We propose predicting delays associated with future spatiotemporal locations, and we show this is useful for improving system performance via incorporating the predictions into path planning and travel time estimation. Our DL model uses convolutional long short-term memory (ConvLSTM) as the core structure, takes the historical congestion condition and planned paths as input, and generates the delays across all nodes in the spatial planning graph for a set of future time windows. When using predictions in a modified path planner, simulation experiments using production data show 4.4% average improvement in throughput performance versus without predictions.**

I. Introduction

This paper introduces a new method for predicting spatiotemporal congestion in large multi-robot systems where future agent states cannot be reliably calculated. Examples of such systems exist in modern logistics buildings such as automated warehouses and are envisioned for networked autonomous mobility-on-demand (AMoD). Congestion prediction is useful in these cases both for directly improving time-optimal path planning decisions and for estimating an agent's expected travel time, which in turn is important in task assignment decisions.

In particular, we consider the context of a mobile robot fleet transporting material in a warehouse: agents must transport payloads from a set of pickup locations to a set of destinations, with these tasks arriving on a rolling basis. Applications include moving inventory shelves for order fulfillment [1] or sorting packages by geographic destination [2]. For example, Figure 1 illustrates a *sortation center floor*, where packages are placed on mobile robots in stations along the exterior of the floor, and each robot delivers its package to a specific interior destination port (based on postal code). Here, each robot continually alternates between a delivery task (transporting a package to its destination) and a pickup task (returning to the stations to acquire a new package), and each task requires a new path plan from the start position to the task end position. System performance is primarily measured by *throughput*, which is the number of tasks completed per unit time. Note this problem is similar to the lifelong multi-agent pickup and delivery (MAPD) problem [3], which is also studied for large-scale warehouses [4], except our simulations are continuous-time and use realistic robot kinematics[1] rather than discrete stepwise motions.

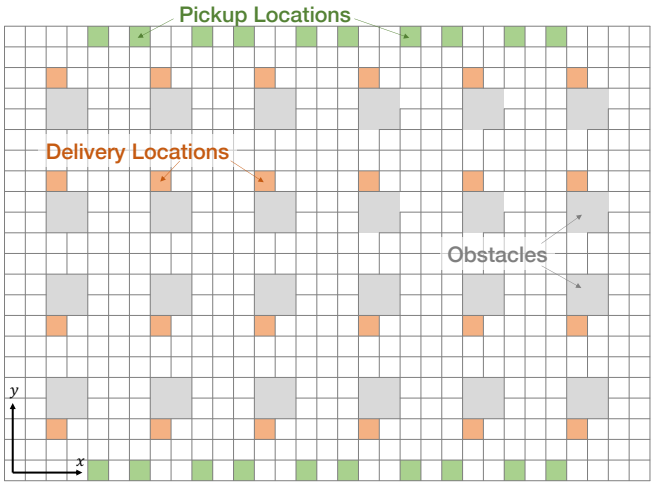

Fig. 1. Example sortation floor. Robots acquire payloads at pickup locations (green cells) and must transport them to specified destination ports (orange cells). The environment has a set of static obstacles (gray cells).

The number of agents ($O(1000)$ robots) makes direct, real-time solutions to multi-agent trajectory planning computationally infeasible (see Sect. II). (Actual sortation floor sizes are typically much larger than shown in Fig. 1.) Further, real-world uncertainties in how these paths will be executed quickly compound, and this uncertainty is amplified at each intersection in the map. Thus, it is futile to "simulate forward" to estimate robot positions beyond several seconds.

While we can neither plan nor predict an individual agent's precise (x, y) position at time t in these large-scale systems, we propose to predict collective measures that are useful to improve system performance. In particular, we focus on the *time delay* that robots should expect to experience in a given spatiotemporal position. Informally, we call this *congestion prediction*, since the time delays result from multiple robots attempting to occupy the same space.

Our contribution is a congestion prediction approach that uses deep learning techniques, with features from recent robot position histories and the currently planned paths of agents (see Figure 2). We design a novel representation of paths as a time series of expected occupancy maps. The models explicitly forecast a time delay at a given (x, y) position in a time window, which can be used for time-

[1]Ge Yu is an Applied Scientist in Amazon Robotics, yugegrace@gmail.com

[2]Michael Wolf is a Principal Applied Scientist in Amazon Robotics, wolfmike@amazon.com

[1]Robot kinematics include acceleration, deceleration, max speed and turning speeds.

