An Interpretable Monitoring Framework for Virtual Physics-Based Non-Interfering Robot Social Planning

Rahul Peddi, *Graduate Student Member, IEEE*, and Nicola Bezzo, *Member, IEEE*

***Abstract*—A majority of collision avoidance and motion planning algorithms deployed on autonomous mobile robots tend to be reactive to the presence and motion of nearby dynamic actors. While these algorithms can produce collision free navigation, they do not necessarily follow social protocol and exhibit unnatural motions that force actors to behave very differently from their planned paths. As humans, we can reason about why and how we might *interfere* with others and use this reasoning to alter our motion proactively. We also adapt our motion based on different *priorities*, which affect how we accommodate and interact with each other. In this letter, we propose an approach for a mobile robot to generate similar non-interfering and priority-based behaviors that pertain to how the robot accommodates dynamic actors. We augment a very efficient but reactive virtual physics-based planner with Hidden Markov Models and a Decision Tree-based monitor that: i) predicts if the robot will interfere with a nearby actor, ii) explains what factors cause the prediction, iii) plans priority-based corrective actions, and iv) updates the prediction model at runtime to improve and refine robot behaviors. Our approach is validated with simulations and extensive experiments on ground and aerial vehicles in the presence of dynamic actors.**

***Index Terms*—Collision avoidance, human-aware motion planning, intention recognition, motion and path planning.**

I. Introduction

As autonomous mobile robots become more ubiquitous, we find them serving different purposes within shared environments, including delivering packages in residential and commercial areas or working in warehouses assisting human workers. Typically, safety is achieved through obstacle avoidance behaviors that treat dynamic actors as though they are stationary, which can force the robots themselves and the actors to move in unnatural ways. In this way, a robot can often find itself too close to some actors, deviating unnecessarily, or "frozen," unable to perform its task [1].

As humans, we combine competing notions of accommodating each other's paths while also maintaining our own path or desired speed. We consider these factors simultaneously to navigate in natural, socially-acceptable ways. Moreover, we prioritize how we accommodate others based on our specific environment or task. For example, a doctor rushing to aid an injured person will prioritize speed as much as safely possible. Importantly, we make these decisions without explicitly predicting the paths of other actors, and we can explain and understand why we make these decisions, continually learning and adapting our behaviors as we experience different scenarios. If a robot could reason about its behavior and plan corrective actions in a similar way, it could generate similarly natural and intuitive motion around other actors, while maintaining priorities related to how much it should accommodate to surrounding actors.

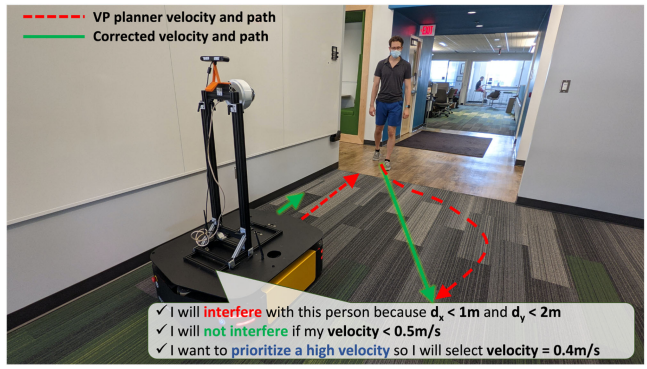


Fig. 1. In our proposed approach, a robot predicts, explains and finds a corrective action on top of a virtual-physics planner to avoid interfering with oncoming actors.

This problem has been explored extensively by the motion planning and machine learning (ML) communities. Virtual physics-based (VP) planners, also known as artificial potential fields, are well known and widely used because they are very efficient and generally effective for performing collision avoidance [2]–[4]. However, VP planners suffer from local minima issues and can result in safe, but undesirable robot behaviors. Advanced ML approaches [5], [6], on the other hand, can explicitly and quickly make predictions about the trajectories of actors, but with more uncertainty due to their black-box models. Moreover, it may not be always necessary or practical to predict the exact trajectory of every nearby actor to generate good robot behaviors [7], [8].

We acknowledge that despite their local minima issues, VP planners are desirable due to their efficiency, general good performance, and low computational burden, and we also recognize that ML techniques can be used for fast prediction, potentially

Manuscript received September 9, 2021; accepted February 3, 2022. Date of publication March 1, 2022; date of current version March 14, 2022. This letter was recommended for publication by Associate Editor A. LaViers and Editor G. Venture upon evaluation of the reviewers' comments. This work was supported by NSF under Grants 1816591 and 1823325, and in part by DARPA under Grant FA8750-18-C-0090. *(Corresponding author: Rahul Peddi.)*

The authors are with the Departments of Engineering Systems and Environment and Electrical and Computer Engineering, University of Virginia, Charlottesville, VA 22904 USA (e-mail: rp3cy@virginia.edu; nb6be@virginia.edu).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2022.3155239, provided by the authors.

Digital Object Identifier 10.1109/LRA.2022.3155239

assisting VP planners, but introduce additional uncertainty due to their typical black-box nature and lack of interpretability. With these considerations in mind, in this work we consider such VP planning methods and augment them to compensate for their local minima issues by including an interpretable decision tree (DT)-based monitoring and planning that bypasses the task of predicting the actor paths and directly predicts and explains if the VP planner will *interfere* with nearby actors. This prediction and explanation is then used to determine how the robot can counteract and avoid interference while maintaining the robot's priorities. To further improve prediction and planning we propose a Hidden Markov Model (HMM)-based framework that learns from observations to adapt the information used by the DTs at runtime. In Fig. 1, we show a motivating example for our work, in which a robot equipped with our approach predicts, explains, and finds a priority-based correction to avoid interfering with an oncoming person.

This work presents three main contributions: 1) the design of an interpretable DT-based monitor that predicts, explains, and finds corrections when a robot will interfere with nearby dynamic actors; 2) the formulation of a priority-based reward function to identify the correction that optimizes how the robot should behave; 3) an HMM-based model to build and update decision trees at runtime and improve predictions and planning over time without storing a large amount of data.

The rest of this letter is organized as follows: in Section II, we discuss related work, in Sections III and IV, we provide preliminaries and formalize the problem addressed in this work. In Section V, we describe our approach and in Sections VI and VII, we present simulation and experimental results. Finally, we conclude and discuss future work in Section VIII.

