Towards Safe Navigation Through Crowded Dynamic Environments

Zhanteng Xie, Pujie Xin, and Philip Dames

Abstract— **This paper proposes a novel neural network-based control policy to enable a mobile robot to navigate safety through environments filled with both static obstacles, such as tables and chairs, and dense crowds of pedestrians. The network architecture uses early fusion to combine a short history of lidar data with kinematic data about nearby pedestrians. This kinematic data is key to enable safe robot navigation in these uncontrolled, human-filled environments. The network is trained in a supervised setting, using expert demonstrations to learn safe navigation behaviors. A series of experiments in detailed simulated environments demonstrate the efficacy of this policy, which is able to achieve a higher success rate than either standard model-based planners or state-of-the-art neural network control policies that use only raw sensor data.**

I. Introduction

Mobile robots are increasingly being used in complex, dynamic, and human-filled environments, such as providing contact-free services during the COVID-19 pandemic, providing customer service in supermarkets, and delivering materials to doctors in hospitals. It is easy to safely avoid static obstacles, such as tables, boxes and chairs, which can be pre-mapped. The main challenge lies in safely navigating around people and other robots, which dynamically move. There are many unknowns in these situations, including the number of pedestrians, the destination for each pedestrian, and the relative speed between the pedestrian and robot, all of which affect the safety [1].

There is extensive prior work on robot navigation, which we divide into two categories: model-based and learning-based. A typical model-based approach is the ROS [2] navigation stack, which uses costmaps to store information about obstacles and the dynamic window approach (DWA) planner [3]. Many other model-based approaches use models of human social interactions to drive robot motion, using social forces [4], [5], Gaussian mixture models (GMM) [6], or a combination of potential functions and limit cycles [7]. All of those model-based approaches require knowledge of the pedestrian kinematics, require multi-stage procedures with preprocessed sensor data, and often require carefully hand-tuning model parameters, making them difficult to implement and generalize to new scenarios.

With the success of deep learning in the computer vision area, many researchers started to apply learning-based approaches to other problems, including robot navigation. Different from the traditional model-based approaches with multi-stage procedures, several recent works use deep neural networks to learn control policies that generate steering commands directly from raw sensor data, *e.g.,* lidar or cameras [8]–[10]. These policies are trained via supervised learning methods, using expert demonstrations to learn safe navigation behavior. However, these end-to-end approaches only focus on static environments.

For navigation in dynamic environments, deep reinforcement learning (DRL) in more commonly used. Fan et al. [11] propose a fully decentralized multi-robot collision avoidance framework using 2-D lidar range data, which is trained using the proximal policy optimization (PPO) algorithm [12]. Sathyamoorthy et al. [13] expand this PPO training policy by using a late fusion network to combine raw lidar data with pedestrian trajectory predictions. Other DRL methods use attention-based networks, which model robot-human interactions with a self-attention model [14], [15] or encode the crowd state with a graph convolutional networks (GCN) [16]. These works all show the importance of leveraging information about pedestrian motion to enable safe navigation in dynamic environments. However, Scholler et al. [17] recently noted that learning-based methods of trajectory prediction often result in behavior that is specific to certain environments, and that the constant velocity kinematic model generalizes much better to new situations. Furthermore, a good reward function is hard to design and DRL networks are also hard to train, though it can be simplified to some degree, for example using AutoRL [18].

In this paper, our primary contribution is the first supervised learning-based navigation policy focusing on crowded dynamic environments. Our solution combines model-based and learning-based approaches, an idea also suggested in a recent survey on deep learning in robotics [19], which allows us to avoid hand-tuning parameters and to incorporate preprocessed sensor data. While other approaches have modeled pedestrian motion, they do so by relying on complex trajectory predictions [13] or abstract human attention information [14]–[16]. Our approach is much simpler and more easily interpretable, requiring only the current kinematics (*i.e.,* position and velocity) of pedestrians. Another key distinction is that we use an early fusion architecture, which greatly simplifies our convolutional neural network (CNN) architecture (compared to late fusion architectures such as [13]) and allows us to fully exploit the input features at an early stage [20]. We demonstrate the efficacy of our approach through a series of simulated experiments, showing that our approach is safer and generalizes to new environments and crowd sizes better than traditional model-based controllers [2] and standard end-to-end approaches [8].

*This work was funded by the Amazon Research Awards Program and NSF grant IIS-1830419.