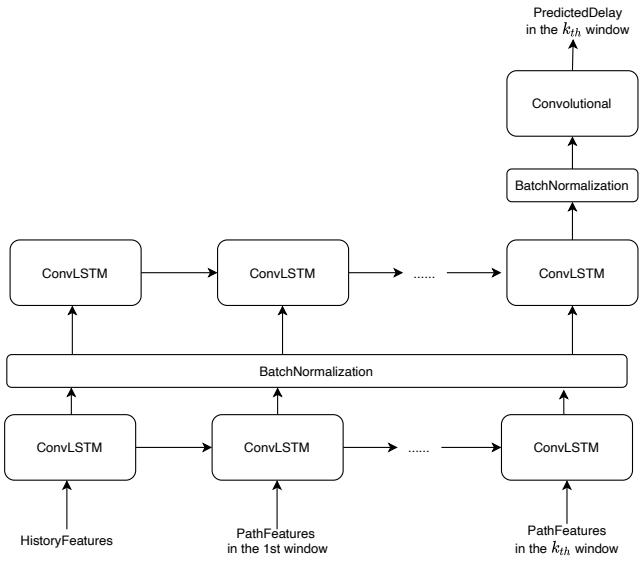

Fig. 2. Prediction model architecture. The two ConvLSTM layers have a convolution kernel of size 5 and size 3, respectively. The output layer is a separable convolution layer with Relu activation to generate spatially smooth prediction results. See Supplementary file for details.

optimal path planning and travel time estimation. We showed that applying the predicted congestion delays as time-varying costs in a single-agent planner resulted in a 4.4% throughput improvement. Using the delays also reduced average RMSE on travel time estimation by 48.6%.

This paper is organized as follows: Section II lists related work and compares our work to existing literature. Section III describes the congestion delay prediction problem and the architecture of ConvLSTM models. Section IV describes two downstream applications of the predicted delays. Section V provides our data sources, simulation setup, and evaluation results. Section VI concludes the paper with future directions to extend our work.

II. Related Work

Neural networks have been widely adopted in time-series prediction problems [5]. The prediction problem considered in our work takes three-dimensional (spatial x, y and temporal t) data as input and generates a delay map as output. A similar setting is the traffic congestion prediction problems, where spatial and temporal relationships are exploited to make future predictions. [6] proposes spatial-temporal dynamic networks to predict traffic volume, where CNN and LSTM structures are applied. [7] proposed PCNN to predict short-term traffic congestion by levels, folding the time-series as a two-dimensional matrix before applying convolution networks. [8] proposes a network structure combining CNN, LSTM and transpose CNN to predict traffic congestion using snapshots of city traffic. [9] proposes a model with GCN and LSTM for traffic forecasting on general graph structure using history traffic patterns. The work that is most similar to our problem setting is by [10] and [11]. [10] uses GPS trajectory data to predict average drive speed in road networks. They found that LSTM performs well in predicting long-term series and CNN is able to learn the spatial features. [11] constructs congestion propagation pattern graphs first, and then applies ConvLSTM on spatial matrices to predict congestion levels. Our work differs from these in that our problem setup has robots moving between equally spaced cells on a rigid floor, with accurate localization measurements. Additionally, our prediction horizon is required to be comparable to robot path execution time (in minute range) and the resolution is required to be in the same scale as the atomic robot movements (in second range).

Recent works have introduced machine learning techniques into path planning problems. [12] applies deep learning techniques in heuristic function learning for path planning algorithms like A* and D*. [13] proposes a decentralized multi-agent path planning algorithm for robots to navigate using their local views and communicate through a graph neural network. We use deep learning techniques to produce an additional cost that can be consumed by arbitrary planning algorithms.

For path planning problems in large-scale robot fleets, multi-agent path finding (MAPF) algorithms have been an active research topic. Classical MAPF algorithms include conflict-based search [14], where conflicts are resolved after individual optimal paths are found, and priority-based search [15], where a priority order is enforced among agents before individual optimal path is searched. [16] proposes a cooperative A* algorithm to avoid collisions for multiple agents. They use a stop action when a location is reserved by a robot and test up to 100 mobile robots. [17] considers path planning with uncertain costs. The "rolling task" nature of our warehouse applications is better captured in the so-called *lifelong* multi-agent path planning problem [3], [4], [18]. However, two main issues still exist for our applications of interest in all the above. First, MAPF algorithms are typically computationally intensive and have difficulty scaling to thousands of robots in real-time systems. Second, the MAPF formulation simplifies the problem into one of moving point robots between adjacent nodes in synchronized discrete time steps (essentially a pebble motion problem [19]); our applications' conditions demand accounting for realistic robot kinematics and the uncertainties of trajectory execution.

