Formalizing Trajectories in Human-Robot Encounters via Probabilistic STL Inference

Alexis Linard, Ilaria Torre, Anders Steen, Iolanda Leite and Jana Tumova

***Abstract*— In this paper, we are interested in formalizing human trajectories in human-robot encounters. We consider a particular case where a human and a robot walk towards each other. A question that arises is whether, when, and how humans will deviate from their trajectory to avoid a collision. These human trajectories can then be used to generate socially acceptable robot trajectories. To model these trajectories, we propose a data-driven algorithm to extract a formal specification expressed in Signal Temporal Logic with probabilistic predicates. We evaluated our method on trajectories collected through an online study where participants had to avoid colliding with a robot in a shared environment. Further, we demonstrate that probabilistic STL is a suitable formalism to depict human behavior, choices and preferences in specific scenarios of social navigation.**

***Index Terms*— Temporal Logic Inference, Signal Temporal Logic, Human-Robot Interaction.**

I. Introduction

The increasing popularity of deploying robots in social environments leads to new research challenges, such as prediction of human behavior in crowds or identification of different navigation styles [1], [2] and thus enable robot controllers that would react efficiently and appropriately to human behavior [3]. However, very often, models of human behavior often lack interpretability. For instance, deep neural networks [4], despite being undoubtedly powerful at making good predictions, are by essence *black boxes* making decisions that are hardly decipherable. At the same time, accountability and interpretability are vital for human-robot interaction to be safety-critical. In this paper, we aim to retrieve interpretable models of human trajectories via temporal logics that are highly expressive yet retain the structure and some resemblance to natural language.

Temporal logics have been adopted to specify complex robotic tasks for more than a decade [5]. Recently, Linear Temporal Logic has also been used to express social norms in human-robot interaction [6] and describe social robot behaviors [7]. Signal Temporal Logic (STL) is a richer variant that can express system properties that include bounds on time and values of system parameters. It is defined over continuous signals [8], and hence can capture properties of trajectories in a 2D workspace, i.e. 2-dimensional signals. In robot motion planning and control, desired behavior specifications and preferences expressed in STL are typically designed manually based on domain and task knowledge.

Temporal logic inference methods can synthesize specifications of desired trajectories from data. Given a series of finite-time signals, STL inference algorithms focus on finding a formula that optimally describes these signals. Proposed inference methods span two categories: parameter synthesis, that is, given the structure of an STL formula to learn parameters to classify signals correctly [9]; and learning both the structure and parameters of the formula. The latter has been approached, e.g., via offline and online supervised learning approaches based on decision trees [10], evolutionary algorithms [11], unsupervised learning approaches [12], or active learning [13], where the authors consider a setting where data is *a priori* not available but interactively retrieved through queries between the learning algorithm and the signal-producing system.

Probabilistic extensions of STL provide tools to model stochastic properties of systems and enforce probabilistic guarantees on achieving them [14], as well as to model uncertainties regarding the environment [15]. One of the probabilistic extensions of STL uses predicates that bound the probability of a state given a normal distribution. In this paper, this is the extension we refer to when talking about probabilistic STL. We propose using this probabilistic STL to model human trajectories as a good compromise between the rigor and interpretability of temporal logics and the essence of human behavior, which can be hard to predict at all times. However, to the best of our knowledge, learning of probabilistic STL formulae has not been addressed before.

This paper aims to bridge the gap between models describing human behavior used to deploy robots in social contexts and formal method-based robot control to accomplish complex missions with guaranteed performance. We aim to learn a specification that reasons on probabilities of human behavior and choices. Our contributions can be summarized as follows: 1) we designed an algorithm to learn probabilistic STL specifications from data, and 2) to evaluate our algorithm, we collected data in a study inspired by the game-theoretic "Game of Chicken" [16], where users in a simulated environment had to navigate a space shared with a robot. The crowd workers controlled a virtual avatar that walked towards a robot moving in the opposite direction along the same path. To avoid a collision, users had to deviate their trajectory. We demonstrated how STL with probabilistic predicates is a suitable formalism for modeling human behavior in the context of social navigation. Thus, the learnt specifications can later be implemented in socially-

The authors are with the KTH Royal Institute of Technology, SE-100 44, Stockholm, Sweden, with the division of Robotics, Perception and Learning, Faculty of Electrical Engineering and Computer Science. This work was supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation and the Swedish Research Council (VR). {linard,ilariat,astee,iolanda,tumova}@kth.se

aware robot controllers to improve the quality of human experience in human-robot encounters.

Closely related work includes [17], where the authors introduced a navigation system consisting of a planner that attempts to optimize a robot's trajectory considering social constraints such as safety, time-to-collision and directional compatibility of human-robot motion. An extension [18] considers reactive planning and decision making in a human-robot co-navigation setting, where humans are treated as co-existing agents. However, these works focus on navigation only and do not account for more complex robot task specifications. Another approach is presented in exploratory experimental works studying how autonomous robots navigate around people in real-world environments [19] and about the way a robot should approach humans [20] from an empirical perspective.

