DynaBARN: Benchmarking Metric Ground Navigation in Dynamic Environments

Anirudh Nair\textsuperscript{1}, Fulin Jiang\textsuperscript{1}, Kang Hou\textsuperscript{1}, Zifan Xu\textsuperscript{1}, Shuozhe Li\textsuperscript{1}, Xuesu Xiao\textsuperscript{1}, and Peter Stone\textsuperscript{1,2}

Abstract—Safely avoiding dynamic obstacles while moving toward a goal is a fundamental capability of autonomous mobile robots. Current benchmarks for dynamic obstacle avoidance do not provide a way to alter how obstacles move and instead use only a single method to uniquely determine the movement of obstacles, e.g., constant velocity, the social force model, or Optimal Reciprocal Collision Avoidance (ORCA). Using a single method in this way restricts the variety of scenarios in which the robot navigation system is trained and/or evaluated, thus limiting its robustness to dynamic obstacles of different speeds, trajectory smoothness, acceleration/deceleration, etc., which we call motion profiles. In this paper, we present a simulation testbed, DynaBARN, to evaluate a robot navigation system’s ability to navigate in environments with obstacles with different motion profiles, which are systematically generated by a set of difficulty metrics. Additionally, we provide a demonstration collection pipeline that records robot navigation trials controlled by human users to compare with autonomous navigation performance and to develop navigation systems using learning from demonstration. Finally, we provide results of four classical and learning-based navigation systems in DynaBARN, which can serve as baselines for future studies. We release DynaBARN open source as a standardized benchmark for future autonomous navigation research in environments with different dynamic obstacles. The code and environments are released at \url{https://github.com/aninair1905/DynaBARN}.

I. INTRODUCTION

With autonomous mobile robots being deployed in the real world, such as for package delivery on crowded sidewalks, autonomous driving in dense traffic, and intelligent warehouses with hundreds of moving objects, safely avoiding dynamic obstacles in the environment while efficiently reaching their goal is now a fundamental robot capability. Navigation among dynamic obstacles has been studied for decades, especially in the domain of multi-agent path planning [1], [2] and autonomous driving [3], [4]. More recently, machine learning techniques for dynamic obstacle avoidance have been applied to multi-agent systems [5], [6] and social navigation [7]–[9].

Despite the abundance of research in navigation with dynamic obstacle avoidance, there is no generally accepted metric by which to compare methods against each other. While difficulty in navigating in a static environment is relatively easy to quantify [10], there are more factors to consider with dynamic obstacles such as obstacles’ speed, direction, acceleration/deceleration, smoothness, etc. However, existing simulation environments to develop and test dynamic obstacle avoidance methods typically only contain obstacles that move using a single method, such as constant velocity, the social force model [11], or Optimal Reciprocal Collision Avoidance (ORCA) [12]. As a result, the obstacles in such simulation environments have similar motion profiles (i.e., speed, direction, acceleration/deceleration, smoothness, etc.) and only the navigation system’s ability to avoid such obstacles is tested, ignoring its robustness to obstacles with other motion profiles. Considering that dynamic obstacles in the real world do not necessarily always follow one single motion profile (e.g., a pedestrian and a scooter may move differently on a sidewalk around a delivery robot), a good dynamic obstacle avoidance method should be able to avoid obstacles with different motion profiles.

To address the lack of a standardized method to compare and test mobile robot navigation in dynamic environments containing obstacles with varying motion profiles, we introduce DynaBARN. DynaBARN is an extension to the Benchmark Autonomous Robot Navigation (BARN) dataset of 300 simulated, static, highly constrained navigation environments [10] with 300 environments containing dynamic obstacles with different motion profiles. These environments are generated to cover a wide variety of obstacle motion profiles of different navigation difficulties, which are systematically quantified by a set of difficulty metrics. We develop a data collection pipeline that records navigation trials performed by human teleoperators, to provide a reference of human performance at dynamic obstacle avoidance. The resulting training data may also be useful for developing learning-based navigation systems. Finally, we provide results of four classical and learning-based navigation systems in DynaBARN, including Dynamic Window Approach (DWA) [13], Timed Elastic Band (TEB) [14], a reinforcement learning approach [15], and a vanilla behavioral cloning method using human demonstrations collected from the pipeline we provide [16], which can serve as baselines for future studies. In summary, this paper makes the following three main contributions.

\begin{itemize}
  \item A benchmark dataset (DynaBARN) of 60 pre-generated dynamic environments in simulation for metric ground navigation, along with an environment generation pro-
\end{itemize}
cedure to generate new dynamic environments,
- A data collection pipeline to collect human-demonstrated navigation trials in the simulated environments,
- Baseline results from a set of existing navigation systems in DynaBARN.

