

Dynamically Constrained Motion Planning Networks for Non-Holonomic Robots

Jacob J. Johnson, Linjun Li, Fei Liu, Ahmed H. Qureshi and Michael C. Yip

Abstract—Reliable real-time planning for robots is essential in today’s rapidly expanding automated ecosystem. In such environments, traditional methods that plan by relaxing constraints become unreliable or slow-down for kinematically constrained robots. This paper describes the algorithm Dynamic Motion Planning Networks (Dynamic MPNet), an extension to Motion Planning Networks, for non-holonomic robots that address the challenge of real-time motion planning using a neural planning approach. We propose modifications to the training and planning networks that make it possible for real-time planning while improving the data efficiency of training and trained models’ generalizability. We evaluate our model in simulation for planning tasks for a non-holonomic robot. We also demonstrate experimental results for an indoor navigation task using a Dubins car.

I. INTRODUCTION

The problem of motion planning is among the core robotics challenges, which aims to find a path between given start and goal states while satisfying a set of desired constraints [1]. In most cases, the desired constraints to be met are merely collision-avoidance. However, with the increasing familiarity and adoption of mobile robots and autonomous vehicles, constrained kinematics such as non-holonomic constraints have become prevalent in many situations that require the planner not just to perform collision-avoidance but also adhere to the system’s dynamical equations [2].

Non-holonomic constraints are governed by dynamical equations which depend on the time-derivative of the system’s configuration space [2]. These constraints arise in many applications, ranging from mobile robot navigation [2] to needle steering in robot surgery [3]. To generalize, it comes in cases where the system’s control space is of a lower-dimension than its configuration space. For instance, in car-like robots, the control inputs are the linear and angular velocities, while the configuration/robot-motion space is three-dimensional (x/y position and heading). Consequently, a feasible trajectory in the robot’s configuration space might not be feasible with respect to the system’s dynamics [2], [4].

Various algorithms have been proposed to address real-time motion planning under non-holonomic constraints, on the spectrum of Sampling-based Motion Planning (SMP) to non-Sampling-based methods [5]–[7]. The advantage of sampling-based planners is that they provide probabilistic completeness, i.e., the probability that the planner finds a solution reaches one as the number of samples reaches infinity.

J. J. Johnson, L. Li, F. Liu, A. H. Qureshi and M. C. Yip are with Department of Electrical and Computer Engineering, University of California San Diego, La Jolla, CA 92093 USA. {jjj025, lili, f4liu, alqureshi, yip}@ucsd.edu

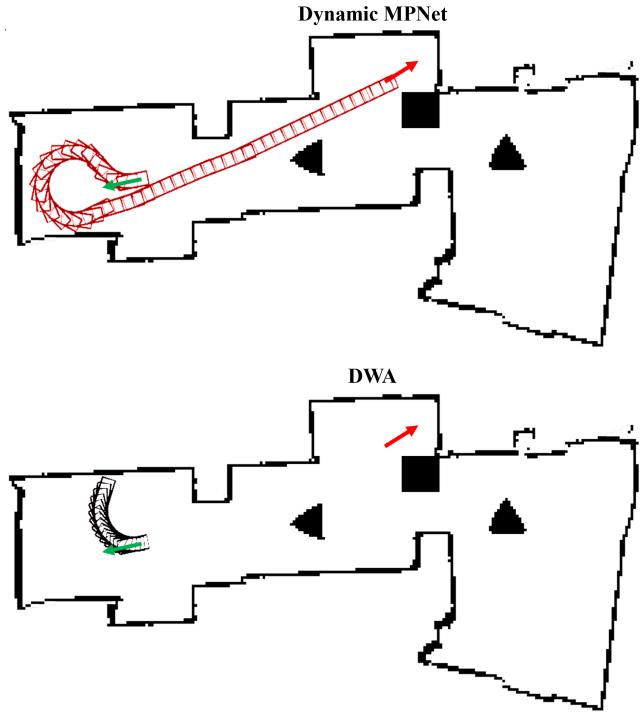


Fig. 1: The trajectory the robot followed for a given start (green arrow) and goal (red arrow) position using, Dynamic MPNet (red), and Dynamic Windows Approach (black) as local planners respectively. Since the DWA planner has no kinematic constraints, it is difficult for the planner to generate paths with U-turns, while since the path generated from Dynamic MPNet encodes kinematic constraints, it is able to generate a successful path towards the goal.

Sampling-based planners were used for real-time planning for non-holonomic systems in the form of anytime planners [8], aiming to improve a current plan while executing an already evaluated plan [8]. Although such planners can easily find a feasible path in uncluttered spaces, they often fail to find a solution in reasonable time in constrained spaces as well as need to relax goal constraints (e.g., final orientation).