II. Preliminaries

Let $\mathbb{R}$ and $\mathbb{N}$ be the set of real and natural numbers, respectively. We use discrete notion of time throughout this paper, and time intervals are in the form $I = [t_1, t_2] \subset \mathbb{N}$, $t_1, t_2 \in \mathbb{N}, t_1 \leq t_2$. $[\tau+t_1, \tau+t_2]$ is denoted by $\tau+I$, $\tau \in \mathbb{N}$. An n-dimensional, real, finite-time, discrete-time signal σ is defined as a sequence of real values $\sigma : \sigma(t_0)\sigma(t_1)\sigma(t_2)\cdots$, where $\sigma(t_i)$ is the value of signal σ at time t_i s.t. $\sigma(t_i) \in \mathbb{S}, \mathbb{S} \subset \mathbb{R}^n$, $t \in \mathbb{N}$. We consider $\mathbb{S} = \mathbb{R}^n$. The set of all signals with values taken in $\mathbb{S}$ is denoted $\mathcal{S}$.

A. Probabilistic Models

A multivariate normal distribution (or *n-dimensional* normal distribution) of an n-dimensional random vector $\mathbf{X} = (X_1, \ldots, X_n)^T$ is defined as:

$$\mathbf{X} \sim \mathcal{N}(\mu, \Sigma) \tag{1}$$

where $\mu \in \mathbb{R}^n$ is an n-dimensional mean vector and $\Sigma \in \mathbb{R}^{n \times n}$ an $n \times n$ covariance matrix such that:

$$\begin{aligned} \mu &= E[X] = (E[X_1], \ldots, E[X_n])^T \\ \Sigma_{i,j} &= E[(X_i - \mu_i)(X_j - \mu_j)] \end{aligned} \tag{2}$$

Given two n-dimensional normal distributions $\mathcal{N}_0(\mu_0, \Sigma_0)$ and $\mathcal{N}_1(\mu_1, \Sigma_1)$, the Kullback–Leibler divergence from $\mathcal{N}_0$ to $\mathcal{N}_1$ is defined as:

$$\begin{aligned} D_{\mathrm{KL}}(\mathcal{N}_0 \parallel \mathcal{N}_1) = \frac{1}{2}\Bigg(& \operatorname{tr}\left(\Sigma_1^{-1}\Sigma_0\right) \\ & + (\mu_1 - \mu_0)^{\mathsf{T}} \Sigma_1^{-1} (\mu_1 - \mu_0) - n + \ln\left(\frac{\det \Sigma_1}{\det \Sigma_0}\right)\Bigg) \end{aligned} \tag{3}$$

where $tr(\Sigma)$ denotes the trace of an $n \times n$ square matrix Σ, and $\det \Sigma$ its determinant.

The Jensen-Shannon Divergence between $\mathcal{N}_0$ and $\mathcal{N}_1$ is defined as:

$$D_{\mathrm{JS}}(\mathcal{N}_0 \parallel \mathcal{N}_1) = \frac{1}{2} D_{\mathrm{KL}}(\mathcal{N}_0 \parallel M) + \frac{1}{2} D_{\mathrm{KL}}(\mathcal{N}_1 \parallel M) \tag{4}$$

where $M = \frac{1}{2}(\mathcal{N}_0 + \mathcal{N}_1)$ is the average of the two distributions. Note that in [21], $\sqrt{D_{\mathrm{JS}}}$ has been proven to be a metric.

The probability at time t_i of an n-dimensional signal σ to fit $\mathcal{N}(\mu, \Sigma)$ is defined as the probability of some unseen data point $y(t_i)$ of being "closer" to the distribution than $\sigma(t_i)$:

$$\mathbb{P}(\sigma(t_i) \mid \mathcal{N}(\mu, \Sigma)) = 1 - \mathbb{P}(Q \leq (\sigma(t_i) - \mu)^T \Sigma^{-1} (\sigma(t_i) - \mu))$$

with $Q = (y(t_i) - \mu)^T \Sigma^{-1} (y(t_i) - \mu)$. Note that Q can be approximated as a chi-square distribution with n degrees of freedom. The definition of $\mathbb{P}(\sigma(t_i) \mid \mathcal{N}(\mu, \Sigma))$ follows principles related to the Mahalanobis distance.

B. Probabilistic Signal Temporal Logic

Similarly to [14], we define an extension of STL [8] to include probabilistic predicates of the form χ^{ϵ}, where $\chi = \mathcal{N}(\mu, \Sigma)$ and $\epsilon \in [0, 1]$, $\epsilon \in \mathbb{R}$ is the probability threshold for a signal $\sigma(t_i)$ at time t_i of fitting χ.