Zhanteng Xie, Pujie Xin, and Philip Dames are with the Department of Mechanical Engineering, Temple University, Philadelphia, PA, USA {zhanteng.xie, pujie.xin, pdames}@temple.edu

II. Safe Navigation Policy

In this section, we describe the underlying problem of safe navigation through crowded dynamic environments, present our deep learning approach, and describe the data structures used to represent the sensor and pedestrian data. Note: We assume the task-level planning is taken care of by some external agent, our method simply requires the robot to know the relative location of its goal.

A. Problem Formulation

In order to safely navigate to a goal through a dynamic environment with moving pedestrians, the robot must extract useful information from sensors and process this information to get the state of the environment, $\mathbf{s}$. The robot then uses this state to compute the suitable steering velocities via a control policy, which takes the form of a parametric model

$$\mathbf{u} = \pi_\theta(\mathbf{s}), \tag{1}$$

where $\mathbf{u}$ are the steering velocities and θ are our model parameters. The steering velocities consist of $\mathbf{u} = [v_x, w_z]$, where v_x is the translational velocity and w_z is the rotational velocity in the robot's local coordinate frame. Note: Although discretizing the steering velocities would simplify this problem, continuous velocities give a more accurate and safer navigation behavior. The state $\mathbf{s} = [\mathbf{r}, \mathbf{p}, \mathbf{g}]$ has three components in our model: lidar data ($\mathbf{r}$), pedestrian kinematics ($\mathbf{p}$), and the goal position ($\mathbf{g}$). Note, all data in the state $\mathbf{s}$ is expressed in the local robot frame. This allows our method to be robust to errors in robot localization with respect to a global reference frame, which happens more frequently in densely-packed dynamic environment as there are fewer stationary landmarks for the robot to localize itself against. Additionally, relative data is more natural for planning purposes since navigating in a dense crowd is more about going with the flow of traffic than meeting some absolute velocity constraints.

B. Network Architecture

We use a deep neural network, outlined in Fig. 1, to represent the parametric model π_θ due to its amazing function approximation capabilities. One key feature of our network is the use of early fusion to combine three separate feature maps, one representing the lidar data $\mathbf{r}$ and two representing pedestrian kinematics $\mathbf{p}$. Compared with late fusion, early fusion has two obvious advantages [20]. First, early fusion can fully exploit the input information since features are fused at the early stage. Second, it has low memory and computation requirements compared to late fusion, making it more suitable for embedded devices and real-time operation.

We combine the three input feature maps using concatenation fusion, where they are simply stacked along the depth dimension. The concatenated maps are then passed through one 2-D convolutional layer, six bottleneck residual blocks [21] with two block skip connections, and two 2-D pooling layers. Every convolutional layer in our deep neural network is followed with one batch normalization layer and one ReLU action layer. At the end of this early fusion network block,

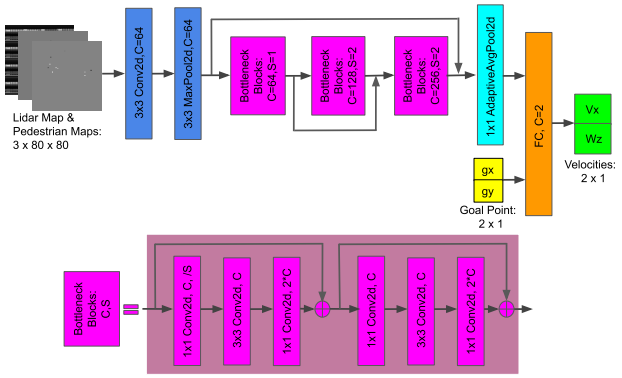


Fig. 1. The architecture of our deep neural network. Our input data consists of three separate channels, one from lidar and two containing pedestrian data, each of which is an 80×80 array. These 3-channel input data are fed to the deep neural network consisting of 1 convolutional layer, 6 bottleneck residual blocks [21], and 2 pooling layers. The output of this chain, along with the relative pose of the goal, are fused in a fully connected layer. The resulting outputs of our network are the linear and angular steering velocities. The values "3×3" or "1×1" in the convolutional layer or pooling layer denote the kernel size. The letters "C" and "S" in the related layers denote the number of output channels and the stride length, respectively.

we can extract the high-level features of lidar and pedestrian information. Then, we fuse those high-level features with the goal information in single a fully connected layer. The goal position is a sub-goal point extracted from a nominal path by the pure pursuit algorithm [22], and is represented by the Cartesian coordinates of the goal in the robot's local coordinate frame, $\mathbf{g} = [g_x, g_y]$. The final steering velocities $\mathbf{u} = [v_x, w_z]$ are generated directly from this fully connect layer without the ReLU action layer.