Recently reinforcement learning based methods that use neural networks have gained traction in solving motion planning problems [9]–[11] for non-holonomic systems. These methods often require careful fine-tuning of reward functions as well as a significant computing resources to search for proper hyperparameters. Another class of motion planning algorithms, called neural motion planners, have emerged that

learns to imitate an oracle planner and exhibits virtues of an ideal planner during online execution [12]–[14]. Motion Planning Networks (MPNet) [12] is one of the first and prominent neural motion planning methods, showing orders of magnitude performance computational speed compared to previous offline methods while producing near-optimal solutions. However, MPNet, along with its extensions [15], only considers collision-avoidance constraints and finds a viable path in the robot’s configuration spaces, i.e., without taking into account the system’s kinematic limitations. Furthermore, these methods also require significant training data to achieve generalizability and, in their current formulation, have yet to be scaled to real-time planning scenarios involving navigating large environments. Thus none of these methods single-handedly has features of an ideal planner, i.e., find near-optimal/optimal paths with high, almost real-time, computational speed and exhibit completeness guarantees.

In this paper, we propose Dynamic Motion Planning Networks (Dynamic MPNet), which extends MPNet to plan under a broad class of non-holonomic constraints in real-time. Dynamic MPNet is a deep neural-network-based iterative planning algorithm. It takes the sub-goals between given start and goal states from a global C-space planner and finds a kinematically-feasible path between them with a high-computational speed and completeness guarantees. Real-time planning is made possible by planning during executing the previous plan, similar to anytime planning systems. We evaluate our framework on Dubins car dynamical model in challenging navigation tasks where state-of-the-art classical methods are failing, including both simulation and real-robot experiments. Our results show that Dynamic MPNet outperforms existing methods in terms of accuracy and average speed for each trajectory, and, similar to MPNet, also generalizes to new planning problems. We also release Dynamic MPNet as an open-source package in the ROS navigation stack for computationally-efficient local path planning under various kinematic constraints¹.

II. PROBLEM DEFINITION

The motion planning algorithm with dynamic constraints can be described as follows. Consider a dynamic system defined by the differential equation $\dot{s}(t) = f(s(t), u(t))$, where $s(t) \in \mathcal{S} \subset \mathbb{R}^n$ describes the state of the system, $u(t) \in \mathcal{U} \subset \mathbb{R}^m$ describes the control inputs to the system and $\dot{s}(t) \in \mathbb{R}^n$ describes the rate of change of the system state. The state space \mathcal{S} can be split into two regions: \mathcal{S}_{obs} states with obstacles and $\mathcal{S}_{free} = \mathcal{S} - \mathcal{S}_{obs}$ obstacle free regions. The goal of the planning algorithm is to find a set of control inputs $u(t)$ that would move the robot from a given initial state $s_{start} \in \mathcal{S}_{free}$ to a given goal state $s_{goal} \in \mathcal{S}_{free}$, such that the sequence of actions $u(t)$ and $s(t) \forall t \in [0, \tau]$ satisfy the given dynamic equation and the states are in free space, i.e. $s(t) \in \mathcal{S}_{free}$. The problem can be expanded further by adding optimality criteria to the sequence of states. In the subsequent sections, we describe our solution to the

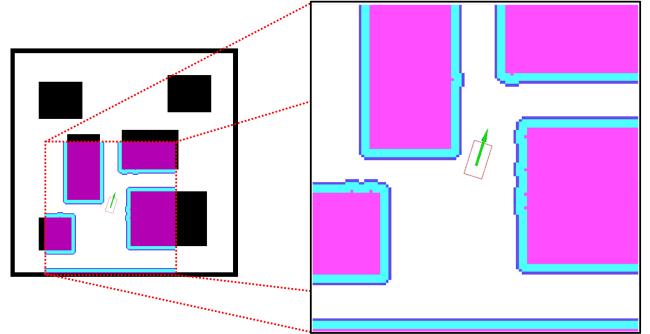


Fig. 2: The area of the costmap passed to MPNet for planning. The local costmap used for planning is always ego-centric to the robot. The blue region indicates the obstacles inflated by the robot footprint.

motion planning problem. The planner is used as a local planner in a hierarchical navigation architecture.

III. METHODS

Dynamic MPNet uses supervised learning to train neural networks that generate near-optimal paths through an environment given a start and goal position. The networks are trained with expert trajectories in randomized, diverse environments so that unseen environments can be planned in with near-expert level cost and with substantially less sampling than classical sample-based planners.

Dynamic MPNet consists of 3 core modules, a *transformer* that centers the local costmap with respect to the robot, an *encoder* network that takes the ego-centered obstacle map and converts it into a latent vector, and a *planner* network that takes the latent encoding, the current or predicted pose, and goal pose and returns the next feasible step. In the following section, we will go over the environment encoding, network training and planning pipeline in more detail.