II. Related Work

The growing presence of autonomous mobile robots in shared environments sparks the need for novel approaches to generate intuitive, socially acceptable robot motion. Virtual physics-based (VP) planners [4], [9] are very efficient and easy to implement, but tend to suffer from local minima issues in which the robot can get trapped. Approaches such as the Social Force Model (SFM) [10] expand this idea by modeling human behaviors. However, SFM-based robot motion planners are often tuned to serve a specific purpose, such as robot companions [3]. Two widely-used planners, *optimal reciprocal collision avoidance* (ORCA) [11] and *dynamic window approach* (DWA) [12], show improvement over basic VP planners, but both can often take circuitous and time-consuming paths to avoid others, sometimes even "freezing" indefinitely [1], unable to find a path to the goal.

More recently, researchers in Machine Learning (ML) have contributed a number of learning-based approaches to solve this problem. Authors in [13]–[15] use deep reinforcement learning (DRL) to plan socially aware collision avoidance. These approaches often generate desirable robot behaviors, but leverage black-box models, making it difficult to understand the reasoning for the outputs. Other works have blended DRL methods with physical obstacle avoidance approaches [16], [17] to achieve more intuitive behaviors, but require a dedicated training phase to improve robot behaviors and are not interpretable. Also leveraging DRL, crowd-based prediction and planning [6], [18] is being explored to handle dense environments using recurrent neural network (RNN)-based and attention-based methods to narrow down how best to navigate through crowds. Several works have also considered how actors will respond to future actions of robot [19], [20], and have emphasized the idea of robot motion that does not disturb nearby humans. We further explore these ideas by directly predicting and explaining whether the robot will *interfere* with the paths of multiple actors, and using the explanation to identify how to best correct robot behaviors in a way that heeds the interfering actors, while avoiding creating interference with other actors.

In previous work [2], we began to address these issues by using Hidden Markov Model (HMM) theory to update at runtime a data-driven predictive model for safe navigation around multiple actors. Our more recent work in this area [7] relaxed the requirement of explicitly predicting actor paths and uses decision trees for control. In this work, we build on these previous contributions and utilize HMM theory to generate and update DTs at runtime, while also using interpretable prediction to support and assist a VP planner on a robot when it is expected to interfere with nearby actors.

III. Preliminaries

Let us consider a mobile robot equipped with a VP planner that is tasked to navigate to a goal in the presence of dynamic actors. Specifically, our planner is inspired by an efficient and scalable virtual spring-mass-damper system borrowed from our previous work [2], in which the input $\boldsymbol{u}$ is comprised of an attractive force that draws the robot towards its goal and repulsive forces that send the robot away from the obstacles. Generally, this formulation is in terms of forces and accelerations, but can be cast as kinematic constraints, depending on the capabilities and inputs of the robots, many of which typically only accept velocity commands. Thus, for ease of implementation on a wide range of robots, we cast the input $\boldsymbol{u}$ as a velocity which is governed by attractive and repulsive effects.

The attractive input that moves the robot from its position $\boldsymbol{p}_r(t)$ towards the goal $\boldsymbol{p}_g$ is computed as follows:

$$\boldsymbol{u}_{att}(t) = k_{att}(||\boldsymbol{p}_r(t) - \boldsymbol{p}_g||)\vec{\boldsymbol{d}}_{\boldsymbol{g}} \tag{1}$$

where $\vec{\boldsymbol{d}}_g$ is the unit vector directed towards the goal and k_{att} is the attractive spring constant. In this work, the resulting velocity vector is limited by a maximum target speed: $||\boldsymbol{u}_{att}(t)|| \leq v^*$. We assume here that the robot is able to quickly reach its target speed, and the desired effect is that the robot slows down and stops once it is close to the goal.

The repulsive inputs are computed as follows:

$$\boldsymbol{u}_{rep}(t) = \begin{cases} k_{rep}(d_{ho}(t))\vec{\boldsymbol{d}}_{\boldsymbol{ho}}, & d_h(t) \leq l_o \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

where k_{rep} is a repulsive spring constant, $d_{ho}(t) = l_h(t) - l_o$ and $l_h(t) = ||\boldsymbol{p}_r(t) - \boldsymbol{p}_h(t)||$ is the distance between robot and dynamic actor h, l_o is the reaction distance threshold, and $\vec{\boldsymbol{d}}_{\boldsymbol{ho}}$ is a unit vector in the direction away from the actor.

We then combine the attractive and repulsive effects along with a damping term with coefficient c_d to compute the input:

$$\boldsymbol{u}(t) = \boldsymbol{u}_{att}(t) + \sum_{j=1}^{n} \boldsymbol{u}_{rep}(t) - c_d\boldsymbol{u}(t-1) \tag{3}$$

VP planners, as it is well-known, are a reactive approach for motion planning and robots using these types of planners can often get stuck in local minima or experience prolonged

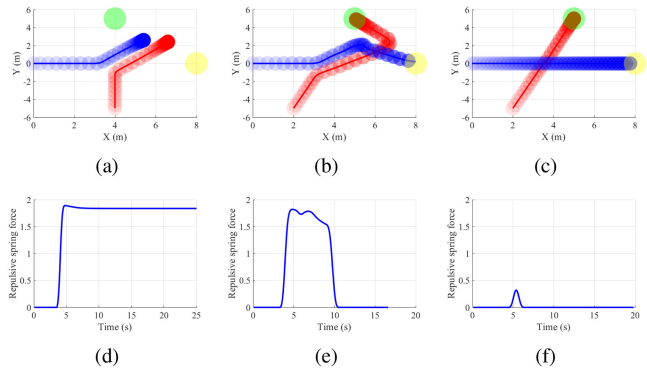


Fig. 2. Examples of VP planner trajectories and repulsive inputs. The trajectories are shown by blue (robot) and red (actor) markers that fade as time passes. Yellow (robot) and green (actor) markers represent the goals.

divergence from the goal. Local minima occur when repulsive and attractive inputs are equal leading to $\boldsymbol{u} = 0$, and prolonged divergence occurs when a repulsive input sends the robot away from the goal and is not approaching $\boldsymbol{u}_{rep} = 0$ quickly enough for the robot to converge to its target. Shown in Fig. 2 are examples of two trajectories in which the VP planner results in a failure (Fig. 2(a)–(b)) due to a local minima and prolonged divergence from path, and one example of a successful VP motion plan (Fig. 2(c)). Figs. 2(d)–(f) display the magnitude of the repulsive input on the robot for each of these trajectories.