We use a supervised learning framework to find the model parameters θ from expert demonstrations, $\bar{\mathbf{u}}$. We use mean square error (MSE) between the expert velocity and the learned velocity as our loss function

$$L(\mathbf{s}) = \frac{1}{N}\sum_{i=1}^{N}\left[\pi_\theta(\mathbf{s}_i) - \bar{\mathbf{u}}_i\right]^2, \tag{2}$$

where N is the number of training data tuples $\mathbf{s}_i$ and $\bar{\mathbf{u}}_i$ are the expert examples. We use Adam optimizer [23], a stochastic gradient descent method, to find the optimal model parameters θ^*. We use the step decay with an initial learning rate of 10^{-3} as the learning rate schedule and set the mini-batch size to 128 during the training process.

C. Pedestrian Data Preprocessing

One key contribution of our work is the inclusion of preprocessed data as the input to our CNN control policy. To extract information about pedestrians, we first feed the raw sensor data into an object detector to extract instantaneous estimates of pedestrian locations. These estimates are fed into a multi-target tracker, which performs data association to yield a collection of target tracks containing position and velocity information. Finally, we put this kinematic data into data structures specifically designed for our network architecture. We hypothesize that this detailed information about the motion of individual pedestrians will enable the

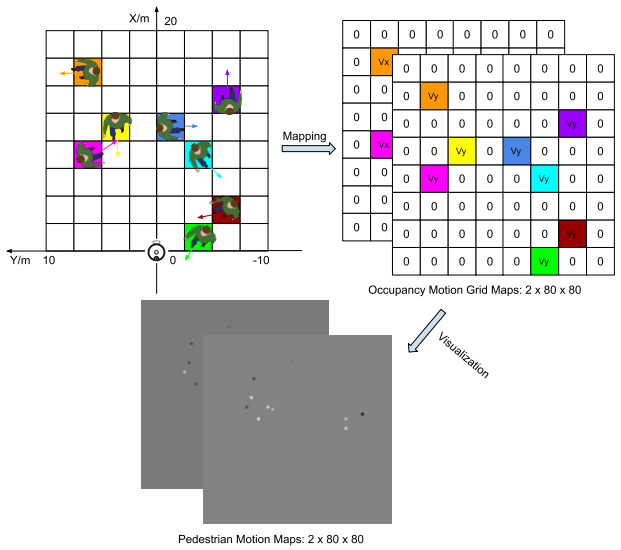


Fig. 2. The preprocessing process for pedestrian data. We get estimates of pedestrian kinematics from the YOLOv3 detector and an MHT tracker. We convert the resulting data into two 80×80 occupancy grid maps, where the pedestrian position determines the cell ID and the velocity gives us the value in that cell. Finally, we standardize the data before feeding them into our network.

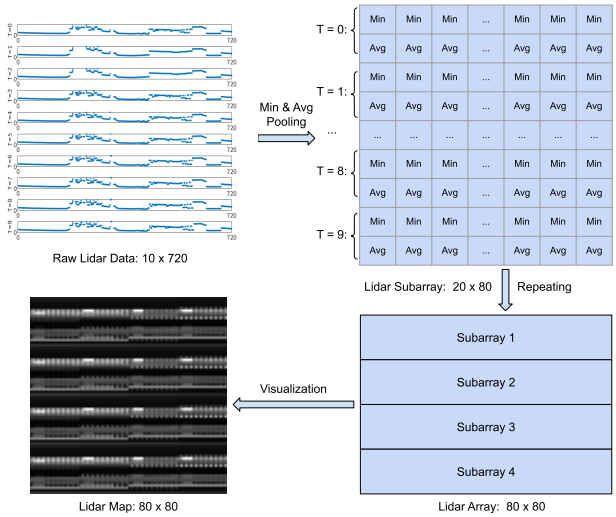


Fig. 3. The preprocessing process for lidar data. The input is raw lidar data containing 10×720 points, since there are 10 scans with 720 measurements per scan. First, we use pooling to downsample each scan to yield a 20×80 array. Second, we repeat this subarray four times to construct an 80×80 lidar array. Finally, we normalize the data in this array before feeding it into our CNN network.

robot to more safely navigate. Note, we using a combination of a stereo camera and 2D lidar.

1) Object Detector: To detect pedestrians, we fuse data from the camera and lidar sensors. The camera data is passed into the YOLOv3 detection algorithm [24] to find labeled bounding boxes for each pedestrian. In parallel, the lidar data is passed into a leg-detection algorithm [25]. We associate the data from these sensors by projecting the laser scan points into the camera image and use this ensemble detector with strong robustness to detect pedestrians and measure their relative positions. Note, we found that the leg detections from the lidar provide a more accurate estimate of the location of each pedestrian while the camera data provides more robust labeling. However, it is possible to use either sensor individually if one desires.