A. Environment Encoding

Most non-holonomic systems reside in environments that can vary significantly in size. Encoding an environment where the map could be arbitrarily large in size is infeasible. This is where a hierarchical approach to the navigation is useful. Using the global plan as a guide, kinematically feasible paths are generated for a local region using Dynamic MPNet. Fig. 2 is an example for the local costmap passed into the network planner. This costmap is always egocentric to the robot (see Fig. 2). Doing this reduces the dimensionality of the input space, as the output of the network is only a function of the current robot orientation, relative goal position, and costmap. It could also be viewed as a normalization step, to disentangle the dependence of training trajectory and world map with the sampled point. It thus helps the model to generalize to the environment better with fewer training samples.

B. Training

The planner is trained with dynamically feasible trajectories such that, during prediction, it will tend to predict

¹https://github.com/jacobjj/mpnet_local_planner

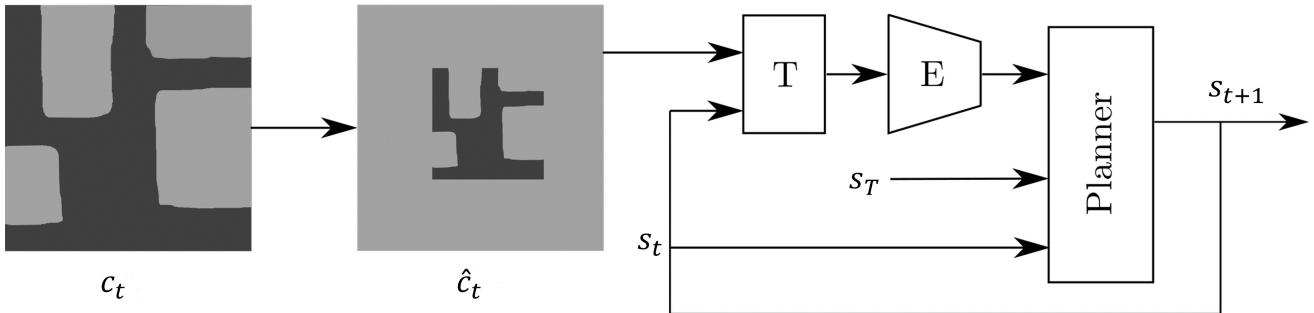


Fig. 3: The graph describes the flow of inputs and outputs for the planner. x_t , x_{t+1} and x_g represents the current, next and target positions respectively. x_g is the sub-goal position from the global plan. C_t and, \hat{C}_t is the costmap before and after padding respectively. The T block centers the padded costmap, \hat{C}_t , with respect to the robot position x_t . E block consists of convolution networks that encode the costmap into latent space vectors. The latent space representation of the costmap, the current robot and goal position are passed to the Planner node to generate the new target point.

dynamically feasible intermediate poses. Given an expert trajectory $\{s_0, s_1, \dots, s_T\}$ that satisfies the dynamical constraints of the robot for the planning problem, the network uses the current position (s_t), goal position (s_T) and obstacle representation (\hat{c}_t) to predict the next state (\hat{s}_{t+1}). These four elements form a training tuple $(s_t, s_T, \hat{c}_t, s_{t+1})$. The encoding and planning networks are trained by reducing the mean-squared error between the predicted state \hat{s}_{t+1} and the actual state from the expert planner s_{t+1} in an end-to-end fashion using gradient descent. The loss function for N such trajectories is given by:

$$L(\theta) = \frac{1}{N T} \sum_{j=1}^N \sum_{t=1}^{T-1} \|s_{j,t} - \hat{s}_{j,t}\|^2 \quad (1)$$

where θ represents the combined parameters of both the encoder and planner network.

C. Planning

The network starts planning using the current position, sub-goal position from the global plan, and the local costmap to generate a sequence of kinematically feasible states. For each step, if the predicted state is kinematically feasible and is collision-free, it is used as the current position for the next prediction. Fig. 3 shows the complete pipeline. Then, for each sampled step during the planning, an egocentric costmap encoding at this immediate (non-initial) location is required to generate the next step. This costmap is achieved by translating the initial costmap. It is necessary to pad the obstacle map such that no loss in obstacle information occurs after transforming the map to an egocentric pose of an intermediate step of a motion plan. Given a grid of size $l \times l$, we pad it to make a new grid of size $2l \times 2l$. This way, for the planner, the world's perception remains the same, since all the padded spaces are still obstacles.

Algorithm 1 outlines the Dynamic MPNet planner, and Algorithm 2 outlines the modified path generation heuristic. The functionality of the different function calls are defined as follows:

1) Padding: The `Pad` function takes a costmap $c_{obs} \in \mathbb{R}^{l \times l}$, and returns a padded costmap $\hat{c}_{temp} \in \mathbb{R}^{2l \times 2l}$. Padded values are assumed to be obstacle regions to prevent the planner to find paths that would navigate this non-physical space. See Fig. 3 for an example.