IV. Problem Formulation

Consider a mobile robot equipped with the aforementioned VP planner navigating a shared environment with other dynamic actors. With the premises presented in Section III, we would like to directly predict when the VP planner will lead to interference and plan motion accordingly to correct the behaviors of the robot. Formally, the problem is:

Problem 1. Interpretable Interference Monitoring: Design a policy to predict and explain future interfering interactions between a robot and nearby actors and find corrective control actions, $\boldsymbol{u}_{corr}$, for a VP planner such that the following non-interfering conditions are met:

$$\begin{aligned} ||\boldsymbol{p}_r(t) - \boldsymbol{p}_i(t)|| &> \delta_s, \quad \forall i = 1, \ldots, N_h(t) \\ ||\boldsymbol{p}_i(t) - \boldsymbol{p}_i^*|| &\leq \sigma_h, \quad \forall i = 1, \ldots, N_h(t) \\ ||\boldsymbol{p}_r(t) - \boldsymbol{p}_r^*|| &\leq \sigma_r \end{aligned} \tag{4}$$

where $\boldsymbol{p}_r(t)$ and $\boldsymbol{p}_i(t)$, represent the robot and actor i positions, $\boldsymbol{p}_r^*$ and $\boldsymbol{p}_i^*$ are the reference positions of robot and actor i along their respective desired paths, and δ_s is a minimum distance threshold that can be obtained through testing (in this work it is set to 1 m), and σ_r and σ_h are deviation thresholds for robot and actor paths, respectively. In this work, we assume the desired path of actors can be estimated by analyzing changes in direction of motion and checking the minimum distance between robot and actor within the robot's sensing range.

The robot also has different behaviors depending on the assigned priorities. For example, in a conservative case, a robot may accommodate actors more, while a more aggressive robot may prioritize moving faster, accommodating less to the motion of nearby actors. In this work, for ease, we assume that the robot

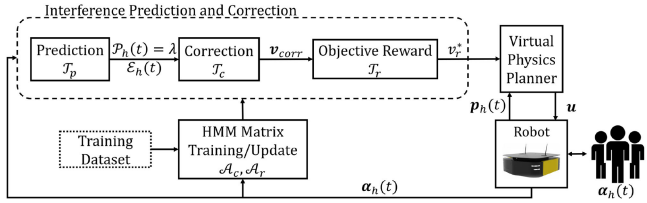


Fig. 3. Block diagram of our priority-based interpretable monitoring and planning framework.

considers commonly observed behavioral priorities [8], [21] that pertain to distance, deviation, and velocity, as follows:

1) Maintain a safe distance from dynamic actors (R_δ).
2) Minimize deviation from the robot's desired path (R_{σ_r}).
3) Minimize actor deviation caused by the robot (R_{σ_h}).
4) Minimize deviation from robot's target speed (R_v).

Given the above priority considerations, we can then formulate the following reward function:

$$R(\boldsymbol{\beta}) = \beta_1 R_\delta + \beta_2 R_{\sigma_r} + \beta_3 R_{\sigma_h} + \beta_4 R_v, \tag{5}$$

where $\boldsymbol{\beta} = [\beta_1 \quad \beta_2 \quad \beta_3 \quad \beta_4]$ is the set of parameters that defines the weights given to each of the 4 considerations. It should be noted that different priorities can be considered beyond those mentioned above.

Then, the problem becomes the following:

Problem 2. Priority-Based Motion Planning: Find a corrective input action within $\boldsymbol{u}_{corr}$ from Problem 1 for the robot that maximizes the objective reward function given a particular set of priorities $\boldsymbol{\beta}$:

$$\boldsymbol{u}^* = \arg\max_{\boldsymbol{u}} R(\boldsymbol{\beta}) \tag{6}$$

Finally, a corollary problem we investigate is how the robot can learn and improve its behavior at runtime in a manner that does not become more computationally expensive over time, and is viable for long term deployment.

V. Approach

Our proposed priority-based interpretable monitoring and planning framework follows the architecture in

At the core of our framework, a series of decision trees is used to: 1) predict and explain interference between the robot and the actors, 2) obtain corrections and 3) select the proper correction for the virtual physics planner based on different priority metrics imposed on the robot. To enable runtime learning and adaptation, a Hidden Markov Model (HMM) is trained both offline and at runtime to characterize probabilities of interference. In the event that the DT predicts that the VP planner will not interfere, then no correction is needed and the robot plans motion according to (3). In what follows we will discuss in more detail each element of Fig. 3.

A. Probability-Based Decision Tree Theory

Decision trees (DTs) are a form of supervised learning that are interpretable, due to their white-box models [22]. DTs take in a set of input variables (attributes) to predict a target output. When the output takes a discrete set of values (classes), the DT is a classification tree. In these trees, the outermost nodes correspond to class labels and internal nodes represent conjunctions of features that return the respective class labels. DTs where the output takes

continuous values are known as regression trees [23], and the outermost nodes in these trees correspond to real numbers. In this work, classification trees are used to predict and explain a future interference, and regression trees are used to estimate priority rewards to decide how to correct robot behaviors.

Typically, DTs are trained with a dataset of historical observations containing input variables and associated labels. A key feature in this work is that DTs are both constructed and constantly updated at runtime, meaning that new data are constantly observed and incorporated into the construction of DTs. Storing historical observation data can become computationally expensive over time, bringing about the need for a faster approach to update and build DTs at runtime.

To this end, we propose a probability model for fast updating and construction of DTs. We take inspiration from the emission matrices within Hidden Markov Models (HMM) [2], in which the state space of the system, $\mathcal{S} \in \mathbb{R}^N$, includes a finite set of unique states $s_i \in \mathcal{S}, i = 1, \ldots, N$, and each state can be associated to an emission, $\mathcal{G} \in \mathbb{R}^M$ and $g_k \in \mathcal{G}$, where $k = 1, \ldots, M$, with $M \in \mathbb{N}$. Given N unique states and M unique emissions, the right-stochastic matrix $\mathcal{A} \in \mathbb{R}^{N \times M}$, in which the sum of rows is 1, lists the probability a_{ik} of obtaining an emission g_k given a state s_i, and is formed as follows

$$\mathcal{A} = \begin{bmatrix} a_{11} & \ldots & a_{1M} \\ \vdots & \ddots & \\ a_{N1} & & a_{NM} \end{bmatrix} \tag{7}$$

In a typical HMM framework, values within $\mathcal{A}$ do not account for the fact that there may be very few occurrences of a particular state, assigning very high probabilities with very little information, which may lead to inaccurate classification. We instead include this consideration by treating emissions as classes and computing classification probabilities that connect a particular state s_i to a class g_k by leveraging a generalized logistic function (GLF) [24], which is an extension of the sigmoid function, of the form:

$$a_{ik}(x) = \frac{L}{1 + Qe^{-b(x - x_0)}}, \tag{8}$$

where L is the maximum value; since this function is used to compute and update probabilities between 0 and 1, a natural choice for this value is $L = 1$. The logistic growth rate is defined by b and x_0 is the midpoint of the curve. The parameter Q can be treated as an initial bias on the midpoint ($x = 0$) of the function. If $Q > 1$, the GLF has a bias towards a lower probability, $a_{ik}(0) < 0.5$, and vice versa. When $Q = 1$, there is no bias and $a_{ik}(0) = 0.5$. The value of Q is a user defined parameter that depends on the application and can be used to tune how conservatively a classification model makes decisions.