2) Object Tracking: The resulting point estimates of pedestrians are fed into a multiple hypothesis tracker (MHT) [26]. We elect to use the MHT over other multi-target trackers since it was recently shown to achieve state-of-the-art results in the MOT (Multiple Object Tracking) Challenge [27], which involves tracking a large number of people in a crowded scene. One distinction from previous work with the MHT, we perform the tracking in the robot's local coordinate frame instead of a global reference frame. The MHT outputs a list of tracks, each containing the relative position and velocity of a single pedestrian. Note, while the MHT can give us past information about the motion of pedestrians, we only use current information because most of data is redundant for motion prediction and consequently ignored by neural networks [17]. We use the constant velocity motion model for pedestrians, as Scholler et al. [17] recommend, and update the MHT at a rate of 10 Hz. We also tested the inclusion of both past and future (*i.e.,* predicted) pedestrian data, which either decreased or did not improve the performance.

3) Data Structure: Given that the number of pedestrians in a crowd varies greatly based on both time and location, we cannot simply use the direct output of the MHT as our network architecture requires a uniformly-sized input. Instead we encode the pedestrian kinematics into occupancy grid-style maps, as Fig. 2 shows. The basic procedure starts with setting the parameters of the grid. We use an 80×80 grid with 0.25 m cells, for a total area of 20×20 m. We selected this cell size as it is small relative to the size of the targets (*i.e.,* people) being tracked, ensuring that each cell in the resulting grid will only have at most one pedestrian in it. We use two simultaneous grids, each one containing information about target velocity in a different direction, here v_x and v_y separately. Note, we use Cartesian instead of polar coordinates as we found these to yield superior performance (results not included due to page limitations).

These x and y velocity grids form our pedestrian kinematic data $\mathbf{p}$. Before this is fed into our early fusion network, we use the standardization procedure:

$$\mathbf{p} = \frac{\mathbf{p} - \mu_{\mathbf{p}}}{\sigma_{\mathbf{p}}}, \tag{3}$$

where $\mu_{\mathbf{p}}$ and $\sigma_{\mathbf{p}}$ are the mean and standard deviation of the data in the map. Compared with min-max normalization, standardization does not have a bounding range and is more robust to outliers.

D. Lidar Data Preprocessing

Most initial works using CNN-based controller for robot navigation used the entire lidar scan message. However, our early fusion architecture requires that lidar feature map to be the same size as the pedestrian feature maps. Thus, we need to convert the lidar data into an 80×80 feature map.

One interesting argument proposed by Pfieffer et al. [28] is that using the entire lidar scan causes the resulting network to overfit to the specific environment(s) it was trained in, as the CNN learns to extract map features when presented with a dense point cloud. Instead, they use a minimum pooling step to downsample their lidar scan from 1080 data points to 36, with this change leading to better navigation performance in the training environment and better generalizability to other environments. Note: Using minimum pooling is a conservative choice, as this will return the nearest lidar return within each section of the scan.

Motivated by this, we explore the use of different lidar preprocessing operations, including reshaping the raw lidar data, projecting scan points into an occupancy grid map, minimum pooling, average pooling, and more. We found that using a combination of minimum pooling and average pooling to extract two separate distance measurements per scan region, as Fig. 3 shows, yielded the best performance (results not included due to page limitations). We hypothesize that this combination allows the robot to avoid collisions with nearby obstacles (minimum pooling) while also better understanding the geometry of the free space around it (average pooling). In the end, each individual lidar scan results in 80 minimum pooled values and 80 average pooled values. We then use a short history of lidar data, as we found that this contains more useful information than only the current scan, while keeping data from beyond a certain time period resulted in no improvement or even degraded performance. The length of this time window is the effective time constant for pedestrian information, which we found to be 0.5 s, or 10 scans.

To put the lidar data into a format that is compatible with the pedestrian map so that the two data sources can be fused early, we stack the historical data in a 2D array, using alternating rows of minimum pooling and average pooling. The resulting array is 20×80, so we then stack four copies of this to create an 80×80 array of the same size the pedestrian map. Note, we do not require there to be any spatial correlations between lidar data and pedestrian kinematics data. Also, changing the size of the pedestrian map will also require changes to the parameters of lidar processing pipeline as the two data sources must be of compatible sizes. Finally, just like the pedestrian data, we apply the standardization procedure prior to feed the lidar data $\mathbf{r}$ into the early fusion network:

$$\mathbf{r} = \frac{\mathbf{r} - \mu_{\mathbf{r}}}{\sigma_{\mathbf{r}}}, \tag{4}$$

where $\mu_{\mathbf{r}}$ and $\sigma_{\mathbf{r}}$ are the average value and standard deviation of the entire scan data structure.