2) Steering: The function `Steer` (x_1, x_2) checks if we can generate a sequence of kinematically feasible states without collision from x_1 to x_2 within a fixed time. It returns a feasible path if it exists otherwise, an empty list. For non-holonomic systems, a differential equation solver or parameterized curves along with a collision checker can be used to implement this function.

3) Network: The function `Net` represents both the encoder and planner neural network combined. It generates the next possible point on the path, given the current and goal position of the robot and the modified costmap. The flow of inputs is described in Fig. 3.

4) Add: The function `Add` (τ, x_1) appends the path τ with the node x_1 .

5) Transform: Given a point x_i and a padded costmap \hat{c} , the function `Transform` (\hat{c}, x_i) translates the costmap in such a way that the costmap is egocentric to the position x_i .

6) Replanning: Only if, for a given start and goal location, the neural planner is not able to provide a feasible path under a suitable amount of time, the function will run classical sampling-based methods until a path is found, specifically to ensure probabilistic completeness.

Algorithm 1: $\tau \leftarrow \text{DynamicMPNet}(x_{start}, x_{goal}, c_{obs})$

```

1  $\hat{c} \leftarrow \text{Pad}(c_{obs})$ ;
2  $\tau \leftarrow \text{NeuralPlanner}(x_{start}, x_{goal}, \hat{c})$ ;
3 if Empty ( $\tau$ ) then
4   |  $\tau \leftarrow \text{Replanner}(x_{start}, x_{goal})$ ;
5 end
6 return  $\tau$ 

```

Algorithm 2: $\tau \leftarrow \text{NeuralPlanner } (x_{from}, x_{to}, \hat{c})$

```

1  $\tau \leftarrow \{x_{from}\};$ 
2 for  $i = 0$  to  $N$  do
3    $x_{temp} \leftarrow \text{Net}(x_{from}, x_{goal}, \hat{c});$ 
4    $\tau_{temp} \leftarrow \text{Steer}(x_{from}, x_{temp});$ 
5   if  $\text{NotEmpty}(\tau_{temp})$  then
6      $\tau \leftarrow \text{Add}(\tau, \tau_{temp});$ 
7      $\tau_{goal} \leftarrow \text{Steer}(x_{temp}, x_{goal});$ 
8     if  $\text{NotEmpty}(\tau_{goal})$  then
9        $\tau \leftarrow \text{Add}(\tau, \tau_{goal});$ 
10      return  $\tau$ 
11    end
12     $x_{from} \leftarrow x_{temp};$ 
13     $\hat{c} \leftarrow \text{Transform}(\hat{c}, x_{from});$ 
14  end
15 end
16 return  $\emptyset$ 

```

IV. EXPERIMENTS

In this section, we describe the data collection, model architecture and experiment setup used in this paper. The neural networks model was defined and trained using the PyTorch [16] python library, and the trained model was loaded into C++ using the torch C++ API. To test the viability of our framework, we integrated the Dynamic MPNet planner to the navigation stack [17] provided by the ROS community and compared it with standard local planners used by the community. We used the default global planner from the ROS navigation stack and the mit-racecar [18] model was used as the robot for the simulations.²

A. Data Collection

For training the model, we collected expert trajectories using the RRT* [1] algorithm using the Open Source Motion Planning Library [19]. For the grid map (Fig. 4) environment, we trained the model on 10,000 RRT* trajectories by randomly sampling a start and goal location for a fixed time, while for the real-world environment, we generated 12,000 trajectories. To further augment our data set, we chose smaller sections of a path and added it to the training data. Thus if we have n points on a path, then we can choose any 2 points and create $\mathcal{O}(\frac{n^2}{4})$ trajectories. The data collection was accelerated by encapsulating the sampling code in a docker container and launching multiple containers concurrently with different seed values.

B. Model Architecture

A convolutional neural network model was used for environment encoding. Prior works in robotics have also used CNN's to process the cost map for planning [20] and path prediction [21]. The input to the encoder was an $l \times l$ dimension cost map. The CNN consisted of 3 convolutional layers, with kernel size [5, 5], [3, 3], and [3, 3],

²<https://youtu.be/1b3i1SSiUms>

and output channels of 8, 16 and 32, respectively. A maxpool and Parametric Rectified Linear Unit [22] (PReLU) layer follow the first two convolutional layers. The output of the final convolutional layer is passed through a PReLU layer to generate the output of the encoder.