In the rest of the paper, we will consider the following fragment of STL with probabilistic predicates:

$$\phi ::= \top \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \Diamond_I \chi^{\epsilon} \tag{5}$$

where $\top$ is the Boolean *True* constant, $\vee$ and $\wedge$ are the Boolean operators for disjunction and conjunction, respectively; and $\Diamond_I$ is the temporal operator *eventually* over bounded interval I.

The qualitative semantics of this fragment are defined as:

$$\begin{aligned} \sigma(t_i) \models \Diamond_I \chi^{\epsilon} \quad &\Leftrightarrow \quad \exists i' \in t_i + I \text{ s.t. } \mathbb{P}(\sigma(t_{i'}) \mid \chi) \geq \epsilon \\ \sigma(t_i) \models \phi_1 \vee \phi_2 \quad &\Leftrightarrow \quad (\sigma(t_i) \models \phi_1) \vee (\sigma(t_i) \models \phi_2) \\ \sigma(t_i) \models \phi_1 \wedge \phi_2 \quad &\Leftrightarrow \quad (\sigma(t_i) \models \phi_1) \wedge (\sigma(t_i) \models \phi_2) \end{aligned}$$

Since we consider modeling trajectories of human-robot encounters and how humans deviate their trajectories, this fragment appears to be expressive enough: in particular, the probabilistic predicates encode desired spatial distributions of trajectories at certain discrete times; the *eventually* operator enforces the probabilistic predicate to hold at least once within a given interval; finally, the *conjunction* and *disjunction* operators encode combinations of events.

III. Problem formulation

We aim to find a probabilistic STL specification in the form of (5) that describes a set of m trajectories $\mathfrak{S}$ discretized by sampling into discrete-time signals $\sigma_i(t_0)\sigma_i(t_1)\sigma_i(t_2)\cdots$, $i \in \{1, \ldots, m\}$. We consider the setting where only positive data is available, i.e. data describing the desired trajectories, but no data violating the specification is present in the dataset. Formally:

Problem 1: Given a set of discrete-time signals $\mathfrak{S}$, learn a probabilistic STL formula ϕ, such that $\forall \sigma \in \mathfrak{S}$, $\sigma \models \phi$ and ϕ is *tight* enough, that is, ϕ is a representation of the set $\mathfrak{S}$ but not of a random or undesired set of trajectories.

IV. Methodology

The overall procedure is described in Algorithm 1 and consists of four steps:

a) First, for each time instant t_i, to find the number of n-dimensional normal distributions (in this case, 2-dimensional since we consider trajectories in 2D)

that cluster data points $\sigma_1(t_i), \sigma_2(t_i), \ldots, \sigma_m(t_i)$ into dense enough clusters (see below for details).

b) Second, to smooth the number of normal distributions over time. This operation enables the creation of windows of successive time steps with the same number of normal distributions that will allow for consistent generalizations, i.e. form a succession of *intervals* on which the *eventually* operators will be defined.

c) Third, in each of the previously defined intervals, to link the distributions of one time step t_i to the closest distribution in the next time step t_{i+1}. Therefore, we create what we refer to as "chains" of normal distributions over time that can later be generalized into one *meta* distribution describing the whole chain in the interval. Such *meta* distribution is surrounded by an *eventually* operator equipped with the related interval.

d) Fourth, to form the output specification as a disjunction of all relevant conjunctions of *meta* distributions over intervals that describe the input trajectories.

Below we describe the four procedures in detail.

a) Retrieving the number of normal distributions adequately clustering the trajectories at time t_i: The first step is to iterate over each time step $t_0, t_1, \ldots, t_H$ of the trajectories (where H denotes the length of the longest signal in the dataset $\mathfrak{S}$) to calculate how many dense clusters of data points can be formed (l.2–6). We want to retrieve the optimal number of normal distributions that describe the data at each time step t_i. To do so, all the data points $\mathfrak{S}_{t_i}$ of the signals at time t_i are iteratively retrieved (l.3). For each time step t_i, a mixture of Gaussians gmm of the form $\{\mathcal{N}_1(\mu_1, \Sigma_1), \ldots, \mathcal{N}_\beta(\mu_\beta, \Sigma_\beta)\}$ is retrieved from $\mathfrak{S}_{t_i}$, using statistical learning (l.4). The number of normal distributions composing gmm is a parameter β, being the maximum number of desired normal distributions describing the data points at any instant. Note here that classical machine learning implementations are available for the `gaussianMixture` function[1]. The normal distributions comprised in gmm are later clustered. The goal of the algorithm `hierarchicalClust` (l.5) is to provide the optimal number of clusters of normal distributions that can be formed within gmm, given a distance metric and a threshold α given as input to the algorithm. It consists of a hierarchical clustering-based approach that first computes a distance matrix storing the pairwise distances between normal distributions in gmm. The distance function used is the square root of the Jensen-Shannon distance between two normal distributions (4). The fact that $\sqrt{D_{JS}}$ is a metric is of utmost importance: in a hierarchical clustering fashion and in order to operate linkage of the distance matrix, it is required that the distance function used is a metric (in the mathematical sense, i.e. satisfying identity of indiscernibles, symmetry and triangle inequality). Linkage is performed, and clusters are formed so that the pairwise distance between elements within a cluster remains below the threshold α. `hierarchicalClust` then returns $|\mathcal{C}|$, the number of optimal clusters of data points. The next operation is to calculate $\mathcal{G}_{t_i}$, the mixture of $|\mathcal{C}|$ normal distributions given the data points $\mathfrak{S}_{t_i}$ (l.6).