III. Results

Unfortunately, the COVID-19 pandemic has prevented us from conducting real-world experiments because our method is centered around navigation through dense crowds. As a result, we have used Gazebo [29] to simulate the robot and PEDSIM [30], a microscopic pedestrian crowd simulation library based on social forces, to simulate pedestrian motion. This section describes this setup in greater detail, presents the procedure we used to train our network, and compares the results to other methods.

(a) Lobby world (b) Square world

Fig. 4. Gazebo simulation environments. The lobby world has two configurations with 34 pedestrians and 50 pedestrians respectively, while the square world only has one configuration with 34 pedestrians. The green curve is the robot trajectory and the red point is the goal point.

We use PyTorch to implement our CNN [31]. To train our network, we use the Owl's Nest high-performance computing (HPC) machine learning server, which has 40 CPU cores and 512 GB of RAM, and 8x NVIDIA Tesla Volta V100 GPUs with NVlink2. We run all simulations on a desktop computer with an Intel Core i7-6800K CPU and a GeForce GTX 1080 GPU with 8 GB memory. This computer runs Ubuntu 16.04 and we use ROS Kinetic Kame and Gazebo 9.13.1.

A. Simulation Configuration

PEDSIM uses the social forces model to guide the motion of individual pedestrians. The basic equation used is

$$\mathbf{F_p} = \mathbf{F_p^{des}} + \mathbf{F_p^{obs}} + \mathbf{F_p^{per}}, \tag{5}$$

where $\mathbf{F_p}$ is the resultant force that determines the motion of a pedestrian; $\mathbf{F_p^{des}}$ pulls a pedestrian towards a destination; $\mathbf{F_p^{obs}}$ pushes a pedestrian away from static obstacles; $\mathbf{F_p^{per}}$ models interactions with other pedestrians (*e.g.,* collision avoidance or grouping). We added another repulsive social force $\mathbf{F_p^{rob}}$ between pedestrians and the robot to model the way people would naturally avoid collisions, thereby allowing our control policy to learn this behavior.

Figure 4(a) shows the main environment, a replica of the lobby in the College of Engineering building at Temple University, that we used to train and test our policy. This environment is roughly 25×10 m in size and is filled with a number of static obstacles (*e.g.,* chairs, tables, a security desk, waste bins, and pillars). The second environment, shown in Fig. 4(b), is an artificially created environment that is about 20×20 m in size. Using these two worlds, we set up three scenarios:

- **Lobby world with 34 pedestrians**: used to collect training data and test basic navigation performance.
- **Lobby world with 50 pedestrians**: used to test generalization across different crowd densities.
- **Square world with 34 pedestrians**: used to test generalization to unseen environments.

The robot model is a Kobuki Turtlebot 2, which has a maximum velocity of 0.5 m/s, equipped with a Hokuyo UTM-30LX lidar and a Stereolabs ZED stereo camera. The Hokuyo lidar has a maximum range of 30 m, a FOV of 270°, and an angular resolution of 0.25° while the ZED camera has a minimum range of 0.5 m, a maximum range of 10 m, and a FOV of 90°.

B. Data Collection

Before we can find the optimal θ^* that minimizes (2), we first need to collect a navigation dataset with expert demonstrations $\bar{\mathbf{u}}$ to train our deep neural network. In the future, once crowds can safely gather again, we plan to collect data from actual humans navigating through crowds. In the simulation, we first tried teleoperating the robot. However, we found this to be very time consuming and it did not lead to reliable behavior. On the other hand, we found the DWA planner from the ROS navigation stack can achieve relatively good navigation performance in our simulation environment. So as a trade-off result, like many other similar works [8], [28], we use the DWA planner as our expert.

To collect our training data, the robot was repeatedly assigned to reach a random goal within the free space of the map. PEDSIM controlled the behavior of the 34 pedestrians, which were divided into clusters with a few members each and with each cluster following a series of way points to circulate through the environment. During navigation we recorded the state $\mathbf{s}$ and the expert steering velocities $\bar{\mathbf{u}}$ (*i.e.,* the output of DWA). Every time that the robot collided with an obstacle, we stopped recording data, truncated the data shortly before the collision, and restarted the simulation to avoid learning behavior that would result in unsafe collisions. Although truncating data near a collision may still lead to undesirable behavior, this data is a very small percentage of the total dataset so we dot not expect it to have a significant effect on the resulting policy. The final dataset contains 157,809 $(\mathbf{r}, \mathbf{p}, \mathbf{g}, \bar{\mathbf{u}})$ tuples, which we divided into three separate subsets for training (103,121 tuples), validation during training (22,349 tuples), and final testing (32,339 tuples).