II. RELATED WORK

In this section, we discuss existing navigation methods in dynamic environments as well as testbeds for dynamic obstacle avoidance.

A. Navigation in Dynamic Environments

Robots researchers have investigated methods for dynamic obstacle avoidance for decades. The social force model [17] determines a pedestrian’s motion influenced by the motion of other pedestrians as well as social groups. In contrast, ORCA [12] and RVO [18] are reactive methods for crowded navigation where each agent takes a share of the responsibility for avoiding pairwise collisions and assumes the other agents will reciprocate.

More recently, learning based approaches for dynamic obstacle avoidance have emerged [19]. CADRL [20] uses Deep Reinforcement Learning (DRL) for collision avoidance; it has been extended to include socially-aware collision avoidance [21] by inducing social norms such as passing on the right, overtaking, and crossing. Moreover, DRL dynamic obstacle avoidance methods have been used with conventional waypoint generators so they can be used in conventional navigation systems [22]. The ability to guide or follow a human has also been incorporated into dynamic obstacle avoidance methods for assistive robotics [23]. In addition to DRL methods, inverse reinforcement learning has also been used for socially compliant robot navigation among pedestrians by modeling human behaviors in terms of a mixture distribution that captures both the discrete navigation decisions, such as going left or right, as well as the natural variance of human trajectories [24].

B. Testbeds for Dynamic Obstacle Avoidance

While testbeds for navigation and obstacle avoidance [10], [25], [26] in static environments [27]–[34] have been used to benchmark research progress, simulation testbeds for dynamic obstacle avoidance have recently been of interest in the research community, concurrent with the interest in developing dynamic obstacle avoidance and social navigation methods.

For social navigation around humans, PedSim [35] simulates crowds of pedestrians and incorporates methods of individual and group behaviors into an environment. PedSim does not incorporate social scenarios into its model. As a result, many subsequent simulation testbeds, such as SEAN 2.0 [36], Arena-Bench [37], and SocNavBench [38], use the social forces model along with PedSim to simulate social scenarios and interactions. However, all pedestrians (obstacles) in the aforementioned simulators have similar motion profiles. For instance, in Arena-Bench, the obstacle speeds are set to a constant 0.3 m/s, and in SEAN 2.0 when the pedestrians do move they move at a constant 1.4 m/s. These testbeds do not incorporate obstacles that have different motion profiles, whereas in reality, a robot must be able to avoid colliding with dynamic obstacles moving at different speeds and with path irregularities. Consider the fact that slowly walking pedestrians and fast swerving scooters may appear at the same time on a sidewalk; or at a busy intersection, cars, motorcycles, bicycles, pedestrians, or pets may move in completely different manners. DynaBARN contains obstacles moving not only at different speeds, but also changing speeds between waypoints to test a navigation system’s robustness to erratic changes. These changes in speed and trajectory are tunable based on a set of intuitive difficulty metrics.

Multi-agent path planning testbeds are also used for CADRL [20] and its socially aware counterpart [21]. However, these testbeds aim at testing multi-agent systems in which each agent learns to navigate while avoiding collisions with the other agents: the agents’ policies are trained together in the same simulation to create one shared policy. DynaBARN does not feature multi-agent path planning, in the sense that dynamic obstacles are not treated as agents, which will react to the robot and other obstacles. Instead, the obstacles’ motions are predetermined by waypoints and it is the navigation system’s responsibility to robustly avoid collisions with obstacles of a variety of motion profiles.

III. APPROACH

In this section, we describe our method of creating the DynaBARN environments. In Sec. III-A, we describe the way the obstacle trajectories are generated from user-specified metrics. In Sec. III-B, we describe how to create environments from the generated obstacle trajectories. In Sec. III-C, we introduce a method to quantify the difficulty of an environment in DynaBARN. In Sec. III-D we describe the human demonstration collection pipeline.