[1] in our implementation, we used the eponymous function from the Python `scipy` library.

Algorithm 1: LEARNSTL($\mathfrak{S}$)

Input: α – max $\sqrt{D_{\text{JS}}}$ in a cluster of normal distr
β – max nb normal distributions clustering datapoints
θ – tightness factor
$H = \max_{\sigma \in \mathfrak{S}} |\sigma|$ – length of longest signal in $\mathfrak{S}$
Output: ϕ – STL formula of the form of (5).

```
// calculate nb of optimal distributions
```
1 $\mathcal{G} \leftarrow \emptyset$
2 **for** $i \in [0, H]$ **do**
3 $\quad \mathfrak{S}_{t_i} \leftarrow \{\sigma(t_i),\ \forall \sigma \in \mathfrak{S}\}$
4 $\quad gmm \leftarrow \texttt{gaussianMixture}(\mathfrak{S}_{t_i}, \beta)$
5 $\quad |\mathcal{C}| \leftarrow \texttt{hierarchicalClust}(gmm, \alpha)$
6 $\quad \mathcal{G}_{t_i} \leftarrow \texttt{gaussianMixture}(\mathfrak{S}_{t_i}, |\mathcal{C}|)$

```
// forming intervals
```
7 $\mathcal{G} \leftarrow \texttt{smoothNbDistr}(\mathcal{G})$

```
// forming meta distributions
```
8 $\Psi \leftarrow \emptyset$
9 **for** $k,\ I \leftarrow \texttt{groupBy}(\mathcal{G})$ **do**
10 $\quad \Psi_I \leftarrow \emptyset$
11 $\quad chains \leftarrow \texttt{linkDistributions}(\mathcal{G}, k, I)$
12 $\quad$ **for** $c \in chains$ **do**
13 $\quad\quad S_c \leftarrow \texttt{signals}(c)$
14 $\quad\quad \mu, \Sigma \leftarrow \texttt{gaussianMixture}(S_c, 1)$
15 $\quad\quad \chi \leftarrow \mathcal{N}(\mu, \Sigma)$
16 $\quad\quad \epsilon \leftarrow P_\theta(\{\mathbb{P}(\sigma(t_\tau) \mid \chi),\ \forall \tau \in I,\ \forall \sigma \in S_c\})$
17 $\quad\quad \Psi_I \leftarrow \Psi_I \cup \{\Diamond_I \chi^\epsilon\}$

```
// construction of the returned formula
```
18 $G \leftarrow \texttt{DAG}(\Psi)$
19 $G \leftarrow \texttt{prune}(G, \mathfrak{S})$
20 **return** $\bigvee_{path \in paths(G)} \bigwedge_{\Diamond_I \chi^\epsilon \in path} \Diamond_I \chi^\epsilon$

b) Forming intervals of a uniform number of distributions: After the mixture of normal distributions $\mathcal{G}$ is calculated for all time steps, we want to smooth the number of normal distributions in $\mathcal{G}$ over time. To illustrate this, take the following example: if $\mathcal{G}_{t_{10}..t_{14}}$ and $\mathcal{G}_{t_{16}..t_{20}}$ are composed of 3 distributions, but $\mathcal{G}_{t_{15}}$ contains 4, one may want to smooth this value in order to, later on, generalize over the *interval* [10..20]. We call the function `smoothNbDistr` (l.7) that runs a Savitzky–Golay filter on the number of normal distributions over time steps to perform smoothing. In case at a given time t_i the number of normal distributions in $\mathcal{G}_{t_i}$ is changed, the filter automatically updates $\mathcal{G}_{t_i}$ so that it will contain the desired number of distributions (we run the `gaussianMixture` function on the data points $\sigma_1(t_i), \sigma_2(t_i), \ldots, \sigma_m(t_i)$ at time t_i, with the smoothed number of distributions). After this operation, we form *intervals* of successive time steps where the trajectories are clustered in the same number of distributions.