III. Congestion Delay Prediction

We assume a uniform, square cellular decomposition of the workspace such as in Fig. 1, where the cell size is selected to allow a maximum of one agent in each cell and agents may exist at neighboring cells. It will be convenient to consider the rectangle circumscribed by the workspace—let N_x and N_y denote the number of cells along the x and y axis, respectively, in this rectangle. We then define a graph $G = (V, E)$ over the workspace such that there exists a vertex $v_{i,j} \in V$ for each cell (where i is the index along the x-axis and j is the index along the y-axis), and the directed edges E represent admissible transitions between adjacent vertices. Note that cells containing static obstacles are forbidden to the agents and thus not connected by edges.

Each edge has a static weight proportional to its idealized travel time cost.

A set of n agents simultaneously occupy the graph, each of which has a task and a corresponding path plan to the task's destination[2]. Note that some agents may have very short path plans, especially if they are near their destination. There is no information about any future tasks of agents. Also, it is allowable for paths to be re-planned during execution, adding uncertainty to the future state of a given agent. We assume the current (x, y, θ) poses and path plans of all agents are known, where $\theta \in [0, 2\pi)$ is the heading of an agent.[3]

Consider a receding time horizon up to T_F into the future, discretized into M equal-duration, non-overlapping time windows. Let T_w denote the duration of each time window ($T_F = T_w \cdot M$). Then the sequence of time windows under consideration is $\{[t_0+(k-1)*T_w, t_0+k*T_w-1]\}_{k=1}^{M}$, where t_0 is the current time. We wish to predict delays that may be experienced if an agent attempts to route through a particular vertex during a time window, where *delay* is defined as the additional travel time experienced by an agent during path execution compared to free-flow motion with idealized kinematics. Delays are associated to the vertex of G where the agent experiences additional time; an agent is considered "at a vertex" if it has not completed the edge traversal to the next vertex. Then *aggregated delay* on a vertex during a time window is defined as the sum of all delays experienced by any robot going through the vertex during the time window. Note that aggregated delays have an approximate interpretation as the duration a vertex is occupied during a time window. The goal is to train a set of models to predict the aggregated delays on the floor for the M time windows.

We represent both input features to our prediction models as well as aggregated delays on the floor in the format of *floor images* (see Figure 3). A floor image is a $N_x \times N_y \times N_f$ matrix, where N_f is the number of channels. The floor images for input features are "*feature images*" and the floor images for aggregated delays are "*delay images*". Additionally, a set of feature images is ordered based on their time index to form a sequence, which is the input to our deep learning prediction models (Sect. III-A). Then, the prediction problem becomes a multi-step time-series prediction problem: Given a time series of feature images, predict the future M delay images.

A. Feature Generation

We use two categories of features: history features, which are derived from robot locations up to the current time point t_0, and path features, which are derived from the planned paths for all robots at time t_0. Our ablation studies indicated both feature types improve performance (see Supplementary file). This matches an intuition that the feature types carry

[2]Idle agents are also allowed in practice and are considered with empty tasks and empty path plans.

[3]Drives localize themselves with reference to fiducials laid out on the floor. Their paths are planned and published by a cloud service on a per-agent basis.

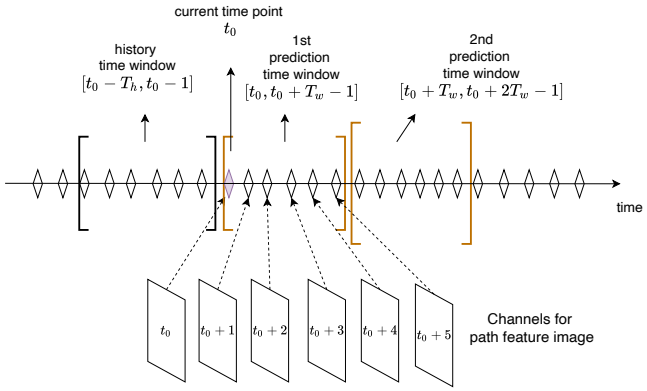

Fig. 3. History time window and prediction time windows with respect to the current time point. All of these time windows are a collection of consecutive discrete time points.

