2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2022)
October 23-27, 2022, Kyoto, Japan

Inference of Multi-Class STL Specifications for Multi-Label Human-Robot Encounters

Alexis Linard, Ilaria Torre, Iolanda Leite and Jana Tumova

Abstract— **This paper is interested in formalizing human trajectories in human-robot encounters. Inspired by robot navigation tasks in human-crowded environments, we consider the case where a human and a robot walk towards each other, and where humans have to avoid colliding with the incoming robot. Further, humans may describe different behaviors, ranging from being in a hurry/minimizing completion time to maximizing safety. We propose a decision tree-based algorithm to extract STL formulae from multi-label data. Our inference algorithm learns STL specifications from data containing multiple classes, where instances can be labelled by one or many classes. We base our evaluation on a dataset of trajectories collected through an online study reproducing human-robot encounters.**

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

I. Introduction

Social navigation of robots among humans is a research topic concentrating much interest due to diverse challenges, such as the prediction of human behavior in crowded social environments to enable efficient robot navigation, or the identification of "navigation styles" for a mobile robot in its surrounding environment [1]. In robot motion planning and control, desired behavior specifications and preferences can be expressed in Signal Temporal Logic (STL) [2], a formalism that can express system properties that include bounds on time and values of system parameters. STL defines specifications over multi-dimensional continuous signals and can, e.g., be used to depict properties of trajectories. Since STL allows rich expressiveness and resemblance to natural language [3], it has recently been used in human-robot interaction applications [4], [5], and human-robot encounters. Indeed, there is a growing need to depict human-robot interactions through formal and rigorous techniques to provide guarantees on the functioning of robotic systems, as well as their interactions with humans and perceived safety [6].

In this paper, we conducted a similar study like the one of [7], but where human users describe several behaviors while avoiding a robot driving in a collision course: humans can be in a hurry (i.e. having to navigate around the robot as fast as possible), maximizing safety (i.e. having to avoid a collision at all costs), or with no particular motivation (i.e. having to navigate around the robot in a reasonable amount of time but without safety constraints). However, as human behavior is hardly predictable and can depict high variability between two individuals, the "*maximizing safety*" recorded trajectories of ones can also be comparable to the "*no special motivation*" trajectories of others (both in terms of spatial and temporal distributions), as different people might have overlapping definitions of what is "*maximizing safety*" and just "*taking a normal walk*". Therefore, it can be the case that trajectories collected during this experiment are at the same time labelled by one and the other human motivations.

To answer the need of specifying robot's behaviors through formal methods, temporal logic inference typically extracts specifications from observed behaviour, classified as either satisfying or violating the target specification. STL inference methods learning both the structure and the parameters of the specifications are classically supervised learning approaches based on decision trees [8]. However, only one specific setting has been considered so far: data is separated into positive (i.e. satisfying the target specification) and negative (i.e. violating the target specification) trajectories. Our research question lies in the following: what if the data is not only labelled by two classes (that is, positive vs. negative), but by several classes instead (that is, class A, class B, class C, etc.), and that trajectories can belong to one or more classes (e.g., trajectory σ satisfies class B and C)?

The contributions of this paper are multi-fold: first, we propose a multi-class approach for the inference of STL formulae. Moreover, we consider the case where data can be labelled with multiple classes. Inspired by the learning approach of Bombara et al. [8], we developed a decision tree-based method that tries to best separate the (multi)labelled signals according to evaluation metrics accounting for the multi-label classification performance, rather than the information gain usually used in binary classification tasks. The second contribution of this paper is an original on-the-fly pruning technique of the decision tree. At each iteration of the construction of the tree's nodes, the STL formulae constituting the nodes may have overlapping semantics. To avoid overlapping STL formulae, we enable the creation of nodes where the intersection of a new node's STL formula with its ancestor nodes' formulae is empty. Finally, our last contribution is the application of STL inference to social navigation: we provide a method learning specifications of human-robot encounters, taking into account several human motivations in the same model. In the future, applications of our learned models to STL-based robot motion planning will appropriately include human preferences, internal motivations and states, as well as perceived safety.

The authors are with the KTH Royal Institute of Technology, SE-100 44, Stockholm, Sweden, with the Division of Robotics, Perception and Learning and are also affiliated with Digital Futures. This work was supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, the Swedish Research Council (VR) (project no. 2017-05102), and partially supported by Sweden's Innovation Agency (Vinnova) through the TECoSA project.
{linard,ilariat,iolanda,tumova}@kth.se

II. Related work

The use of temporal logics as a tool to specify robotics tasks in an interpretable way has been recently explored in the field of human-robot interaction. For instance, Porfirio et al. use temporal logics to express social norms and social robot behaviors in human-robot interaction [4], [9], to specify, verify, and improve the communication modalities of humans with robots. In a study inspired by the "Game of Chicken" [7], human users in a simulated environment had to navigate a space shared with a robot, moving in the direction of the human and along the same path. To avoid a collision, users had to deviate from their trajectory. The goal was to learn probabilistic STL specifications [5] from these human-robot trajectories to model distributions of human behaviors when facing the incoming robot.