The value of x must capture the desired comparison between output classes. In this work, x is computed as a weighted difference between observations of a certain state in each class:

$$x = \begin{cases} \frac{N_{s_{ik^*}} - N_{s_{ik}}}{N_{s_i}} \cdot \min\left(1, \frac{N_{s_i}}{N_l}\right) & N_{s_i} > 0 \\ 0 & N_{s_i} = 0 \end{cases} \tag{9}$$

where $k^* = \neg\lambda$ and $k = \lambda$ represent the classification outputs in our work, $N_{s_{ik^*}}$ and $N_{s_{ik}}$ are the number of non-interfering and interfering observations at state s_i, respectively, and N_{s_i} is the total number of observations of s_i. The value N_l is a buffer chosen to reduce the impact in cases with very few observations

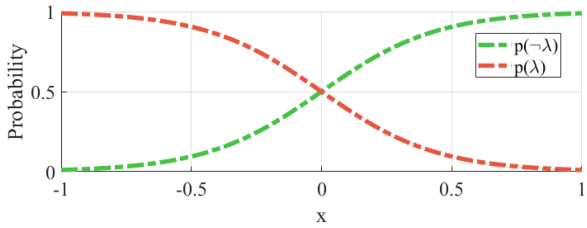


Fig. 4. Generalized logistic function used to compute $P(\neg\lambda)$. Also shown is the complementary curve for $P(\lambda)$

while allowing cases with sufficient observations ($N_{s_i} \geq N_l$) retain their intrinsic values. Shown in Fig. 4 is the GLF that we obtained via experimental testing for this work, where the midpoint is $x_0 = 0$, the parameter $Q = 1$, and the growth rate is $b = 4.5$, giving the following expression: $p(x) = \frac{1}{1+e^{-4.5x}}$.

Since growing DTs requires a dataset containing input states and outputs, we leverage $\mathcal{A}$ to generate a fixed number of data, in which the distribution of outputs corresponds to class probabilities in $\mathcal{A}$. An important benefit of using this method to build and update DTs and probabilities at runtime is that the only data storage requirements for each state are the values contained within (9), which can be adjusted at runtime by updating the number of state and output observations, and recalculating probabilities with (8) and (9). The complexity of this method does not increase with time, and is viable for long term deployment.

B. HMM and DT Training

The training data used to build HMM matrices consists of trajectories of a robot using the aforementioned VP planner with different target speeds in the presence of one actor moving from various initial positions to various goals. In training and simulations, actors are also controlled by the presented VP planner, to account for the fact that other actors are non-hostile and will alter their paths to avoid collisions with the robot.

The attributes (inputs to DTs) are based on the available sensor data about the robot and actors. Specifically, we leverage the following attributes that we have experimentally validated in our previous work [7]:

$$\boldsymbol{\alpha} = [d_x \quad d_y \quad \theta \quad v_h \quad v_r] \tag{10}$$

where d_x, d_y, and θ, are relative x-y positions and heading and v_h and v_r are the actor and robot speeds, respectively. Each unique set of attributes represents one of the states, or rows, within the probability matrix: $\boldsymbol{\alpha}_i = s_i \in \mathcal{S}$, with $i = 1, \ldots, N$. The collected attributes are limited to a small sensing range around the robot since the goal of the robot is to avoid interfering with nearby actors. The attributes are discretized uniformly to prevent an infinite or exploding state space. While uniform discretization is leveraged given the smaller sensing range, it is worth noting that for a robot considering a much larger sensing range or a different application, a variable discretization may be more desirable, so that predictions regarding actors closer to the robot are more accurate and granular than those further from the robot.

The output classes are either interfering (λ) or not interfering ($\neg\lambda$), and are assigned based on a violation of any condition in (4) at any point in a trajectory. The regression outputs describe how well a set of attributes performs for a certain set of priorities (Prob. 2 in this work). Below, we define how each of the rewards within (5) are computed.

The safe distance reward, R_δ, is constructed such that the maximum reward of 1 is assigned when the threshold constraint δ_s is not violated and the reward proportionally decreases as a violation worsens.

The reward for actor path deviation is assigned differently as follows, since the desired effect is lower deviation:

$$R_{\sigma_h} = \begin{cases} 0, & ||\boldsymbol{p}_h(t) - \boldsymbol{p}_h^*|| > \sigma_h \\ e^{-\frac{||\boldsymbol{p}_h(t) - \boldsymbol{p}_h^*||}{\sigma_h}}, & ||\boldsymbol{p}_h(t) - \boldsymbol{p}_h^*|| \leq \sigma_h \end{cases} \tag{11}$$

where σ_h represents the maximum permitted deviation that satisfies conditions in (4). The robot path deviation reward R_{σ_r}, is similar to the actor deviation reward, but instead uses robot current and desired position.

The velocity reward can be represented by any function that has a global maximum at the desired speed and decreases as the robot's speed diverges from v_{des}. We capture this effect with a quadratic function that has a vertex at $(v_{des}, 1)$, as follows: $R_v = -a(v_r - v_{des})^2 + 1$, where a is a constant that determines the rate at which the reward decreases.

During training, an observed set of attributes is rounded to the nearest state, $s_i \in \mathcal{S}$, which is then associated to a class label $(\lambda, \neg\lambda)$ and a priority based reward output (5). Then, the class probabilities are computed with (8) and (9) to populate a classification matrix $\mathcal{A}_c$, and the regression matrix, $\mathcal{A}_r$ is built based on the priority-based rewards obtained during training.