The planner was a fully connected neural network with six layers. A PReLU [22] and a Dropout [23] layer follow the first four hidden layers. Dropout is used not only during training to prevent overfitting [23] but also during prediction to introduce stochasticity that tends to make motion planning networks more robust [12]. A PReLU follows the penultimate hidden layer and the output of the final hidden layer is passed through a tanh nonlinearity. Both networks were trained in an end-to-end fashion.

C. Kinematic Model of a Car-Like Robot and Dubins curve

In this paper we consider non-holonomic kinematics following the Dubins vehicle model [24] though in practice the constraint can be of another variety. The Dubins model is given by:

$$\dot{\mathbf{s}}(\mathbf{t}) = \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} v_s \cos(\phi) \\ v_s \sin(\phi) \\ \frac{v_s}{d} \tan \phi \end{bmatrix} \quad (2)$$

where v_s is the speed of the car, ϕ is the steering angle and d the distance between the rear and front axle. We use Dubins curves as a steering function for Dynamic MPNet. Given the state variables for a Dubins vehicle, the shortest path is a unique path among 6-basis trajectories [24]. Each of these trajectories is represented by a sequence of left, right and straight turns. To steer between given two states, the shortest among the 6-trajectories are used.

D. Trajectory Tracking

Given a sparsely sampled path produced by Dynamic MPNet, a trajectory tracking problem is then presented during runtime to move the vehicle between the sampled points. This is necessary to ensure that due to unmodelled effects and under noise and disturbances that the vehicle follows, to the best of its capability, the solution to a dynamically feasible path provided by Dynamic MPNet.

We formulate the trajectory tracking problem as a constrained discrete optimization problem with a finite horizon. In order to follow the trajectory generated from MPNet, we sample the path nodes adaptively to proper sparsity with similar distances and apply a Nonlinear Model Predictive Control (NMPC, [25]) with the following objective function:

$$\begin{aligned} & \text{minimize}_{\mathbf{t}=0}^N w_s \|\mathbf{s}(\mathbf{t}) - \hat{\mathbf{s}}(\mathbf{t})\| + w_u \|\phi(t)\| + w_a \|\Delta\phi(t)\| \\ & \text{subject to } \Delta\mathbf{s}(\mathbf{t}) = \begin{bmatrix} v_s \cos(\phi) \\ v_s \sin(\phi) \\ \frac{v_s}{d} \tan \phi \end{bmatrix} \Delta t \end{aligned} \quad (3)$$

where Δt and Δs are time step and state difference respectively, w_s is state loss weight, w_u , w_a are weights for control loss, $\hat{\mathbf{s}}$ are sampled path nodes from Dynamic MPNet, v_s is preset as constant velocity and N is the prediction horizon. In

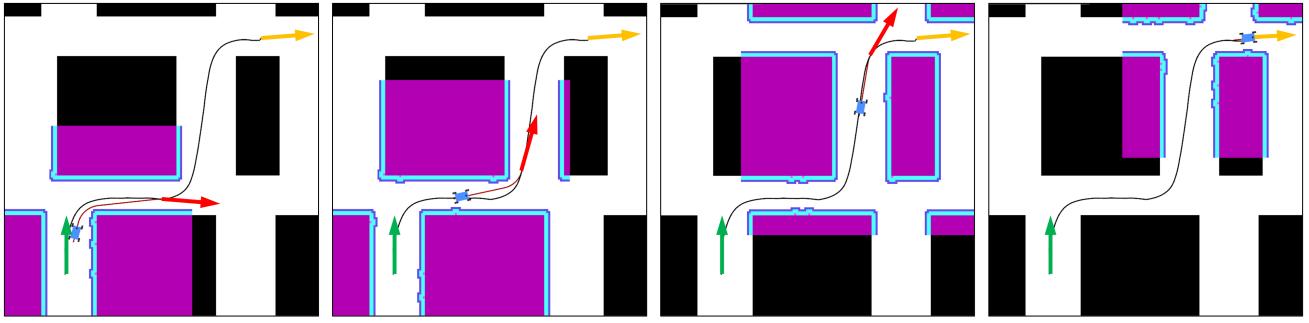


Fig. 4: For a given start (green arrow) and goal (orange arrow) position, the plan generated by the Dynamic MPNet (red path) for a given sub-goal (red arrow). The black trajectory is the global plan. The colored region represents the local costmap used by the Dynamic MPNet.

each control cycle, the control output is computed by solving the problem with Interior Point Optimizer (Ipopt, [26], [27]) implemented with algorithmic differentiation library CppAD [28] and the first action is taken from solution.

The prediction horizon was determined using path curvature and maximum velocity and was in sync with the control frequency to reach the targeted state. The w_a term contributes to decreasing the vibration due to maneuvering, which guarantees the smoothness of the trajectory.