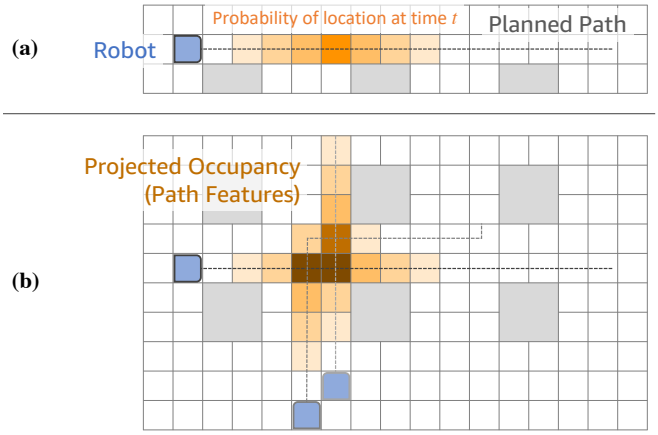

Fig. 4. Generating path features from planned paths. (a) A single robot's future location along its planned path. (b) Individual probabilistic occupancies are summed from multiple robots to serve as path features.

complementary information—history features relate to the "inertia" of congestion patterns and path features inform future movement of robots.

History features are binary features: let $H_t \in \mathbb{R}^{N_x \times N_y}$ denote the history feature at time t, with elements denoted by $h_{i,j}$. Then $h_{i,j} = 1$ if there is any robot experiencing non-zero delay on vertex $v_{i,j}$; $h_{i,j} = 0$ otherwise. We compute history feature matrix H_t for a set of discrete time points $t \in \{t_0-T_h, t_0-T_h+1, \ldots, t_0-1\}$, where the time interval $[t_0-T_h, t_0-1]$ is referred to as the *history time window* (see Figure 3). These binary history features in the history time window are ordered by the time point and stacked along the last dimension to formulate the the history feature image $FH_{t_0} = [H_{t_0-T_h}, H_{t_0-T_h+1}, \ldots, H_{t_0-1}] \in \mathbb{R}^{N_x \times N_y \times T_h}$, where T_h is the number of channels for history feature images.

Path features are generated from the current planned paths for all agents. We model a trajectory for each agent by projecting its planned path forward using a probabilistic occupancy. For each discrete future time point t, each agent's expected location is calculated by applying a kinematic model along its planned path. Since this projection intentionally considers only a single agent, the actual location

can deviate significantly from the expected one even after a few vertices. In our case, this uncertainty is modeled well by a Gaussian distribution, with a standard deviation that grows linearly with time (see Supplementary file for more detail). Applying this Gaussian distribution to the vertices in the agent's planned path gives a discrete probability of vertex occupancy at time t, as depicted in Fig. 4(a). Note that, for larger t, an agent's path may be completed and thus have no contribution to expected occupancy.

Let $O_t \in \mathbb{R}^{N_x \times N_y}$ denote the path feature at time t and $o_{i,j}$ denote its element. Then $o_{i,j}$ is defined as the expected number of robots that will (attempt to) occupy the vertex $v_{i,j}$ at time t, illustrated in Fig. 4(b). Note that $o_{i,j}$ ignores inter-robot collisions and can be fractional and greater than 1. For the k^{th} time window $[t_0 + (k-1) * T_w, t_0 + k * T_w - 1]$, we order path features in this time window by their time points and stack them along the last dimension to formulate the path feature image (see Figure 3). Let $P^k_{t_0}$ denote the path feature for the k^{th} time window at current time point t_0. Then $P^k_{t_0} = [O_{t_0+(k-1)*T_w}, O_{t_0+(k-1)*T_w+1}, \ldots, O_{t_0+k*T_w-1}] \in \mathbb{R}^{N_x \times N_y \times T_w}$, where T_w is the number of channels.

For the k^{th} prediction time window, input features are constructed by appending path features for the first k prediction time windows to the history feature, $\text{Features}(k, t_0) = [FH_{t_0}, P^1_{t_0}, \ldots, P^k_{t_0}]$. Therefore, all prediction time windows share the same history feature, but the length of the path features grows linearly as the number of time windows increases. For long prediction horizons, adjacent time points can be combined to reduce the dimensions of input features for better scalability.

B. ConvLSTM Models

For multi-step time-series prediction, there are generally two classes of approaches: recursive approaches and direct approaches. Recursive approaches train a shared model for different prediction steps, while direct approaches train a different model for each prediction step. We present our solution based on direct methods for higher accuracy, with a separate model trained for each prediction time window. All of our prediction models share the same model architecture.