c) Forming meta distributions in intervals: The next step is, for each interval I (provided by the `groupBy` function, – l.9), to form meta distributions covering the time steps in the interval. First, we create k empty chains, k being the number of normal distributions in the interval. The key insight is to perform a 1-1 mapping of the closest normal distributions from one time step to the next in I. The linkage is done such that a given distribution in $\mathcal{G}_{t_i}$ at time $t_i \in I$ is linked to the closest distribution in $\mathcal{G}_{t_{i+1}}$, using $\sqrt{D_{JS}}$ as a distance (l.11). Now, for each chain, we will calculate the related probabilistic predicate (l.12). We first retrieve all the data points S_c corresponding to trajectories being classified by any of the chain distributions in $\mathcal{G}$ (l.13). We then infer a meta distribution χ modeling all these data points S_c (l.14–15). The probability threshold ϵ is set to the θ^{th} percentile (noted P_θ) of the set of probabilities for each data point in S_c to fit χ (l.16). Note that the parameter θ acts here as a tightness factor, since the closer it will be to 1, the greater the probability of data points to satisfy χ will be; conversely, a θ closer to 99 will impose a high bound on the minimum probability of satisfaction of χ. Then, we add the probabilistic predicate χ^ϵ surrounded by an eventually operator with interval I to the set of STL formulae at interval I, Ψ_I (l.17), and we repeat this for the rest of the chains.

d) Construction of the returned STL formula: From the set of STL formulae of the form $\Diamond_I \chi^\epsilon$ in Ψ, we construct a DAG such that edges connect components with adjacent intervals (l.18). Since it is possible that some of the signals in $\mathfrak{S}$ do not use all combinations of adjacent STL formulae, we prune the graph G such that unnecessary edges w.r.t. the dataset $\mathfrak{S}$ are removed (l.19). The returned formula is the disjunction of all paths in G, where a path is defined as the conjunction of STL formulae of the form $\Diamond_I \chi^\epsilon$ acting as vertices of the path (l.20). About our dataset of trajectories of human-robot encounters, the disjunctions represent as many possible combinations of choices (speed up, slow down, deviate to the left or the right) taken by the humans to avoid the robot. The conjunctions represent the succession of probabilistic distributions of their spatial location at different time steps.

V. Experiments

We implemented and tested our learning algorithm for modeling trajectories in Python 3.8[2]. We ran our experiments on an Intel i7-8665U CPU and 32GB RAM.

A. Data collection

Participants played a game where they had to reach a target zone while navigating around a robot moving towards them. The game was developed in Unity version 2019.4.16f1 and exported to WebGL. Screenshots from the game are shown in Fig. 1. We recruited 50 participants (16 female, median age 33 years old) on the online platform Amazon Mechanical Turk (AMT), due to COVID-19 restrictions at the time. When they accepted to take part in the study, they were redirected to a webpage that contained a digital consent form. They completed 20 trials of the game. Each trial proceeded as follows: participants walked towards a goal, represented by a red flag, by means of an avatar (Fig. 1). Participants could use 4 types of actions triggered by the 4 keyboard arrows: $\uparrow$ to start moving and accelerate the speed of the avatar; $\downarrow$ to decelerate the speed; $\leftarrow$ and $\rightarrow$ to deviate the avatar's trajectory to the left or to the right. In order to reach the goal, they had to navigate around a virtual robot. Specifically, we used a 3D model of Softbank Robotics's Pepper[3], a widely-used social robot [22]. Since the human and robot avatars start from a large distance from each other, we added a zoomed-in view of the robot so that participants had enough time to see the robot before starting any movement. After the last trial, participants answered some questions about their gender, age, country of origin and experience with robots, and were given the unique survey code needed to be paid in AMT. The whole experiment lasted approximately 10 minutes.

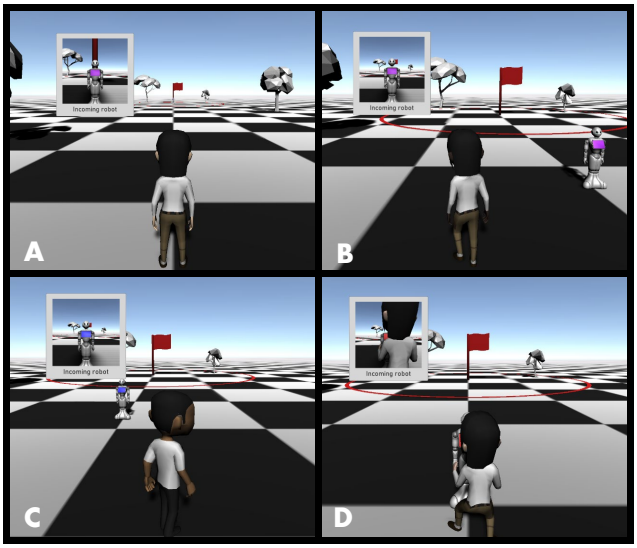


Fig. 1: Screenshots from the experiment: robot's and participant's avatars walking towards each other. A: the participant's starting point – B: participant deviating to the left – C: participant deviating to the right – D: participant and robot colliding.