V. RESULTS

We evaluate the learned model in simulation on seen and unseen environments, and a real-world indoor environment using a Dubins car robot. The Dubins car is set up similar to the MIT Racecar [18]. The local planners available for car-like robots in the ROS-navigation stack is the Dynamic-Window Approach (DWA) planner [29] and Timed-Elastic-Band (TEB) [30]. Although the TEB is the only ROS local planner for car-like robots, the optimization problem was not able to generate paths for Dubins car. As a result we compared our algorithm with DWA. We also implemented an Anytime RRT* local planner by generating the path from RRT* rather than from the Neural Planner. We compare the robot’s accuracy and average speed over a batch of planning problems. Each planning problem was verified to have a solution using an offline RRT* planner. For each test case, the problem is considered solved if the robot center is able to achieve the target position within a radius of 0.2m (about half the length of the robot) and target orientation within 15°. The same threshold is used for all local planners. Each planner runs at 5Hz, giving it 200 ms to find and optimize a feasible plan. In the following sections we report our results from our experiments.

To evaluate the sampling speed of Dynamic MPNet on the trained simulated environment in terms of execution time and path length, we measured the time taken to sample n points using Dynamic MPNet is given in Table I. These times indicate that the Dynamic MPNet planner is able to generate a path within 20Hz if set to high resolution of 50 points per local path generated. Thus Dynamic MPNet is able to generate kinematically reachable points in real-time.

TABLE I: Planning Time of Dynamic MPNet vs. Sampling Resolution

Number of samples	5	10	25	50
Compute Time (ms)	11.33	15.29	22.07	44.67

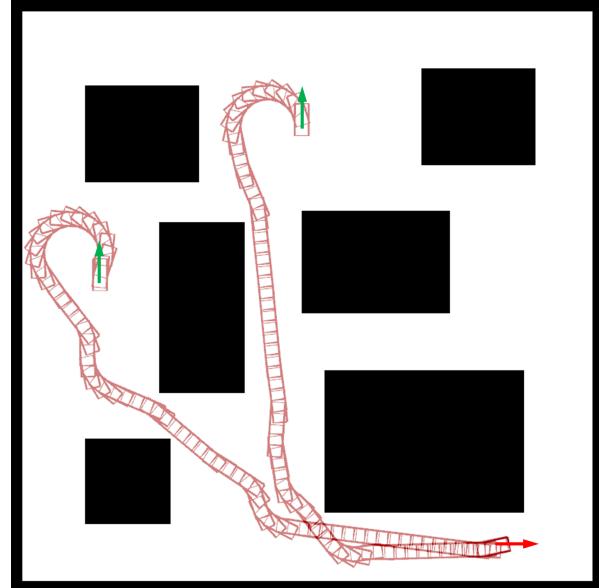


Fig. 5: The Dynamic MPNet generating trajectories for an untrained map for two different start and end goal on a synthetic map. The planner can generate trajectories that comply with the kinematic constraints of the robot, thus achieving higher accuracy compared to DWA.

The Dynamic MPNet was trained on a synthetic grid world environment. One of the paths generated by the trained planner is shown in Fig. 4. To evaluate the generalizability of the planner, we created a synthetic map that is different from the training map, but shares a lot of common obstacle features such as 90° turns from the original map. Paths generated on this environment is shown in Fig. 5. Table II compares the percentage of planning problems solved with standard planners. Dynamic MPNet is able to plan 34% more planning problems compared to DWA on the unseen map. Hence we were able to achieve generalizability with much fewer training paths.

TABLE II: Percentage of Planning Problems Successfully Completed

Environment	DWA	Anytime RRT*	Dynamic MPNet
Unseen map	47%	74%	80%
Real world map	44%	58%	76%



Fig. 6: The RC robot used for this work. On board LIDAR (Hokuyo UTM-30LX-EW) and IMU (RealSense D435i) sensors were used for localization.

In addition to the simulation experiments, real-world experiments with an RC Dubins Car (see Fig. 6) in one of our mapped office buildings were used. Fig. 8 compares one of the trajectories planned by DWA and Dynamic MPNet for the same start and goal point. The red path given by Dynamic MPNet is 9.43m long while the DWA path is 10.85m long. Since Dynamic MPNet is trained on RRT* paths, the local paths generated would be near optimal. In Fig. 7 we compare the distance of each trajectory and time take to complete the planning problems solved by both DWA and Dynamic MPNet for the unknown and real world maps. A linear regression model was fit to both the models to estimate the average speed of the robot. Table III summarizes the results. Dynamic MPNet is able to solve planning problems faster compared to DWA.

In Fig. 1 we can observe one of the biggest drawbacks of the DWA planner which is that it does not take the kinematics constraint of the robot into consideration for generating a plan. Since the Dynamic MPNet generates kinodynamically feasible paths, it is able to plan for the given goal position and orientation. As a result, MPNet is able to solve a larger percentage of planning problems compared to standard planners for both simulated and real world maps.