The input to the prediction models is a sequence of floor images. We use ConvLSTM [20] as the core layers to exploit both the spatial and temporal relationships within features. Long-short term memory (LSTM) models are a special kind of recurrent neural networks that are known for capturing longer dependencies in sequences. ConvLSTM inherits this advantage from classical LSTM by including an input gate, a memory gate, a forget gate and hidden states in its cell. Additionally, ConvLSTM replaces the matrix multiplications at each gate with a convolutional operation between the gate parameters and the inputs, cell outputs or hidden states. This convoluntional kernel enables the model to capture the spatial relationship first and then exploit the temporal relationship.

Our prediction model has 2 layers of ConvLSTM layers, each followed by a batch normalization layer for regularization purposes (see Figure 2 for the architecture for the k^{th} prediction window). The output layer is a convolutional layer, to generate spatially smooth prediction values, with output dimension (N_x, N_y). The loss function is vertex-wise mean square error, defined in (1):

$$\text{Loss}(Y, D) = \frac{1}{\sum_{i,j} \text{Mask}(i,j)} \sum_{i,j} (y_{i,j} - d_{i,j})^2 \, \text{Mask}(i,j), \tag{1}$$

where $y_{i,j}$ and $d_{i,j}$ are the elements at vertex $v_{i,j}$ for predicted delay image Y and ground truth delay image D, respectively. $\text{Mask}(\cdot,\cdot)$ is a binary vertex mask for filtering out the forbidden vertices. Ground truth delays are computed by comparing the robot's actual trajectories with *ideal* trajectories, with no other robot interactions.

IV. Application of Predicted Delays

A. Arrival Time Estimation with Delay

Accurately estimating an agent's *arrival time* at the task's destination vertex helps the system's decision-making, such as agent task allocation and delivery sequence estimation. Once congestion delays are predicted, we can estimate an agent's remaining travel time simply by summing up the modeled free-flow travel time of the remaining path segment and the predicted delays on these vertices. The free-flow travel time is given by an empirical kinematic model of robot motion, similar to that described for path features in Sect. III-A but explicitly using a *maximum* speed instead of an *expected* executed speed. For the predicted delay, we sum the elements of $\{D^k_{t_0}\}_k$ corresponding to the robot's path, with simple bookkeeping required to select the appropriate index k as the robot proceeds in time, and scale the sum by a hyper-parameter α. This hyper-parameter α accounts for a difference in our prediction assumptions and how we apply the delay for travel time estimation: The predicted delays are collective, and here we use α to approximate the proportion of the delay that will be experienced by a single agent.

B. Path Planning with Delay

Another important application for predicted delays is for improving path planning, which can increase system performance measures such as throughput in pickup and delivery problems. A traditional single-agent planning approach is to use the A* algorithm to search through the planning graph G with static edge weights, to identify the sequence of edges that give the lowest total cost. In our approach, we add time-varying delay cost to the vertices, based on the predicted delay values $\{D^k_{t_0}\}_k$. To integrate the time-varying delay costs, the path searching algorithm must compute the expected arrival time at each vertex along candidate paths as in Sect. IV-A, for bookkeeping which index k to use for the predicted delays. This ensures comparable computational complexity as A* in two-dimensional search space because the planning graph vertices V represent only the x–y space, and neighboring vertex expansion is considered fully dependent on the spatial path. Manhattan distance with kinematic parameters is used as the heuristic for A*, which satisfies the condition that the heuristic always under-estimates the path cost.

V. Simulation Experiments

We evaluated our proposed congestion prediction models using simulation experiments. Our custom industrial simulation environment mocks production software stacks for faithful tracking of all robot states. We measured the root mean square error (RMSE) between the predicted delays and actual delays for each prediction window and examined impact in the two applications detailed in Sect. IV.

A. Simulation Setup and Model Hyperparameters

Our simulation settings are inspired by Amazon sortation center applications, similar to those described in [4]. We consider two different sortation floors with four-connected vertices and bidirectional edges, see Figure 1. The sizes of the two sortation floors are 179-by-69 and 166-by-74 vertices, with around 20% of the vertices covered by obstacles. The number of mobile robots are 810 and 850, respectively.

The incoming package data is from Amazon sortation center production replay of randomly selected dates. A typical distribution of these package destinations is nonuniform, with around 40% of packages going to the top 10% of destination ports and a long tail where only 3% of packages go to the bottom 60% of destination ports. For each sortation floor, we randomly selected 6-8 production dates, referred to as *simulation scenarios*. We first run the simulation for these scenarios with a naive prediction model that predicts future delays as delays from the past 10s and use a single-agent A* path planning algorithm (10 random seeds for each simulation scenario). These simulation runs provide our *baseline* results (referred to as "Hist_base").