C. Prediction and Explanation

At runtime, to predict and explain interference with a surrounding actor, we build local decision trees with an actor's observed attributes $\boldsymbol{\alpha}_h(t)$. Local trees are more desirable for prediction and explanation, because DTs constructed using the entire state space can contain irrelevant information, reducing the quality of predictions and explanations [23]. The local tree, $\mathcal{T}_p^h$, is trained using a subset of states $\boldsymbol{s}_h(t)$ from $\mathcal{S}$ that are within a distance Δ of the attributes of $\boldsymbol{\alpha}_h(t)$:

$$\boldsymbol{s}_h(t) \subset \mathcal{S} \text{ s.t. } ||\boldsymbol{\alpha}_h(t) - s_i|| \leq \Delta \quad \forall s_i \in \mathcal{S} \tag{12}$$

The parameter Δ defines how close the local training states are to the observed state, and this value should be chosen based on the quality of the training set. It should be noted that too low a Δ may remove relevant data, while too large a Δ provides little benefit over a global tree. Then, the prediction $\mathcal{P}_h(t) \in [\lambda, \neg\lambda]$ is obtained as follows: $\mathcal{P}_h(t) = \mathcal{T}_p^h(\boldsymbol{\alpha}_h(t))$.

An explanation $\mathcal{E}_h(t)$ is then computed by traversing the path Γ from the root of the tree, $\mathcal{V}_0$, to a prediction leaf, $\mathcal{V}_p$, taking the conjunction of each split criterion, c, for the N_i nodes along the path:

$$\mathcal{E}_h(t) = \bigwedge_{k=1}^{N_i} c_k \quad \text{with} \quad \Gamma \quad | \quad \mathcal{P}_h(t) \tag{13}$$

Traversing all paths, $\Gamma_j \in \boldsymbol{q}$ with $j = 1, \ldots, N_{\boldsymbol{q}}$, that lead to the opposite decision provides a set of counterfactuals, $\boldsymbol{\mathcal{C}}_h(t)$:

$$\boldsymbol{\mathcal{C}}_h(t) = \bigvee_{j=1}^{N_{\boldsymbol{q}}} \bigwedge_{k=1}^{N_j} c_k \quad \text{with} \quad \Gamma_j \quad | \quad \neg\mathcal{P}_h(t) \tag{14}$$

where each path Γ_j contains N_j nodes to the leaf [7].

In Fig. 5 we show an example of a decision tree that predicts and explains a future interference with one approaching actor.

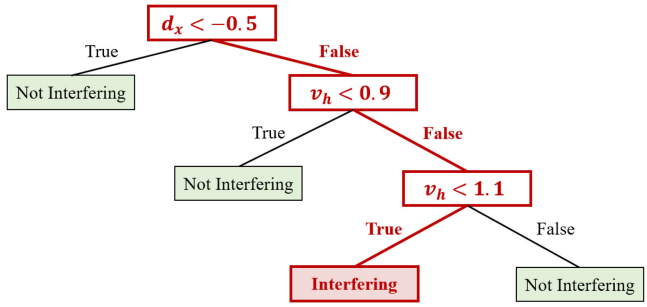


Fig. 5. Prediction and explanation decision tree $\mathcal{T}_p^h$.

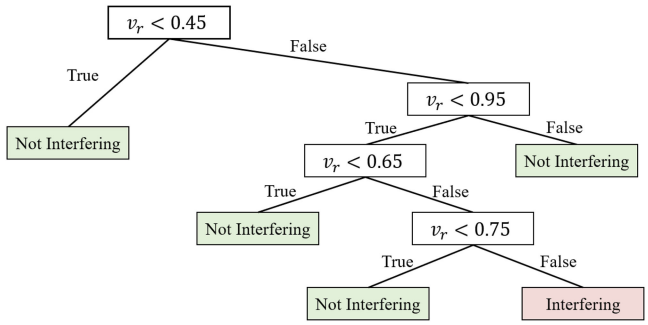


Fig. 6. Correction decision tree $\mathcal{T}_c^h$.

The attributes in this particular example are the following:

$$\boldsymbol{\alpha}_h(t) = \begin{bmatrix} d_x & d_y & \theta & v_h & v_r \\ 0 & -3 & \pi/4 & 1.0 & 0.8 \end{bmatrix}$$

The prediction is $\mathcal{P}_h(t) = \lambda$ (interfering), indicated by the red path in the figure. Through (13), the following explanation is obtained:

$\mathcal{E}_h(t) =$ ***Interfering*** because: $\{d_x > -0.5 \wedge 0.9 < v_h < 1.1\}$

Through (14), the following counterfactuals are then obtained:

$\boldsymbol{\mathcal{C}}_h(t) =$ ***Not Interfering*** when:

$$\begin{array}{l} \{d_x < -0.5\} \quad \vee \\ \{d_x > -0.5 \wedge v_h < 0.9\} \quad \vee \\ \{d_x > -0.5 \wedge v_h > 1.1\} \end{array}$$

D. Counterfactual Analysis and Priority-Based Correction

When the VP planner is predicted to interfere with an actor, the counterfactuals of the DTs tell which attributes, when changed, can revert the prediction. The robot, however, cannot change attributes that are related to actor motion, and can only control its maximum target speed, v_r, which is used to obtain $\boldsymbol{u}_{att}$ (1). To find corrective target speeds for the robot, we build a set of secondary trees, $\mathcal{T}_c^h$ and $\mathcal{T}_r^h$, in which we fix all attributes except v_r and find similar states from $\mathcal{S}$ as done in (12), creating a new set $\boldsymbol{s}_c \subset \mathcal{S}$ (note that $\boldsymbol{s}_h \subset \boldsymbol{s}_c$). In this way, new DTs are still relevant to other local attributes but include varied robot speed, enabling DTs to find from within the finite state space $\mathcal{S}$ via (14) a set of non-interfering target speeds, $\boldsymbol{v}_{corr}^h$.

In Fig. 6, we show the correction tree associated with the example introduced in Section V-C. The robot was moving at a

TABLE I
REWARDS FROM $\mathcal{T}_r^h$

$v_i \in \boldsymbol{v}_{corr}$	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	1.0
Reward	0.601	0.690	0.744	0.747	0.801	0.771	0.732	0.554	0.619

desired speed of $v_r = 0.8$. Through (14), we obtain the following set of counterfactuals:

$\boldsymbol{C}_h(t) =$ ***Not Interfering*** when:

$$\begin{aligned} &\{v_r < 0.45\} \quad \vee \\ &\{0.45 \leq v_r < 0.65\} \quad \vee \\ &\{0.65 \leq v_r < 0.75\} \quad \vee \\ &\{v_r > 0.95\} \end{aligned}$$

In this example, the set of non-interfering speeds from our state space $\mathcal{S}$ is $\boldsymbol{v}_{corr}^h = [0 \quad 0.1 \quad 0.2 \quad 0.3 \quad 0.4 \quad 0.5 \quad 0.6 \quad 0.7 \quad 1.0]$

While each target speed within $\boldsymbol{v}_{corr}^h$ may satisfy the conditions for a non-interfering case, some may result in better or worse performance, depending on the priority behavior of the robot (Problem 2).