A. Obstacle Trajectory Generation

Motion profiles for the obstacles are the trajectories represented by multiple using multiple waypoints, each of which contains two values: the position and timestamp of the obstacle. Formally, we define a trajectory for an obstacle as a sequence of waypoints \((c_i)_{i=0}^n\) with \(c_i = ((x_i, y_i), t_i)\), where \((x_i, y_i)\) is the coordinate point and \(t_i\) is the timestamp of the obstacle at that coordinate point. We assume all obstacles are cylindrical with the base parallel to the ground and therefore ignore their orientation for simplicity. In this work, the environment is a 20 \(\times\) 20 meter grid world. The trajectories are generated using a polynomial function that is fit to randomly sampled points within the grid. The number of randomly sampled points depends on the order of the polynomial. More specifically, \(n + 1\) randomly sampled points determine the equation of a polynomial of degree \(n\). Thus, from a range of orders given by the user, \(\text{order}_{\text{min}}\) and \(\text{order}_{\text{max}}\) as the minimum and the maximum orders respectively, the method samples a random order \(n\) and randomly samples \(n+1\) points...
from the $[-10,10] \times [-10,10]$ grid. A trajectory generated by a lower degree polynomial will be smoother resembling a straighter line than a trajectory made from a higher degree polynomial, which will look less smooth and more erratic. Once the polynomial is fit to the randomly-sampled points, the method then calculates the $(x_i,y_i)$ coordinate points at every integer $x$-value in $[-10,10]$ using the polynomial and then puts them into an array $c$. The method then takes the coordinate points where the polynomial intersects the edges of the $20 \times 20$ grid and appends them to the array $c$. Any points in $c$ outside of the $20 \times 20$ grid as a result of the polynomial fitting are removed. The array $c$ is then, with equal probabilities, sorted either descendingly or ascendingly with respect to the x-values. This random sorting avoids the bias towards environments that only have obstacles move in one direction. We use a straight line to approximate the movement between two points. Examples of trajectories in one direction. We use a straight line to approximate the movement between two points. Examples of trajectories in one direction.

For each subsequent coordinate point in $c$, the obstacle reaches each waypoint. The first point in $c$ is assigned the time 0 seconds since it is the starting waypoint. For each subsequent coordinate point in $c$, the distance $d$ between the current point $c_i$ and the previous coordinate point $c_{i-1}$ is calculated. The method then samples the speed (meters per second) $s \sim N(\text{avg speed}, \text{avg std}^2)$ from $c_{i-1}$ to $c_i$, where $\text{avg speed} \sim \mathcal{U}\{\text{speed}_\text{min}, \text{speed}_\text{max}\}$ and $\text{avg std} \sim \mathcal{U}\{\text{std}_\text{min}, \text{std}_\text{max}\}$. $N$ is a normal distribution parameterized by mean $\text{avg speed}$ and variance $\text{avg std}^2$, and $\mathcal{U}$ is a continuous uniform distribution. The parameters $\text{speed}_\text{min}, \text{speed}_\text{max}, \text{std}_\text{max},$ and $\text{std}_\text{min}$ are all given by the user. A trajectory with a high standard deviation will cause less predictable changes in speed from waypoint to waypoint. Additionally, the speed is limited below by $\text{speed}_\text{min}$ and above by $\text{speed}_\text{max}$.

Once $s$ is sampled, the time taken by an obstacle to travel from $c_{i-1}$ to $c_i$ can be computed as $\Delta t_i = \frac{d}{s}$. The timestamp of the next waypoint can, therefore, be computed recursively by $t_i = t_{i-1} + \Delta t_i$ to get the overall time at which the obstacles will reach $(x_i, y_i)$. We can then create a trajectory for an obstacle that is composed of a sequence of waypoints $\{c_i\}_{i=0}^N$ in which each waypoint has a position and a timestamp which the obstacle reaches said position. The obstacle trajectory generation process is shown in Alg. [1]

### B. Environment Generation

To generate an environment in DynaBARN, the user inputs $\text{obstacles}_{\text{max}}$ and $\text{obstacles}_{\text{min}}$ which are the maximum and minimum numbers of obstacles, respectively, for the environment. Then for each obstacle, a trajectory of waypoints is created using the method described in Sec. III-A. The result is a list of trajectories, one for each obstacle.