From the baseline simulation runs, we randomly pick 4 simulation runs and uniformly sample training and validation data from these runs. Prediction models are trained on an Nvidia GPU using Keras with Tensorflow2 [21]. We set the learning rate as 0.005 and the batch size as 4 (restricted by memory). We predict 6 consecutive time windows of 10 seconds, which spans 60 seconds into the future (the average duration of a sortation task from empirical data is in the range of 60s to 90s). The training usually converges within 15-20 epochs with the training time, increasing as the number of prediction windows increases: for the prediction window 0-10s into the future, the training time is around 30 minutes; for the prediction window 50-60s into the future, the training time is around 1 hour.

B. Prediction Model Evaluation

Table I shows the overall RMSE as well as *categorized RMSE*, where vertices in testing samples are first categorized based on their ground-truth delays and then RMSE in each category is computed independently; e.g., category "GT 0–3 s" includes all vertices with a ground-truth delay in $(0, 3]$ seconds. Table I includes values for our method plus two sets of alternatives—*baselines* that include no learning and *other model structures* using the same input features. The baseline delay estimates are "Zero_base", which sets the delay at every vertex in each time window as zero, and "Hist_base", described above. Our method outperforms both baselines in prediction RMSE, illustrating the efficacy of our feature construction and learning approach.

TABLE I
Delay Prediction RMSE (in seconds).

	Models	Zero_ base	Hist_ base	Our Conv-LSTM	D-GCN LSTM	3GCN LSTM	CNN GCN
Horizon 0-10s	Overall	1.90	1.19	0.92	1.03	1.02	1.03
	GT 0 s	0.00	0.65	0.36	0.41	0.47	0.46
	GT 0–3 s	1.72	2.55	1.83	2.02	2.23	2.16
	GT 3–6 s	4.45	3.81	3.03	3.17	3.26	3.25
	GT 6–9 s	7.54	4.36	3.82	4.14	4.00	4.07
	GT 9–12 s	9.97	3.06	3.05	3.60	3.07	3.15
Horizon 20-30s	Overall	1.91	1.58	1.15	1.35	1.35	1.35
	GT 0 s	0.00	1.05	0.59	0.72	0.65	0.73
	GT 0–3 s	1.72	2.48	1.69	1.71	1.60	1.74
	GT 3–6 s	4.45	3.98	3.12	3.22	3.23	3.23
	GT 6–9 s	7.54	5.26	4.41	4.95	5.09	4.93
	GT 9–12 s	9.97	4.98	4.36	5.45	5.74	5.36
Horizon 50-60s	Overall	1.91	1.72	1.24	1.45	1.45	1.45
	GT 0 s	0.00	1.16	0.69	0.73	0.70	0.73
	GT 0–3 s	1.72	2.48	1.71	1.61	1.60	1.61
	GT 3–6 s	4.45	4.04	3.21	3.34	3.38	3.34
	GT 6–9 s	7.54	5.56	4.60	5.37	5.43	5.37
	GT 9–12 s	9.97	5.59	4.63	6.18	6.24	6.16

Additionally, ConvLSTM outperforms other model structures investigated. We compared three other structures:

- D-GCN-LSTM: This model has two GCN-LSTM layers [9]: a GCN-LSTM layer is a graph convolutional layer (GCN) followed by an LSTM layer.
- 3-GCN-LSTM: This model has three layers of GCN followed by an LSTM layer.
- CNN-GCN: This model has an one-dimensional convolutional layer (CNN) that works on the feature of each node and a GCN layer, inspired by [22].

One potential reason for CNN to outperform GCN in our case is that our movement floor is rather regular and vertices are equally spaced, and an image representation preserves the grid structure among vertices, which adds restrictions to robot velocity. This helps CNN to extract more information.

Table I also shows how the RMSE increases as the prediction horizon increases. There are several factors contributing to this observation. As the time horizon increases, history features become less relevant and uncertainty of path features grows. Additionally, path features become more sparse as planned paths reach their destinations. As for different categories, the RMSE increases as the ground truth values increase. We investigate the cross-category errors for the prediction models, where vertices in testing samples are categorized based on their ground truth and predicted values, as shown in Figure 5. The prediction models tend to under-predict higher delays, which is partially due to highly unbalanced vertex-wise delay data. In each delay image, less than 3% of the vertices have a non-zero delay value, which is partially due to the lack of observation data at any time point (the ratio between the number of robots and the number of vertices is less than 10%).