C. Training Results

Using the training dataset from Sec. III-B, we trained three neural networks to demonstrate the efficacy of our novel early fusion neural network with the pedestrian motion information. The first neural network is a state-of-the-art, end-to-end network that uses raw lidar data and the relative goal position as its input [8]. The second one is our architecture using only lidar data and the relative goal, to make a direct comparison with [8]. The last one is our full architecture, including the pedestrian data.

As in [9], we use the following two metrics to evaluate the regression performance of each network:

- **Root mean square error (RMSE)**:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\bar{\mathbf{u}}_i - \mathbf{u}_i)^2}, \tag{6}$$

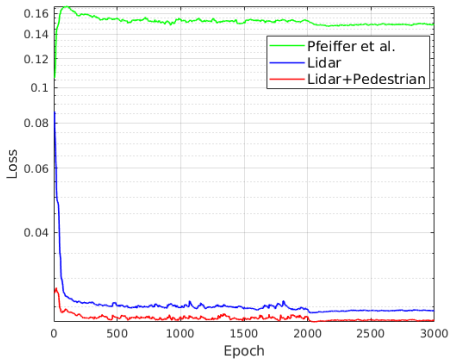


Fig. 5. Validation loss curves during training process, where all three networks converge to stable values.

TABLE I
TRAINING RESULTS

Method	RMSE	EVA	# of Params	FPS
Pfeiffer et al. [8]	0.4265	1.0419	50.732 M	**696.675**
Lidar	0.1794	0.1774	**28.925 M**	266.982
Lidar + Pedestrian	**0.1673**	**0.1543**	28.939 M	255.127

- **Explained variance ratio (EVA)**:

$$EVA = \frac{\sum_{i=1}^{N}(\bar{\mathbf{u}}_i - \mathbf{u}_i)^2}{\sum_{i=1}^{N}(\bar{\mathbf{u}}_i - \mu_{\bar{\mathbf{u}}})^2}. \tag{7}$$

Figure 5 shows the validation loss curves for each network during training and Table I shows the final training results after 3000 epochs. We can observe that although our novel network architecture is nearly two times smaller than Pfeiffer's network [8], it yields significantly better regression performance with smaller RMSE and EVA. Furthermore, our network with pedestrian motion information performs better than only using lidar data, demonstrating that pedestrian motion can help train the deep neural network. Finally, due to the use of early fusion, the addition of pedestrian data has a negligible effect on the number of parameters (only a 0.05% increase). All three methods are clearly capable of real-time processing, with the average frames per second (FPS) of over 250, far above the 40 Hz of the sensors.

D. Navigation Results

While achieving superior loss during training is encouraging, we ultimately care about the resulting behavior of our policy during navigation. We test each of the policies from Sec. III-C, along with the DWA planner [3], in each of the three scenarios from Sec. III-A. We compare the performance using the following metrics, which are commonly used in the autonomous navigation literature [11], [13], [16]:

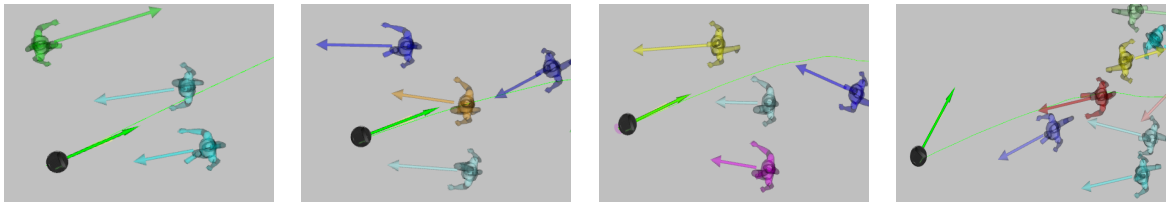
- **Success rate**: the fraction of collision-free trials.
- **Average time**: the average travel time of trials.
- **Average length**: the average trajectory length of trials.
- **Average speed**: the average speed during trials.