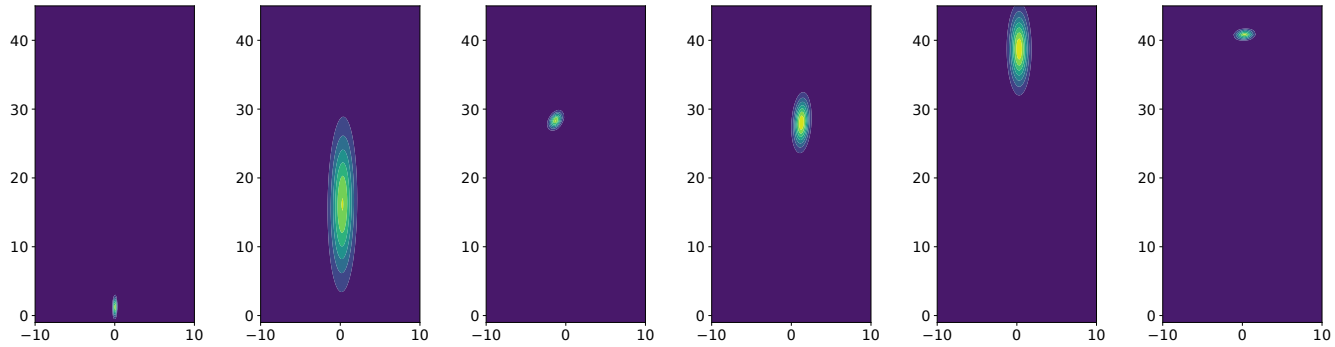
B. Results

We collected a total of 1,000 trajectories recorded with a frequency of 10 Hz, i.e. time steps of 100ms. The initial x-position of both the participants and the robot was set to 0. The initial y-position of the participant was 0, and the one of the robot 43. As pre-processing, we filtered out:

1) the trajectories where participants deviated too much to the left or to the right. The average of the absolute value of x positions was 1.47, with a standard deviation of 1.16. We filtered out trajectories containing at least one data point with absolute value of x greater than $1.47 + 2 \times 1.16 = 3.79$.

[2]Software and data available at https://github.com/allinard/stl_hrencounters

[3]https://www.softbankrobotics.com/emea/en/pepper



(a) χ_1 (b) χ_2 (c) χ_3 (d) χ_4 (e) χ_5 (f) χ_6

Fig. 2: Spatial representations of the 6 probabilistic predicates in the learnt specification.

2) the trajectories where participants completed the task too slowly. The minimum completion was done in 72 time steps, the maximum in 248 time steps, with an average of 81 and a standard deviation of 17.32. We removed trajectories completed with a number of time steps greater than $\lfloor 81 + 2 \times 17.32 \rfloor = 115$.
3) the trajectories where participants collided with the robot, which represented 127 trajectories. Note that we used these colliding trajectories later on for evaluation purposes, which we refer to as $\mathfrak{S}_-$.

As a result, we used 814 trajectories, which we refer to as $\mathfrak{S}_+$.

a) Learned specification: Using Algorithm 1, we learned the following specification. We empirically set parameters $\alpha = 1.6$; $\beta = 5$; $\theta = 27$; $w = 9$; $p = 1$. In Fig. 2 we show a spatial representation of the different probabilistic predicates composing the STL formula.

$$\phi = \big((\phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_5 \wedge \phi_6) \\ \vee (\phi_1 \wedge \phi_2 \wedge \phi_4 \wedge \phi_5 \wedge \phi_6) \big) \tag{6}$$

$$\phi_1 = \Diamond_{[1,10]} \chi_1^{0.46}, \quad \mu_1 = [0.019\ 1.226], \quad \Sigma_1 = \begin{bmatrix} 0.022 & 0.01 \\ 0.01 & 0.769 \end{bmatrix}$$
$$\phi_2 = \Diamond_{[11,50]} \chi_2^{0.26}, \quad \mu_2 = [0.266\ 16.154], \quad \Sigma_2 = \begin{bmatrix} 0.959 & 0.401 \\ 0.401 & 45.024 \end{bmatrix}$$
$$\phi_3 = \Diamond_{[51,55]} \chi_3^{0.301}, \quad \mu_3 = [-1.28\ 28.359], \quad \Sigma_3 = \begin{bmatrix} 0.284 & 0.153 \\ 0.153 & 0.548 \end{bmatrix}$$
$$\phi_4 = \Diamond_{[51,55]} \chi_4^{0.557}, \quad \mu_4 = [1.273\ 28.036], \quad \Sigma_4 = \begin{bmatrix} 0.39 & 0.221 \\ 0.221 & 4.756 \end{bmatrix}$$
$$\phi_5 = \Diamond_{[56,103]} \chi_5^{0.29}, \quad \mu_5 = [0.284\ 38.727], \quad \Sigma_5 = \begin{bmatrix} 0.588 & -0.002 \\ -0.002 & 11.038 \end{bmatrix}$$
$$\phi_6 = \Diamond_{[104,114]} \chi_6^{0.385}, \quad \mu_6 = [0.277\ 40.827], \quad \Sigma_6 = \begin{bmatrix} 0.53 & 0.039 \\ 0.039 & 0.189 \end{bmatrix}$$