VI. CONCLUSIONS

In this work, we have used neural motion planners to generate paths for non-holonomic robots successfully. We

TABLE III: Average Vehicle Speed of Dynamic MPnet v.s.
DWA

Environment	DWA Speed (m/s)	Dynamic MPNet Speed (m/s)
Unseen map	0.211 ± 0.016	0.340 ± 0.006
Real world map	0.263 ± 0.014	0.336 ± 0.003

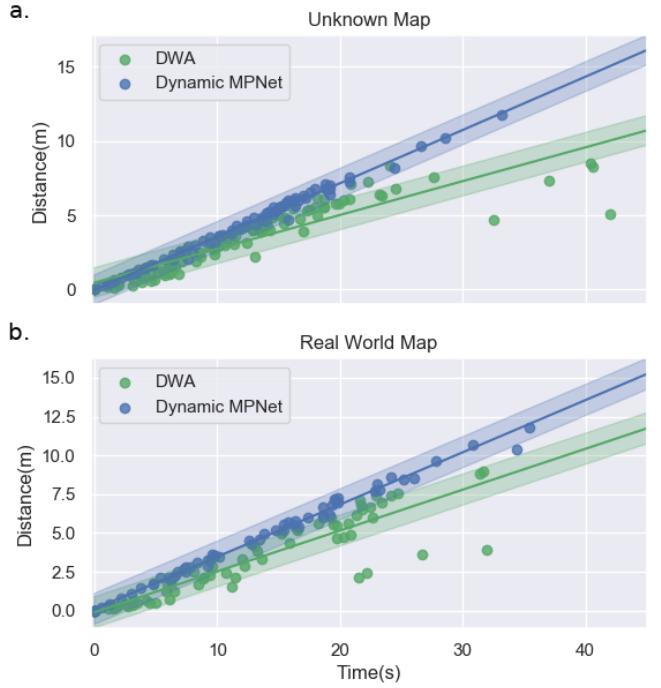


Fig. 7: Total time and total distance taken by Dynamic MPNet and DWA planners on the same set of planning problems for an (a) unknown map (b) real world map. A linear regression model was fit to each dataset to evaluate the average speed of the car for the local planners. The shaded region represents standard error in the estimates. Dynamic MPnet not only moves faster and more consistently than DWA, it solves problems that require longer planning distances as well.

achieved this by introducing a new framework to facilitate real-time planning for non-holonomic robots. Compared to traditional sampling-based methods, Dynamic MPNet can achieve faster average speeds with higher accuracy. In our future studies, one of our primary goals is to extend Dynamic MPNet to problems with moving obstacles by leveraging its remarkable properties of finding near-optimal paths in almost real-time computation speed. One of the short coming of real time planning method is it's dependence on the global plan to generate a feasible path to the goal because of which real time navigation methods fail on a large number of planning problems for non-holonomic systems. A future goal would be to integrate the constraint planner to generate a kinematically feasible global plan. Another future objective is to utilize Dynamic MPNet for needle steering in surgical tasks that, in most cases, require a planner to satisfy underlying non-holonomic constraints.

REFERENCES

- [1] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” 1998.
- [2] J.-P. Laumond, S. Sekhavat, and F. Lamirault, “Guidelines in nonholonomic motion planning for mobile robots,” in *Robot motion planning and control*. Springer, 1998, pp. 1–53.

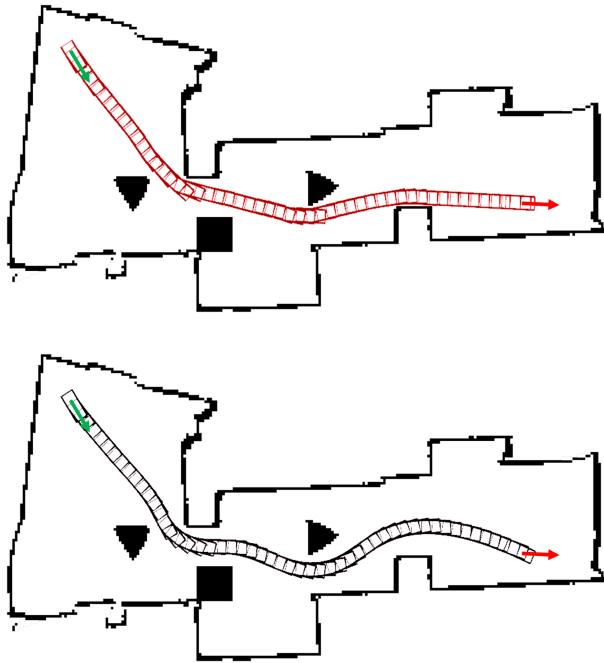


Fig. 8: The trajectory the robot followed for a given start (green arrow) and goal (red arrow) position for DWA (black) and Dynamic MPNet (red). The path Dynamic MPNet takes is much closer to the obstacle compared to DWA and hence shorter.