The optimal target speed for given priority parameters is computed by comparing the output of non-interfering speds, $v \in \boldsymbol{v}_{corr}^h$, in the reward tree of actor h, as follows:

$$v^* = \arg\max_{v} \mathcal{T}_r^h(\boldsymbol{v}_{corr}) \tag{15}$$

This is then set as the maximum target speed for the attractive input, discussed in Section III.

Shown in Table I is the output of the reward tree with the following parameters $\boldsymbol{\beta} = [0.25 \quad 0.25 \quad 0.25 \quad 0.25]$, taken from the running example in the previous sections (Figs. 5 and 6). The non-interfering speeds are compared using the predicted reward, and through (15), we obtain that $v^* = 0.4$ m/s.

E. Extension to Multiple Actors

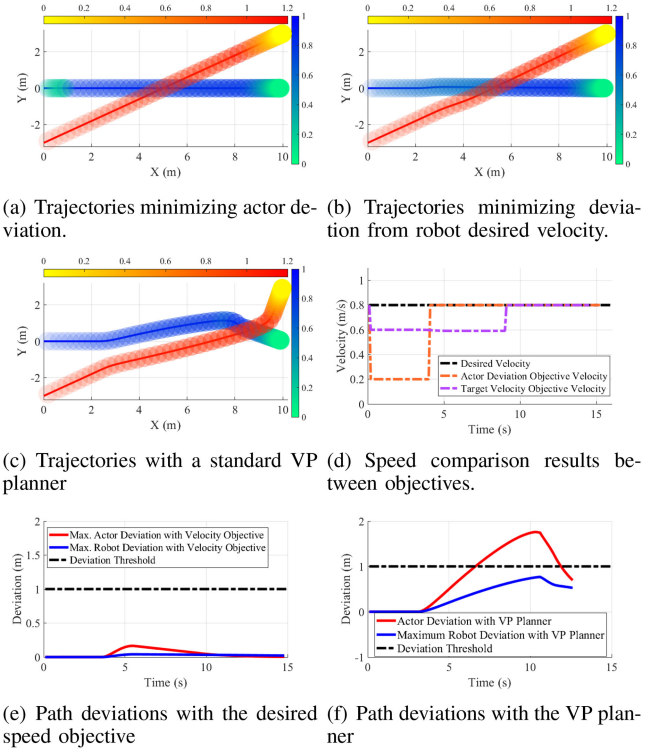
The set of non-interfering speeds for multiple surrounding actors is the intersection between individual sets. Let v_{corr}^h represent a set of non-interfering speeds for a single actor h within the sensing range of the robot. Then the intersection of sets of all actors is represented as follows:

$$\hat{\boldsymbol{v}}_{corr} = \bigcap_h \boldsymbol{v}_{corr}^h \tag{16}$$

The set of combined non-interfering speeds, $\hat{\boldsymbol{v}}_{corr}$ includes speeds for which $\mathcal{P}_h = \neg\lambda \quad \forall h = 1, \ldots, N_h$, where N_h is the number of actors within the sensing range of the robot. Then the optimal target speed for all actors is found similarly as in (15), but now includes summing the rewards:

$$v^* = \arg\max_{v} \sum_{h=1}^{N_h} \mathcal{T}_r^h(\hat{\boldsymbol{v}}_{corr}) \tag{17}$$

There may, however, exist cases in which no global solution can be found, that is $\hat{\boldsymbol{v}}_{corr} = \varnothing$, due to particularly dense conditions and contradicting corrections. In these cases, the system reverts to a fail-safe mode, in which the robot moves at a very low target speed with our VP planner. We treat this as fail-safe because, in addition to our assumption that actors are non-hostile, assigning a low speed has been shown to effectively communicate to actors that the robot is accommodating [8] giving the actors more time to react and perform avoiding maneuvers. However, when actors take upon the burden of avoidance, interfering conditions are often violated. These cases, as with all observations, are recorded and probabilities are updated through (8) and (9) at runtime, refining predictions so that the robot can learn to avoid getting into a similar interfering state in the future.



(a) Trajectories minimizing actor deviation.
(b) Trajectories minimizing deviation from robot desired velocity.
(c) Trajectories with a standard VP planner
(d) Speed comparison results between objectives.
(e) Path deviations with the desired speed objective
(f) Path deviations with the VP planner

Fig. 7. Trajectories and results of robot (cool colors) and actor (warm colors) comparing the presented approach with different objectives and the standard VP planner without our approach. Velocities are indicated by the color-bars within the trajectories.

VI. VIRTUAL SIMULATIONS

In this section we provide several MATLAB simulations to evaluate and compare our approach with other methods. Simulation training included a robot equipped with the previously described VP planner with target speeds in the interval $v_r = [0.0, 1.0]$ m/s, discretized at 0.1 m/s increments. The desired speed was set to $v_{des} = 0.8$m/s. In the training, the robot performed its trajectory in the presence of one actor also running the aforementioned VP planner, which had different initial and final positions in each trajectory with a nominal speed of $v_h = 1$m/s.

In the first simulation, we show the outcome of the running example presented throughout this letter so far (Figs. 5 and 6), in which a robot with a 5 m sensing radius is navigating in the presence of one actor and has a goal at (10, 0) m. The distance and deviation thresholds are $\delta_s, \sigma_h, \sigma_r = 1$m. Fig. 7 compares our approach under two different priority behaviors, and contrasts results with the standard VP planner.

In Fig. 7(a) the robot prioritizes minimizing actor deviation by using priority reward parameters $\boldsymbol{\beta} = [0.2 \quad 0.1 \quad 0.6 \quad 0.1]$. The resulting trajectory consists of a very low target speed of 0.2 m/s, and no deviations. In Fig. 7(b) we repeat the same case study,

TABLE II
RESULTS FROM COMPARATIVE SIMULATIONS

	Success Rate (%)	Average Actor Deviation (m)	Average Max Robot Deviation (m)	Target Speed Deviation (m/s)	Average Added Time (%)
Standard VP Planner	62%	0.792	1.311	0	27%
Neutral Priorities	87%	0.351	0.407	0.598	45%
Maintaining Desired Speed	86%	0.356	0.463	**0.507**	**41%**
Minimizing Actor Deviation	**89%**	**0.298**	**0.283**	0.702	49%
ORCA	75%	0.433	0.981	0.544	41%
DWA	44%	0.361	1.511	0.821	64%

but the robot is now tasked to prioritize maintaining its desired speed ($\boldsymbol{\beta} = [0.2 \quad 0.1 \quad 0.1 \quad 0.6]$). As can be noted in Fig. 7(d) the robot maintains a speed of 0.6 m/s, closer to the desired 0.8 m/s, the maximum actor deviation (Fig. 7(e)) is 0.17 m, and the robot's is 0.05 m due to the underlying VP planners but interference conditions (4) are not violated. Fig. 7(c) shows the VP planner without our approach. The robot interferes with the actor's path significantly, causing a maximum actor deviation of 1.76 m (Fig. 7(f)), which violates the interference threshold.