### C. Difficulty

a) **Motion Profile Difficulty**: We provide a method to quantify the difficulty of each environment in DynaBARN. There are 6 parameters that can be set for each obstacle in DynaBARN as described in Sec. III-A: the maximum and minimum for each of speed ($\text{speed}_\text{max}$ and $\text{speed}_\text{min}$), standard deviation of speed between waypoints ($\text{std}_\text{max}$ and $\text{std}_\text{min}$), and order ($\text{order}_{\text{max}}$ and $\text{order}_{\text{min}}$). An obstacle with a higher average speed will be more difficult for a robot to react to compared to one with a lower average speed. Similarly, it is more difficult for a navigation system to avoid colliding with an obstacle with a larger standard deviation rather than one with a smaller one. For instance, with a high standard deviation, an obstacle can change between high and low speeds from waypoint to waypoint, making its motion relatively unpredictable for a robot navigation system. Moreover, a trajectory based on a higher order polynomial will have a less straightforward path and will also be less predictable to a robot navigation system. Using these parameters, we can select ranges that divide an obstacle’s motion profiles into two categories: easy and hard. Based on the Jackal robot which we use to conduct navigation tasks (see Sec. IV for specifications), a categorization of the motion profiles of the obstacles can be designed as follows:

Of course these motion profiles can be changed based on the user’s preferences.

b) **Overall Environment Difficulty**: The amount of obstacles in an environment can also affect difficulty, as shown
Fig. 1: An example of the process of creating coordinate points and trajectories. In the top row, 4 points are randomly sampled from the graph. Using these 4 points, the 3-degree polynomial is fit, and we get the remaining waypoints at every whole number x-values from \([-10, 10]\), as shown by the blue points in the graph 2nd from the left. Notice in the bottom right graph for x-values in between -5 and -2.5, the corresponding coordinate points for the polynomial lie outside the \([-10, 10] \times [-10, 10]\) grid and are ignored, so the trajectory between these two points remains a straight line. In the graph 3rd from the left, we see the trajectory of the obstacle, shown in blue, will travel from (10,0) to (-10,-10). More trajectories are generated in a similar fashion, and the last graph on the right displays all the trajectories for that environment.

<table>
<thead>
<tr>
<th>Easy</th>
<th>Hard</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1, 2]</td>
<td>[3, 4]</td>
</tr>
<tr>
<td>[0.5, 1.0]</td>
<td>[1.0, 2.0]</td>
</tr>
<tr>
<td>[0.01, 0.1]</td>
<td>[0.1, 0.2]</td>
</tr>
</tbody>
</table>

TABLE I: Easy and Hard Obstacle Motion Profiles

in [37]. An increased number of obstacles will naturally make the environment more difficult to navigate, and an environment with many obstacles that have a hard motion profile will be even more difficult. To determine difficulty, we first choose the number of obstacles for an environment and then choose the motion profiles for the obstacles. A lower number of obstacles with easy motion profiles will result in easy difficulty while a lower number of obstacles but hard motion profiles will result in medium difficulty. Similarly, an environment with a high number of obstacles with easy motion profiles will also result in medium difficulty, but an environment with both high number of obstacles and hard motion profiles will result in a hard environment difficulty. Fig 2 shows a tree-diagram for determining the difficulty.

D. Human Demonstration Collection

We develop a human demonstration data collection pipeline for future researchers to collect human demonstrations of navigation behaviors in dynamic obstacle environments. Our open-sourced data collection pipeline automatically sets up the ROS environment and installs the required packages for running a simulated unmanned ground vehicle, a Clearpath Jackal, in Gazebo simulation. After finishing the setup, users will be prompted to choose an environment number to open the corresponding dynamic environment. After the selection, Gazebo will be launched and a Jackal will be spawned at (0.0, 11.0), while the navigation goal is set to (0.0, -9). The user will need to drive through the moving obstacles (red cylinders in Fig 3) and try to reach the goal (the green dot in Fig 3) using a joystick. When the Jackal reaches the goal, Gazebo will stop and exit. The human demonstration data is recorded as ROS bag files saved in the local folder. In the ROS bag file, all topics relevant to navigation have been recorded, including front/scan, cmd_vel, and other move_base related topics, which can be used to train an imitation learning policy. The human demonstration pipeline can be extended to any Gazebo world, such as the static BARN environments [10], as long as the world has a goal object. While future researchers can use our pipeline to collect navigation demonstration data collection from a wider variety of human experts, for the ease of use, we collect 60 demonstration trials for DynaBARN from two co-authors of the paper.