In each scenario, we run 4 tests from the same initial conditions, where each test consists of the robot navigating

TABLE II

NAVIGATION RESULTS

Environment	Method	Success Rate	Average Time (s)	Average Length (m)	Average Speed (m/s)
Lobby world with 34 pedestrians	DWA [3]	0.79	**12.468**	**5.164**	**0.414**
	Pfeiffer et al. [8]	0.73	17.549	5.589	0.318
	Lidar	0.78	14.788	5.483	0.371
	Lidar + Pedestrian	**0.84**	15.732	5.598	0.356
Lobby world with 50 pedestrians	DWA [3]	0.67	**13.586**	**5.164**	**0.380**
	Pfeiffer et al. [8]	0.64	18.203	5.858	0.322
	Lidar	0.68	14.635	5.411	0.370
	Lidar + Pedestrian	**0.78**	18.229	5.823	0.319
Square world with 34 pedestrians	DWA [3]	0.72	**21.319**	**8.601**	**0.403**
	Pfeiffer et al. [8]	0.54	57.979	10.826	0.187
	Lidar	0.62	29.950	9.968	0.333
	Lidar + Pedestrian	**0.75**	26.370	9.330	0.354



(a) DWA [3] (b) Peiffer et al. [8] (c) Lidar (d) Lidar + Pedestrian

Fig. 6. Robot reactions to moving pedestrians using each of the control policies. Each figure shows the pedestrians (colored figures), the current velocity of each pedestrian (colored arrows), the robot (black circle), the nominal path to the goal (green line), and the computed velocity (green arrow). When the robot encounters a group of pedestrians, (a) DWA [3], (b) Pfeiffer's policy [8], and (c) our lidar-only CNN are more likely to move forward along the nominal path and ignore those pedestrians until the last moment. (d) On the other hand, our CNN-based policy with pedestrian information reacts earlier.

through the same sequence of 25 goal points. Although the initial conditions of each test are the same, each trial yields different navigation behavior due to sensor noise, social force interactions, etc. Table II summarizes our results. See the accompanying video for a demonstration of the safe navigation behavior of our CNN-based control policy.

We first focus on the original environment used for testing, the lobby world with 34 pedestrians, where we observe several interesting phenomena. First, our novel CNN-based policies with preprocessed data have a higher success rate and a higher average speed than Pfeiffer's policy [8], which uses raw lidar data. This demonstrates that the combination of model-based and learning-based approaches can yield better navigation performance than a purely learning-based approach. Second, despite the fact that our proposed CNN policy with lidar-only input has similar training results and average speed to our policy with pedestrian data, the latter has a significantly higher success rate. This shows that pedestrian information is useful and plays a key role in enabling safe navigation in crowded and dynamic environments.

Our proposed CNN-based policy with pedestrian motion data has a higher success rate than the DWA planner [3], albeit with a lower average speed. We believe there are two main reasons for this. First, because we removed all collisions from our training dataset, our proposed network only learned the *safe* navigation behaviors from the expert (DWA). Second, the pedestrian kinematics help our network understand the motion of pedestrians and learn safe steering behaviors better. Figure 6 demonstrates this, where we can see that our proposed CNN-based policy with pedestrian motion data is more likely to change its heading direction early to avoid pedestrians. As a result of this early avoidance behavior, our CNN-based policy travels a longer distance and takes longer to reach the goal. However, we believe this is worth doing to prevent collisions and that this slowdown could be reduced with better expert data.

The only observable trend in the failure cases of our controller was a slightly increased rate of collisions near the goal location or in particularly dense crowds, though neither occurred regularly. We hypothesize that collisions in dense crowds were more common due to two factors: 1) we used the DWA planner, which fails more often in dense crowds, as our expert, and 2) the social force model breaks down in these situations, with pedestrians often bumping into one another. Future work will aim to mitigate this issue by using an experienced human expert to train our system and real pedestrians to train/test our system (once the COVID-19 pandemic subsides and it is again safe to conduct real-world experiments with dense crowds of people).

Table II also shows that our policy generalizes better to new environments and crowd sizes than the other approaches, particularly the lidar-only CNNs. Notably, the difference in the success rate between our CNN policy and that of Pfeiffer et al. [8] is much larger in the more crowded environment and the unseen environment than that in the training environment. These results demonstrate that our CNN-based policy is able to more easily transfer the learned navigation policies to the more crowded (with almost 50% more people) environments and unseen environments. Despite these improvements, the generalizability of our policy is still limited, with the robot

missing the goal or moving erratically in some unfamiliar scenarios. Future work will mitigate this issue by training our CNN-based policy in different environments and with different crowd sizes.