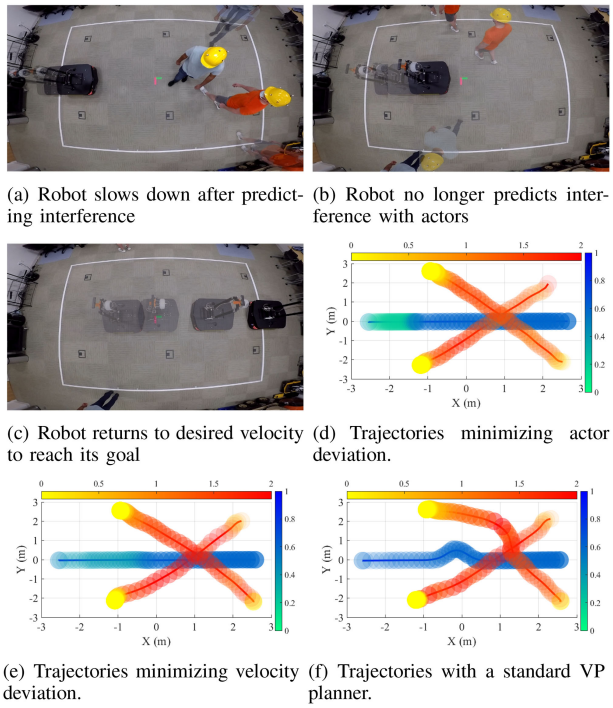
We also extensively tested our approach in dense multi-actor environments, where the robot traverses 12 m, sharing an environment with 10 simulated actors running a standard VP planner to avoid the robot. A video for this simulation is included in the supplemental material. We compare our approach under different priorities with two widely-used planners: ORCA [11] and DWA [12], which is the standard planner on ROS-enabled systems. The results over 10 trials, in which the robot navigates through a total of 100 actors, are shown in Table II.

The success rate is defined as the proportion of total actors that were interfered. When prioritizing minimal actor deviation, our approach outperforms others with an 89% success rate. However, we find that results are similar under different priorities, since the solution set, $\hat{\boldsymbol{v}}_{corr}$, is often limited in dense environments. The actor deviation objective is the most conservative, resulting in lower speeds and higher added time of 49%, when compared to the time an unobstructed path to the goal would have taken. The standard VP planner has the lowest added time, but has a success rate of 64%. In the cases in which our approach fails, we found that the robot was negotiating with dense crowds ($N_h > 6$) within its sensing range and no solutions were available.

The presented approach outperforms ORCA in all metrics, but we do note that ORCA outperforms the standard VP planner, because ORCA expects surrounding actors to follow the same algorithm and perform reciprocal avoiding behaviors. While actors using VP planners are not exactly finding reciprocal velocities, they are partially satisfying ORCA expectations by avoiding the robot. DWA, on the other hand, is searching for a set of admissible velocities given the occupancy in the robot's sensing range. Since this approach is not considering the future motion of dynamic actors, we observe a larger robot deviation, added time, and a poorer success rate, when compared to other approaches.

VII. HARDWARE EXPERIMENTS

The proposed approach was also validated experimentally using unmanned ground (UGV) and aerial (UAV) vehicles in indoor environments. HMM and DT operations were implemented in MATLAB and interfaced with ROS through the ROS Toolbox, and executed at 10 Hz. Multiple experiments were performed, but to conserve space, we only show a few representative cases and more are available in the provided supplemental material. For all experiments, the distance threshold $\delta_s = 1$m and the deviation thresholds $\sigma_r, \sigma_h = 0.5$ m. The training speeds were set to 0.2, 0.4, and 0.6 m/s while $v_{des} = 0.6$ m. The robot starts at $(-2.5, 0)$ m and is tasked to reach a goal at (2.5, 0) m.



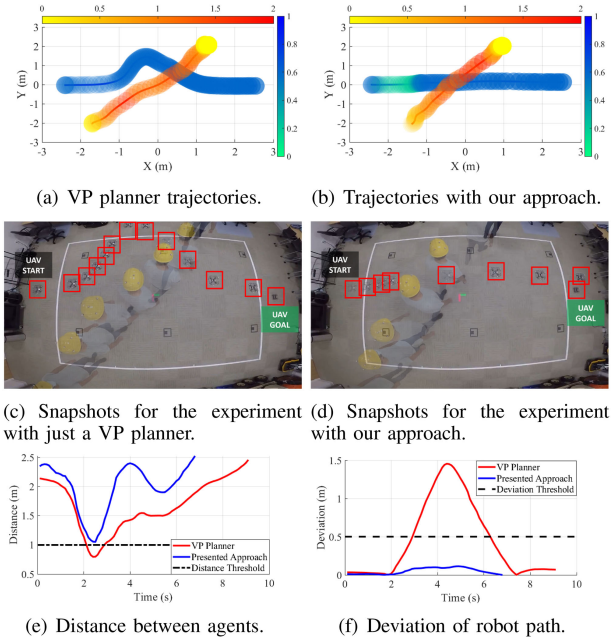
(a) Robot slows down after predicting interference
(b) Robot no longer predicts interference with actors
(c) Robot returns to desired velocity to reach its goal
(d) Trajectories minimizing actor deviation.
(e) Trajectories minimizing velocity deviation.
(f) Trajectories with a standard VP planner.

Fig. 8. Ground vehicle experiment trajectories and snapshots of robot (cool colors) and actor (warm colors) comparing the presented approach with different objectives and the standard VP planner without our approach.

Fig. 8 shows results from ground vehicle experiments on a Clearpath Robotics Ridgeback Omnidirectional Industrial Robot inside our lab tracked by a VICON motion capture system (MOCAP). With the more conservative priorities (Figs 8(a)–(d)), the robot reduces its target speed to $v_r = 0.2$m/s and no deviation on either the robot or the actors paths is detected. When the robot prioritizes maintaining desired speed, our approach only reduces to $v_r = 0.4$ m/s with a minor deviation from its path of 0.04 m. Under the standard VP planner, the robot deviates 0.53 m, and the actor deviates $\sim$1 m from the desired path, violating the 0.5 m threshold.