We can see from the retrieved specification the ability of our algorithm to learn disjunctions of events (in our case, disjunctions of clusters of trajectories). Indeed, concerning our case study, we can see that the right/left trajectory deviations are correctly identified by 2 of the probabilistic predicates (ϕ_3–Fig. 2c and ϕ_4–Fig. 2d). Besides, combinations of different user choices and preferences are established by the *disjunctions* of the returned specification.

b) Evaluation: Our algorithm returned the learned specification in (6) in about 20s. We state the evaluation

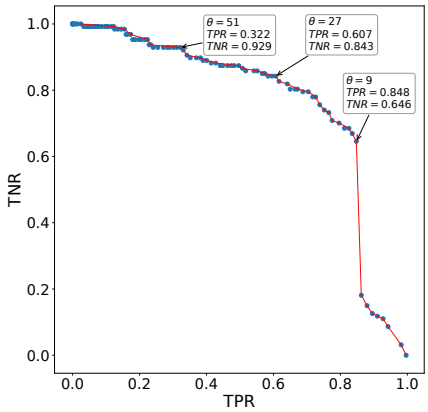


Fig. 3: Pareto front of TPR and TNR, for different θ's.

results in terms of True Positive Rate (TPR – the proportion of signals in $\mathfrak{S}_+$ that satisfy the learned specification) and True Negative Rate (TNR – the proportion of *colliding* trajectories in $\mathfrak{S}_-$ that violate the learned specification). The goal is to achieve both good TPR and TNR. In Fig. 3, we show trade-offs between the obtained TPR and TNR for different values of the tightness parameter θ, and show the pareto front in red. As an example, we achieved a good compromise between TPR and TNR with $\theta = 27$, that is, TPR $= 0.607$ and TNR $= 0.843$. More generally, in order to evaluate our method as a classifier of trajectories, we ran a cross-validation of 10 folds on our dataset. We achieved an average TPR of 0.563 (variance of 0.003) and an average TNR of 0.815 (variance of 0.003).

C. Discussion

The interpretation of our results is the following: the learned specification, despite rejecting some good trajectories, is tight enough to refuse satisfaction of most of the bad trajectories (i.e., trajectories that result in human-robot collision). These are encouraging results. Since we want to use the specification on robot controllers, we can be confident that the generated trajectories given this specification will follow the behavior described by humans in these environments. Indeed, by construction of Algorithm 1, the tightness

parameter θ directly influenced how the parameter ϵ in the formulae of the form $\diamondsuit_I \chi^\epsilon$ is set. As a consequence, a tighter θ parameter induces that fewer data points in the space are capable of satisfying the corresponding χ^ϵ, whereas a low tightness induces high permissiveness and the capturing of false positives. Empirically, however, tightness degrees can be explored, and local optimization of the θ parameter is recommended in order to achieve the best balanced accuracy, TNR or TPR according to the needs. Note also that we chose to use the *eventually* operator instead of the *always* operator in our fragment (5), since the *always* operator is too restrictive, by enforcing a given condition to hold through an entire interval. Nonetheless, we argue that our technique applies to any scenario where valid trajectories can be clustered into user choices describing multiple classes of trajectories. We aim to apply it to other case studies in the context of social navigation. The fact that our method is parameterizable is an asset to model different contexts, depending on how safety critical the application is (one may require satisfaction of probabilistic predicates to be more restrictive than other scenarios).

D. Future work

A possible modification of our algorithm would be to weight the disjunction operators by how many trajectories in the dataset use a given element in the disjunction. This is an interesting approach for synthesizing trajectories. Our next step is to achieve control synthesis based on the learnt STL formula with probabilistic predicates, resulting in the production of trajectories satisfying the STL specification. Typically, given an STL formula (e.g. the learnt specification), mixed integer semi-definite programming approaches can be employed, such as the one proposed in [14]. Our goal is to generate trajectories based on the learned specification that we would show to users in simulations. We hope to reintegrate generated trajectories in a user study, refine the learned specification based on user experience, and eventually converge to specifications of trajectories for social navigation of robots among humans.