IV. Conclusion

In this paper we proposed a novel CNN-based control policy to enable a robot to safely navigate through crowded dynamic environments. Our approach differs from prior research in two key ways. First, we use preprocessed data about pedestrian kinematics as opposed to raw sensor data. Second, we use a novel early fusion architecture to fuse a history of lidar data with the pedestrian kinematics. We demonstrate though extensive training and navigation experiments that this pedestrian kinematic data plays a key role in dynamic navigation and that our CNN-based policy has a significantly higher success rate than standard model-based planners or CNN policies with only sensor data. Furthermore, our proposed CNN-based policy generalizes better to different crowd sizes and unseen dynamic environments. Finally, our policy only requires information in the robot's local frame, making it robust to errors in localization that are common in crowded, dynamic environments.

Acknowledgment

This research includes calculations carried out on HPC resources supported in part by the National Science Foundation through major research instrumentation grant number 1625061 and by the US Army Research Laboratory under contract number W911NF-16-2-0189.

References

[1] H.-T. L. Chiang, B. HomChaudhuri, L. Smith, and L. Tapia, "Safety, challenges, and performance of motion planners in dynamic environments," in *Robotics Research*. Springer, 2020, pp. 793–808.

[2] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[3] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[4] D. Helbing and P. Molnar, "Social force model for pedestrian dynamics," *Physical Review E*, vol. 51, no. 5, p. 4282, 1995.

[5] G. Ferrer, A. Garrell, and A. Sanfeliu, "Robot companion: A social-force based approach with human awareness-navigation in crowded environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 1688–1694.

[6] M. Sebastian, S. B. Banisetty, and D. Feil-Seifer, "Socially-aware navigation planner using models of human-human interaction," in *IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2017, pp. 405–410.

[7] M. Boldrer, M. Andreetto, S. Divan, L. Palopoli, and D. Fontanelli, "Socially-aware reactive obstacle avoidance strategy based on limit cycle," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3251–3258, 2020.

[8] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena, "From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots," in *IEEE International Conference on Robotics and Automation*, 2017, pp. 1527–1533.

[9] A. Loquercio, A. I. Maqueda, C. R. Del-Blanco, and D. Scaramuzza, "DroNet: Learning to fly by driving," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1088–1095, 2018.

[10] L. Tai, S. Li, and M. Liu, "A deep-network solution towards model-less obstacle avoidance," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 2759–2764.

[11] T. Fan, P. Long, W. Liu, and J. Pan, "Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios," *The International Journal of Robotics Research*, vol. 39, no. 7, pp. 856–892, 2020.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[13] A. J. Sathyamoorthy, J. Liang, U. Patel, T. Guan, R. Chandra, and D. Manocha, "Densecavoid: Real-time navigation in dense crowds using anticipatory behaviors," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 11 345–11 352.

[14] C. Chen, Y. Liu, S. Kreiss, and A. Alahi, "Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning," in *IEEE International Conference on Robotics and Automation*, 2019, pp. 6015–6022.

[15] L. Liu, D. Dugas, G. Cesari, R. Siegwart, and R. Dubé, "Robot navigation in crowded environments using deep reinforcement learning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 5671–5677.

[16] Y. Chen, C. Liu, B. E. Shi, and M. Liu, "Robot navigation in crowds by graph convolutional networks with attention learned from human gaze," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2754–2761, 2020.

[17] C. Schöller, V. Aravantinos, F. Lay, and A. Knoll, "What the constant velocity model can teach us about pedestrian motion prediction," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1696–1703, 2020.

[18] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with AutoRL," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2007–2014, 2019.

[19] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford *et al.*, "The limits and potentials of deep learning for robotics," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 405–420, 2018.

[20] D. Feng, C. Haase-Schütz, L. Rosenbaum, H. Hertlein, C. Glaeser, F. Timm, W. Wiesbeck, and K. Dietmayer, "Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1341–1360, 2020.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[22] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, Tech. Rep., 1992.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[24] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[25] N. Bellotto and H. Hu, "Multisensor-based human detection and tracking for mobile service robots," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 1, pp. 167–181, 2008.

[26] K. Yoon, Y.-m. Song, and M. Jeon, "Multiple hypothesis tracking algorithm for multi-target multi-camera tracking with disjoint views," *IET Image Processing*, vol. 12, no. 7, pp. 1175–1184, 2018.

[27] C. Kim, F. Li, A. Ciptadi, and J. M. Rehg, "Multiple hypothesis tracking revisited," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 4696–4704.

[28] M. Pfeiffer, S. Shukla, M. Turchetta, C. Cadena, A. Krause, R. Siegwart, and J. Nieto, "Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4423–4430, 2018.

[29] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, 2004, pp. 2149–2154.

[30] C. Gloor, "PEDSIM: Pedestrian crowd simulation," *URL http://pedsim.silmaril.org*, vol. 5, no. 1, 2016.

[31] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019, pp. 8026–8037.