We also successfully tested the applicability of our approach outside MOCAP settings. We deployed our technique on the same UGV using only the on-board RGB-D camera for person detection and Lidar sensors for localization in the presence of 3 actors. Videos of these experiments and more cases are provided in the supplemental material.

In the UAV experiment, we replicated a failure case shown in simulation (Fig. 7(c)) with an aerial vehicle. We used the DJI Tello mini drone in the same lab environment, with the same goal and parameters. Fig. 9 contains trajectories and snapshots from these experiments. The UAV with the standard VP planner deviates 1.45 m, taking a longer path to reach the goal. With our approach, the robot adapts its speed and deviates only 0.11 m. Notably, the UAV reaches the goal 2 s *faster*, despite slowing down to avoid interference.



(a) VP planner trajectories. (b) Trajectories with our approach.

(c) Snapshots for the experiment with just a VP planner. (d) Snapshots for the experiment with our approach.

(e) Distance between agents. (f) Deviation of robot path.

Fig. 9. UAV experiment trajectories, snapshots, and results.

VIII. Conclusions & Future Work

In this work, we have presented a novel approach that leverages virtual physics planning by compensating for local minima failures through a decision tree-based explainable monitor to create proactive non-interfering behaviors between a robot and nearby dynamic actors. A HMM-based probability model was also introduced to enable DTs to update and improve predictions at runtime. Finally, we presented a reward-based method to trigger different non-interfering behaviors given the priority objectives of the robot. The main benefits of our approach are that it leverages the efficiency and low overhead of VP planning methods and provides human interpretable explanations when interference is detected. This framework scales well in real world settings as demonstrated by the different experiments conducted with different robots with different sensing capabilities. Dealing with multiple actors still remains a challenge in dense situations where a global solution doesn't exist. While in this letter, the robot switches into a fail-safe mode, we note that there can be situations in which a sequence of actions may be able to create non-interfering behavior while maintaining a certain level of performance of the robot. In future work, we plan to expand on these thoughts and further investigate how we can consider crowd interactions and prioritize actors carrying out tasks of varying value in dense environments.

References

[1] P. Trautman and A. Krause, "Unfreezing the robot: Navigation in dense, interacting crowds," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2010, pp. 797–803.

[2] R. Peddi, C. Di Franco, S. Gao, and N. Bezzo, "A data-driven framework for proactive intention-aware motion planning of a robot in a human environment," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 5738–5744.

[3] G. Ferrer, A. Garrell, and A. Sanfeliu, "Robot companion: A social-force based approach with human awareness-navigation in crowded environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2013, pp. 1688–1694.

[4] Y. Huang *et al.*, "A motion planning and tracking framework for autonomous vehicles based on artificial potential field elaborated resistance network approach," *IEEE Trans. Ind. Electron.*, vol. 67, no. 2, pp. 1376–1386, Feb. 2020.

[5] S. Eiffert, H. Kong, N. Pirmarzdashti, and S. Sukkarieh, "Path planning in dynamic environments using generative RNNS and Monte Carlo tree search," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 10263–10269.

[6] S. Liu, P. Chang, W. Liang, N. Chakraborty, and K. Driggs-Campbell, "Decentralized structural-RNN for robot crowd navigation with deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 3517–3524.

[7] R. Peddi and N. Bezzo, "Interpretable run-time prediction and planning in co-robotic environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 2504–2510.

[8] T. Kruse, A. Kirsch, H. Khambhaita, and R. Alami, "Evaluating directional cost models in navigation," in *Proc. ACM/IEEE Int. Conf. Hum.-Robot Interact.*, 2014, pp. 350–357.

[9] A. C. Woods and H. M. La, "A novel potential field controller for use on aerial robots," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 49, no. 4, pp. 665–676, Apr. 2019.

[10] D. Helbing and P. Molnar, "Social force model for pedestrian dynamics," *Phys. Rev. E*, vol. 51, no. 5, 1995, Art. no. 4282.

[11] J. van den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal N-body collision avoidance," in *Robotics Research*. Berlin, Germany: Springer, 2011, pp. 3–19.

[12] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robot. Automat. Mag.*, vol. 4, no. 1, pp. 23–33, Mar. 1997.

[13] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 1343–1350.

[14] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 3052–3059.

[15] T. van der Heiden, F. Mirus, and H. van Hoof, "Social navigation with human empowerment driven deep reinforcement learning," in *Proc. Int. Conf. Artif. Neural Netw.*, 2020, pp. 395–407.

[16] A. J. Sathyamoorthy, U. Patel, T. Guan, and D. Manocha, "Frozone: Freezing-free, pedestrian-friendly navigation in human crowds," *IEEE Robot. Automat. Lett.*, vol. 5, no. 3, pp. 4352–4359, Jul. 2020.

[17] S. S. Samsani and M. S. Muhammad, "Socially compliant robot navigation in crowded environment by human behavior resemblance using deep reinforcement learning," *IEEE Robot. Automat. Lett.*, vol. 6, no. 3, pp. 5223–5230, Jul. 2021.

[18] C. Chen, Y. Liu, S. Kreiss, and A. Alahi, "Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning," in *Proc. Int. Conf. Robot. Automat.*, 2019, pp. 6015–6022.

[19] T. Kruse, A. K. Pandey, R. Alami, and A. Kirsch, "Human-aware robot navigation: A survey," *Robot. Auton. Syst.*, vol. 61, no. 12, pp. 1726–1743, 2013.

[20] B. Petrak, G. Sopper, K. Weitz, and E. André, "Do you mind if i pass through? Studying the appropriate robot behavior when traversing two conversing people in a hallway setting," in *Proc. 30th IEEE Int. Conf. Robot Hum. Interactive Commun.*, 2021, pp. 369–375.

[21] M. Herman, V. Fischer, T. Gindele, and W. Burgard, "Inverse reinforcement learning of behavioral models for online-adapting navigation strategies," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2015, pp. 3215–3222.

[22] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Evanston, IL, USA: Routledge, 2017.

[23] C. D. Franco and N. Bezzo, "Interpretable run-time monitoring and replanning for safe autonomous systems operations," *IEEE Robot. Automat. Lett.*, vol. 5, no. 2, pp. 2427–2434, Apr. 2020.

[24] M. Cutler and J. P. How, "Autonomous drifting using simulation-aided reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2016, pp. 5442–5448.