VI. Conclusion

We developed a technique to infer probabilistic STL formulae formalizing human-robot encounters. We collected human data in a simulation that places users in a virtual environment with a robot, where the user needs to reach a target zone by avoiding the robot. From the trajectories recorded from the study, we learned a specification of social navigation. Out approach learned a formula tight enough to reject most of the colliding trajectories. In the future, we will deploy robot's controllers embedding the social navigation specification, test its integration, and refine the specification thanks to feedback on user experience.

References

[1] L. Sun, Z. Yan, S. M. Mellado, M. Hanheide, and T. Duckett, "3dof pedestrian trajectory prediction learned from long-term autonomous mobile robot deployment data," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 5942–5948.

[2] A. Robicquet, A. Sadeghian, A. Alahi, and S. Savarese, "Learning social etiquette: Human trajectory understanding in crowded scenes," in *European conference on computer vision*. Springer, 2016, pp. 549–565.

[3] A. Rudenko, L. Palmieri, M. Herman, K. M. Kitani, D. M. Gavrila, and K. O. Arras, "Human motion trajectory prediction: A survey," *The International Journal of Robotics Research*, vol. 39, no. 8, pp. 895–935, 2020.

[4] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi, "Social gan: Socially acceptable trajectories with generative adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2255–2264.

[5] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for mobile robots," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2005, pp. 2020–2025.

[6] D. Porfirio, A. Sauppé, A. Albarghouthi, and B. Mutlu, "Authoring and verifying human-robot interactions," in *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, 2018, pp. 75–86.

[7] D. Porfirio, A. Sauppe, A. Albarghouthi, and B. Mutlu, "Transforming robot programs based on social context," in *Proceedings of the 2020 CHI conference on human factors in computing systems*, 2020.

[8] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 2004, pp. 152–166.

[9] E. Asarin, A. Donzé, O. Maler, and D. Nickovic, "Parametric identification of temporal properties," in *International Conference on Runtime Verification*. Springer, 2011, pp. 147–160.

[10] G. Bombara, C.-I. Vasile, F. Penedo, H. Yasuoka, and C. Belta, "A decision tree approach to data classification using signal temporal logic," in *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*. ACM, 2016, pp. 1–10.

[11] L. Nenzi, S. Silvetti, E. Bartocci, and L. Bortolussi, "A robust genetic algorithm for learning temporal specifications from data," in *International Conference on Quantitative Evaluation of Systems*. Springer, 2018, pp. 323–338.

[12] A. Jones, Z. Kong, and C. Belta, "Anomaly detection in cyber-physical systems: A formal methods approach," in *53rd Conference on Decision and Control (CDC)*. IEEE, 2014, pp. 848–853.

[13] A. Linard and J. Tumova, "Active learning of signal temporal logic specifications," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2020, pp. 779–785.

[14] D. Sadigh and A. Kapoor, "Safe control under uncertainty with probabilistic signal temporal logic," in *Robotics: Science and Systems (RSS '16)*, 06 2016.

[15] M. Tiger and F. Heintz, "Incremental reasoning in probabilistic signal temporal logic," *International Journal of Approximate Reasoning*, vol. 119, pp. 325–352, 2020.

[16] I. Torre, A. Linard, A. Steen, J. Tumova, and I. Leite, "Should robots chicken? how anthropomorphism and perceived autonomy influence trajectories in a game-theoretic problem." *To appear in Proceeding of the 16th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2021.

[17] H. Khambhaita and R. Alami, "Viewing robot navigation in human environment as a cooperative activity," in *Robotics Research*. Springer, 2020, pp. 285–300.

[18] P.-T. Singamaneni and R. Alami, "Hateb-2: Reactive planning and decision making in human-robot co-navigation," in *International Conference on Robot & Human Interactive Communication, 2020*, 2020.

[19] M. Díaz-Boladeras, D. Paillacho, C. Angulo, O. Torres, J. González-Diéguez, and J. Albo-Canals, "Evaluating group-robot interaction in crowded public spaces: A week-long exploratory study in the wild with a humanoid robot guiding visitors through a science museum," *International Journal of Humanoid Robotics*, vol. 12, no. 04, 2015.

[20] A. K. Ball, D. C. Rye, D. Silvera-Tawil, and M. Velonaki, "How should a robot approach two people?" *Journal of Human-Robot Interaction*, vol. 6, no. 3, pp. 71–91, 2017.

[21] D. M. Endres and J. E. Schindelin, "A new metric for probability distributions," *IEEE Transactions on Information theory*, vol. 49, no. 7, pp. 1858–1860, 2003.

[22] M. E. Foster, R. Alami, O. Gestranius, O. Lemon, M. Niemelä, J.-M. Odobez, and A. K. Pandey, "The mummer project: Engaging human-robot interaction in real-world public spaces," in *International Conference on Social Robotics*. Springer, 2016, pp. 753–763.