IV. EXPERIMENTS AND RESULTS

In this section, we implement and benchmark four commonly-used baselines for dynamic obstacle avoidance tasks using DynaBARN. We sample 60 environments out of the 300 generated worlds in DynaBARN to use for our baselines: 20 easy, 20 medium, and 20 hard. The four baselines are (1) Dynamic Window Approach (DWA) [13]; (2) Timed Elastic Band (TEB) [14]; (3) end-to-end

1Link to the Human Demonstration Collection pipeline repository is at https://github.com/kevinhou912/ROS-Jackal-Data_Collection-Local.git
Reinforcement learning (RL) [15]; and Behavior Cloning (BC) [16]. The implementation details of the baselines are as follows:

a) Two Classical Local Planners: We employ two classical local planners: Dynamic Window Approach (DWA) [13] and Timed Elastic Band (TEB) [14]. These local planners are commonly used by the community due to their reliability in most navigation scenarios with open-sourced implementations that are integrated with the move_base navigation stack [40]. The hyper-parameters of the two local planners are manually tuned to improve their performances in the hard environments. Then, the hyper-parameters are fixed for all the experiments.

b) Behavior Cloning: The demonstration data is composed of 60 trajectories, with each collected from sixty dynamic environments, and another 60 trajectories collected from randomly sampled 300 BARN environments [10]. We learn a policy \( \pi := S \rightarrow A \) to perform local motion planning. Here, \( S \) is the state space with each state \( s \in S \) represented by a tuple \((l, g_x, g_y)\), where \( l \) is the 720-dim laser scan and \((g_x, g_y)\) are the x-y coordinates of the local goal location provided by Dijkstra’s global planner; \( A \) is the action space with each action \( a = (v, w) \in A \), where \( v \in [-0.5, 2] \) and \( w \in [-3.14, 3.14] \) encode the linear and angular velocities of the robot respectively. The ranges of \( v \) and \( w \) are limited by the physical property of the Jackal robot. We represent such a policy by a multi-layer perceptron (MLP) neural network. The architecture of the neural network is shown in Fig. 4.

More specifically, the 722-dimensional input is fed into a feature extraction network composed of three fully-connected layers with 512, 256, and 128 hidden units respectively. Then, the 128-dim feature embedding is fed into an actor network that connects a 64-hidden-unit layer and a 32-hidden-unit layer sequentially, and outputs a two-dimensional vector as the action. All the hidden layers are followed by the Tanh activation [41]. We collect one human-demonstrated trajectory from each of the 60 dynamic environments in DynaBARN and another one from each of the 60 randomly selected static environments in BARN [10].

c) Reinforcement Learning (RL): we also include a navigation policy learned from self-supervised reinforcement learning. More specifically, we employ Twin-delayed Deep Deterministic Policy Gradient (TD3) [15]. As one of the state-of-the-art off-policy RL algorithms, TD3 is relatively sample efficient and handles continuous actions by design. We use the same neural network (NN) architecture as the BC policy to represent the policy of the TD3 agent. As seen in Fig. 4 (bottom), to represent the critic network in the actor-critic framework, only the final layer of the actor network is modified to output the value prediction. To facilitate training, we also implement a parallelized training scheme that distributes multiple actors to a computing cluster to speed up the data collection process [30]. We train one navigation policy for 20 million time steps for each difficulty level with 20 training environments, and test them in the same environments they are trained on.
the success rate of TEB drops to almost zero, while DWA still maintains about 17%. Lastly, behavior cloning has the worst performance in all three difficulty levels, which reflects the difficulty of learning high-accuracy collision-avoidance behaviors from limited data.

Fig. 5: Success rate of four baselines: DWA, TEB, TD3, and BC evaluated on easy, medium, and hard environments.

V. CONCLUSIONS

We present DynaBARN, a benchmark for autonomous robot navigation in dynamic environments where obstacles move according to a variety of different motion profiles. In order to test the robustness of autonomous navigation systems against different obstacle movement, we develop a systematic way to generate obstacle motion profiles and dynamic environments of different difficulties. We also provide a human demonstration data collection pipeline and four different navigation systems as baselines, whose performance is benchmarked in DynaBARN. DynaBARN is developed to encourage further research on robust navigation in the presence of moving obstacles, including, and especially, the development of new, more robust algorithms than the baselines.

One limitation of the current version of DynaBARN is that all obstacles are cylindrical, which limits the navigation system’s robustness to obstacles with different shapes. One interesting extension to DynaBARN is to add randomized obstacle shapes. Another limitation is the lack of interaction between the obstacles and the robot: obstacles in the real world may or may not react to the obstacles, and if they do, they may react in different ways, e.g. yielding to the robot earlier or later. Another future extension to DynaBARN is to add a variety of interaction types to the obstacles.

REFERENCES