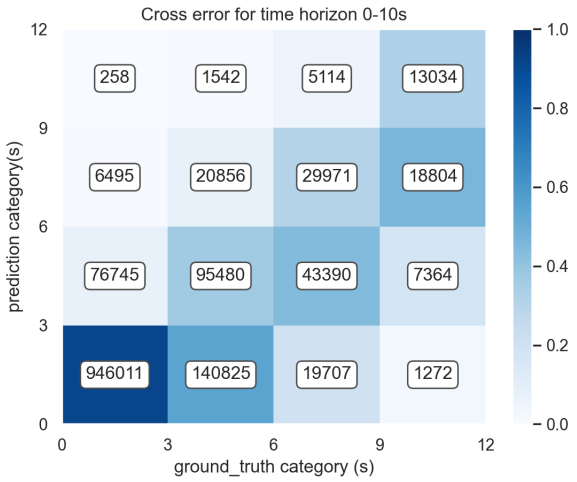

Fig. 5. Cross category prediction heatmaps. The numbers represent the number of samples in each category.

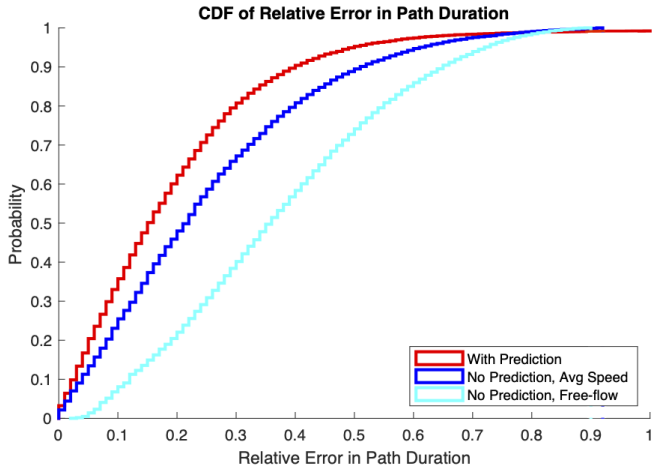

Fig. 6. Relative errors for arrival time estimation.

C. Applications of Prediction Models

For the application of arrival time estimation, we set $\alpha = 0.66$ and examined the error between the predicted and ground truth path durations for 10,000 completed paths from a random test scenario. We compare to two baselines: a single-agent free-flow model that ignores delays, and an empirical "average speed" model, regressed from the simulation data, to capture a general expected motion without predicting spatiotemporal occurrence of delay[4]. Using the predicted delay resulted in RMSE on travel duration of 7.6 s, compared to 14.7 s for the free-flow model (48.6% reduction) and 10.5 s for the average-speed model (27.6% reduction). Figure 6 shows the cumulative distribution functions of the relative error (i.e., path duration error divided by actual duration) for these 10,000 paths, illustrating the consistency of how our prediction reduces the error relative to other options.

For the application of robot fleet path planning, we compute the throughput in average number of package delivery tasks completed per hour (PPH). We compared our approach using delay prediction as time-varying costs to a baseline approach that uses a naive delay prediction: all future delays are predicted the same as the delays from the past 10s. The improvement per simulation scenario, shown in Table II, are aggregated from 10 simulation runs with different random seeds for each scenario. The approach that uses predicted delay as time-varying path planning costs improves system performance by an average of 4.4% and consistently improves throughput across all scenarios.

[4]The "average movement speed" used here is optimistic (learned from the test data) and is only for comparison to spatiotemporal predictions.

TABLE II
SORTATION THROUGHPUT PERFORMANCE

Relative Change	Scenario	Average	Standard deviation
SortationFloor1	1a	4.37%	0.365%
	1b	3.40%	0.291%
	1c	4.61%	0.356%
	1d	1.03%	0.362%
	1e	7.65%	0.460%
	1f	5.87%	0.242%
	1g	4.27%	0.314%
	1h	4.28%	0.487%
	Average	*4.43%*	*0.360%*
SortationFloor2	2a	5.26%	0.616%
	2b	3.31%	0.330%
	2c	2.60%	0.312%
	2d	5.83%	0.967%
	2e	5.50%	0.770%
	2f	3.94%	0.467%
	Average	*4.40%*	*0.577%*

VI. CONCLUSION

We proposed models for predicting congestion-related delays in robot fleets and showed it can improve path planning efficiency and travel time estimation accuracy. Our primary application is the large robot fleet in the warehouse environment, where the scale of mobile robots is beyond the limitations of multi-agent path planning algorithms. We model the delay prediction problem as a multi-step time-series prediction problem by representing both input features and delay ground truth as floor images. Our models are built using the ConvLSTM layers to exploit the spatial and temporal relationships among input features and can predict up to 60 seconds into the future with the resolution of 10-second time windows. Our models do not require human-annotated data for training. Simulation experiments show that when the prediction models are used, throughput in two sortation centers improves more than 4.4%. The predicted delays can also be used to improve estimated travel times for robots by 48.6% compare to the kinematic baseline.