- [3] R. Alterovitz, T. Siméon, and K. Goldberg, “The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty,” 2007.
- [4] A. De Luca, G. Oriolo, and C. Samson, “Feedback control of a nonholonomic car-like robot,” in *Robot motion planning and control*. Springer, 1998, pp. 171–253.
- [5] W. Xu, J. Wei, J. M. Dolan, H. Zhao, and H. Zha, “A real-time motion planner with trajectory optimization for autonomous vehicles,” in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 2061–2067.
- [6] C. Rösmann, F. Hoffmann, and T. Bertram, “Kinodynamic trajectory optimization and control for car-like robots,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 5681–5686.
- [7] S. Karaman and E. Frazzoli, “Sampling-based optimal motion planning for non-holonomic dynamical systems,” in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 5041–5047.
- [8] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, “Anytime motion planning using the rrt,” in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 1478–1483.
- [9] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, “Deep reinforcement learning with successor features for navigation across similar environments,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 2371–2378.
- [10] A. Faust, K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, and J. Davidson, “Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 5113–5120.
- [11] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis, “Learning navigation behaviors end to end,” *CoRR*, abs/1809.10124, 2018.
- [12] A. H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip, “Motion planning networks,” in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 2118–2124.
- [13] A. H. Qureshi, Y. Miao, A. Simeonov, and M. C. Yip, “Motion planning networks: Bridging the gap between learning-based and classical motion planners,” *IEEE Transactions on Robotics*, 2020.
- [14] A. H. Qureshi and M. C. Yip, “Deeply informed neural sampling for robot motion planning,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 6582–6588.
- [15] T. Jurgenson and A. Tamar, “Harnessing reinforcement learning for neural motion planning,” *arXiv preprint arXiv:1906.00214*, 2019.
- [16] 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. 8024–8035.
- [17] “Ros navigation-stack.” [Online]. Available: <https://github.com/ros-planning/navigation>
- [18] “mit-racecar simulator.” [Online]. Available: https://github.com/mit-racecar/racecar_simulator
- [19] I. A. Şucan, M. Moll, and L. E. Kavraki, “The Open Motion Planning Library,” *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, <http://ompl.kavrakilab.org>.
- [20] H. Banzhaf, P. Sanzenbacher, U. Baumann, and J. M. Zöllner, “Learning to predict ego-vehicle poses for sampling-based nonholonomic motion planning,” *CoRR*, vol. abs/1812.01127, 2018. [Online]. Available: <http://arxiv.org/abs/1812.01127>
- [21] U. Baumann, C. Guiser, S. Herman, and J. M. Zöllner, “Predicting ego-vehicle paths from environmental observations with a deep neural network,” *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–9, 2018.
- [22] L. Trottier, P. Giguère, and B. Chaib-draa, “Parametric exponential linear unit for deep convolutional neural networks,” *CoRR*, vol. abs/1605.09332, 2016. [Online]. Available: <http://arxiv.org/abs/1605.09332>
- [23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *J. Mach. Learn. Res.*, vol. 15, no. 1, p. 19291958, Jan. 2014.
- [24] L. E. Dubins, “On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents,” 1957.
- [25] L. Grüne and J. Pannek, *Nonlinear Model Predictive Control*. London: Springer London, 2011, pp. 43–66. [Online]. Available: https://doi.org/10.1007/978-0-85729-501-9_3
- [26] J. T. Betts, S. K. Eldersveld, P. D. Frank, and J. G. Lewis, “An interior-point algorithm for large scale optimization,” in *Large-Scale PDE-Constrained Optimization*, L. T. Biegler, M. Heinkenschloss, O. Ghattas, and B. van Bloemen Waanders, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 184–198.
- [27] A. Wchter and L. T. Biegler, “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming,” *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, Mar. 2006. [Online]. Available: <https://link.springer.com/article/10.1007/s10107-004-0559-y>
- [28] B. M. Bell and J. V. Burke, “Algorithmic differentiation of implicit functions and optimal values,” in *Advances in Automatic Differentiation*, C. H. Bischof, H. M. Bücker, P. Hovland, U. Naumann, and J. Utke, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 67–77.
- [29] 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.
- [30] C. Rösmann, F. Hoffmann, and T. Bertram, “Kinodynamic trajectory optimization and control for car-like robots,” 10 2017.
- [31] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
- [32] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: <http://arxiv.org/abs/1512.03385>
- [33] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, pp. 529–533, 2015.
- [34] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *CoRR*, vol. abs/1312.6114, 2013.