There are several directions for possible future investigations. We represent the floor features as floor images and design a global prediction model; developing prediction models with only local information is left as future work. There is also potential for extensions to other domains such as automobiles, where road networks can be approximated by rigid network structures and GPS data are available.

ACKNOWLEDGMENTS

We would like to thank Kostas Bekris, Pavithra Madhavan, Pablo Fernandez, Lesley Yu, Jeremy Wyatt as well as the Amazon Robotics RAD team for their contributions.

REFERENCES

[1] Peter Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29:9–20, 03 2008.

[2] Qian Wan, Chonglin Gu, Sankui Sun, Mengxia Chen, Hejiao Huang, and Xiaohua Jia. Lifelong multi-agent path finding in a dynamic environment. In *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 875–882, 2018.

[3] Hang Ma, Jiaoyang Li, T. K. Satish Kumar, and Sven Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. *CoRR*, 2017.

[4] Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W Durham, TK Satish Kumar, and Sven Koenig. Lifelong multi-agent path finding in large-scale warehouses. In *AAMAS*, pages 1898–1900, 2020.

[5] Ray J Frank, Neil Davey, and Stephen P Hunt. Time series prediction and neural networks. *Journal of intelligent and robotic systems*, 31(1):91–103, 2001.

[6] Huaxiu Yao, Xianfeng Tang, Hua Wei, Guanjie Zheng, and Zhenhui Li. Revisiting spatial-temporal similarity: A deep learning framework for traffic prediction. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 5668–5675, 2019.

[7] Meng Chen, Xiaohui Yu, and Yang Liu. Pcnn: Deep convolutional networks for short-term traffic congestion prediction. *IEEE Transactions on Intelligent Transportation Systems*, 19(11):3550–3559, 2018.

[8] Navin Ranjan, Sovit Bhandari, Hong Ping Zhao, Hoon Kim, and Pervez Khan. City-wide traffic congestion prediction based on cnn, lstm and transpose cnn. *IEEE Access*, 8:81606–81620, 2020.

[9] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*, 2017.

[10] Shuming Sun, Juan Chen, and Jian Sun. Traffic congestion prediction based on gps trajectory data. *International Journal of Distributed Sensor Networks*, 15(5):1550147719847440, 2019.

[11] Xiaolei Di, Yu Xiao, Chao Zhu, Yang Deng, Qinpei Zhao, and Weixiong Rao. Traffic congestion prediction by spatiotemporal propagation patterns. In *2019 20th IEEE International Conference on Mobile Data Management (MDM)*, pages 298–303. IEEE, 2019.

[12] Takeshi Takahashi, He Sun, Dong Tian, and Yebin Wang. Learning heuristic functions for mobile robot path planning using deep neural networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 764–772, 2019.

[13] Qingbiao Li, Fernando Gama, Alejandro Ribeiro, and Amanda Prorok. Graph neural networks for decentralized multi-robot path planning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11785–11792. IEEE, 2020.

[14] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[15] Hang Ma, Daniel Harabor, Peter J Stuckey, Jiaoyang Li, and Sven Koenig. Searching with consistent prioritization for multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7643–7650, 2019.

[16] David Silver. Cooperative pathfinding. *Aiide*, 1:117–122, 2005.

[17] Michael P Wellman, Matthew Ford, and Kenneth Larson. Path planning under time-dependent uncertainty. *arXiv preprint arXiv:1302.4987*, 2013.

[18] Hang Ma, Wolfgang Hönig, TK Satish Kumar, Nora Ayanian, and Sven Koenig. Lifelong path planning with kinematic constraints for multi-agent pickup and delivery. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7651–7658, 2019.

[19] D. Kornhauser, G. Miller, and P. Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *25th Annual Symposium onFoundations of Computer Science, 1984.*, pages 241–250, 1984.

[20] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai kin Wong, and Wang chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting, 2015.

[21] Francois Chollet et al. Keras, 2015.

[22] Boyan Xu and Hujun Yin. Graph convolutional networks in feature space for image deblurring and super-resolution. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.