STL inference techniques that span structure and parameter learning comprise techniques building the STL formula and mining its parameters. For unsupervised learning, existing works relate the temporal inference process to a one-class SVM based optimization function [10], [11], grid-based projection of the signals to infer clusters of signals [12], or multi-dimensional binary search [13]. Concerning supervised learning, Nenzi et al. [14] employ an evolutionary algorithm to learn an STL formula from positive and negative signals. Linard et al. [15] developed an active learning approach when no data is available beforehand. In a different fashion, the authors of [8] proposed both offline and online decision tree-based methods to learn an STL formula, where the decision tree represents the STL formula. It consists of nodes containing an STL sub-formula locally separating data, where the formula is chosen among a set of STL *primitives*, i.e. STL formulae enforcing spatio-temporal constraints on the trajectories, where spatial and temporal parameters are optimized to classify instances at a given node appropriately. The specification for a given class is then obtained by recursively parsing the tree's nodes leading to a leaf labelled by the class. This approach catches our attention: it can be modified to consider other optimization criteria, such as multi-label related criteria, and the formalism of single label leaves can be extended to include multi-labelled leaves. Recently, some extensions of [8] include boosted decision tree learning for STL inference [16], or the combination of decision trees and neural networks [17], but to the best of our knowledge, none looked into multi-class problems.

III. Preliminaries

Let $\mathbb{R}$ and $\mathbb{N}$ be the set of real and natural numbers including zero, respectively. We use a 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, finite, discrete-time signal σ is defined as a sequence $\sigma : \sigma(t_0)\sigma(t_1)\sigma(t_2)\ldots$, where $\sigma(t_i) \in \mathbb{R}^n$ is the value of signal σ at time $t_i \in \mathbb{N}$. The j-th component of signal value $\sigma(t_i)$ is denoted by $\sigma(t_i)^j$, where $j \in [1, n]$. The set of all signals with values taken from $\mathbb{R}^n$ is denoted by Σ. The syntax of STL is defined as:

$$\phi ::= \top \mid \pi \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \mathcal{U}_I \phi_2 \tag{1}$$

where $\top$ is the Boolean *True* constant, π a predicate over $\mathbb{R}^n$ in the form of $f(x) \sim \mu$, $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $\mu \in \mathbb{R}$, and $\sim \in \{\leq, >\}$; $\neg$ and $\wedge$ are the Boolean operators for negation and conjunction, respectively; and $\mathcal{U}_I$ is the temporal operator *until* over bounded interval I. Other Boolean operations are defined using the conjunction and negation operators to enable the full expression of propositional logic. Additional temporal operators *eventually* and *globally* are defined as $\Diamond_I \phi \equiv \top \mathcal{U}_I \phi$ and $\Box_I \phi \equiv \neg \Diamond_I \neg \phi$, respectively. Further, the semantics of STL are defined as:

$$\begin{array}{lcl}
\sigma(t_i) \models \pi & \Leftrightarrow & f(\sigma(t_i)) \sim \mu \\
\sigma(t_i) \models \neg\phi & \Leftrightarrow & \neg(\sigma(t_i) \models \phi) \\
\sigma(t_i) \models \phi_1 \wedge \phi_2 & \Leftrightarrow & (\sigma(t_i) \models \phi_1) \wedge (\sigma(t_i) \models \phi_2) \\
\sigma(t_i) \models \phi_1 \mathcal{U}_I \phi_2 & \Leftrightarrow & \exists i' \in t + I \text{ s.t. } \sigma(t_{i'}) \models \phi_2 \\
 & & \wedge \forall i'' \in [t, t'],\ \sigma(t_{i''}) \models \phi_1
\end{array}$$

Parametric Signal Temporal Logic (PSTL) [18] is an extension of STL where formulae (time bounds and predicates parameters) are parameterized. We refer to *primitives* as PSTL formulae with parameters Θ. Further, the parameter space of Θ is a user-given value.

IV. Problem formulation

Our work is motivated by the need of specifying human-robot encounters in a rigorous way. We aim to find an STL specification that describes a set of 2-dimensional trajectories, where multiple classes of human behaviors can label each trajectory. In this paper, we will consider a 2-dimensional space (with dimensions x and y) and related *primitives*, that describe bounded 2D spatial properties that have to hold within a given interval:

$$p_1 = \Diamond_{[\tau_1,\tau_2]}(\alpha \leq x \leq \beta \wedge \gamma \leq y \leq \delta) \tag{2a}$$

$$p_2 = \Box_{[\tau_1,\tau_2]}(\alpha \leq x \leq \beta \wedge \gamma \leq y \leq \delta) \tag{2b}$$

$$p_3 = \Box_{[\tau_1,\tau_2]}\neg(\alpha \leq x \leq \beta \wedge \gamma \leq y \leq \delta) \tag{2c}$$

with $\Theta = \{\tau_1, \tau_2, \alpha, \beta, \gamma, \delta\}$ and $\tau_1, \tau_2, \alpha, \beta, \gamma, \delta \in \mathbb{R}$. Intuitively, primitive p_1 of (2a) means that the x and y dimensions of the signal have to be comprised between $[\alpha, \beta]$ and $[\gamma, \delta]$ respectively, *at least once* during time interval $[\tau_1, \tau_2]$. Primitive p_2 of (2b) means that the x and y dimensions of the signal have to be comprised between $[\alpha, \beta]$ and $[\gamma, \delta]$ respectively, *during the entire* time interval $[\tau_1, \tau_2]$. Finally, primitive p_3 of (2c) means that the x and y dimensions of the signal *can never* be comprised between $[\alpha, \beta]$ and $[\gamma, \delta]$ respectively, *during the entire* time interval $[\tau_1, \tau_2]$. The choice of these primitives is motivated by our case study: we seek easily interpretable formulae, representing semantic *satisfiability zones* of trajectories in the 2D space. Note that the resulting STL formulae (with set parameters) are convex. Hence, resulting STL formulae can be seen a 3D convex polytopes (and more exactly 3D cuboids – with 2 dimensions for the spatial dimensions plus 1 for time). In Sect. V-B, we will define geometrical operations on such polytopes, standing for the intersection and difference of 2 STL formulae.

Algorithm 1: MULTICLASS_STL_LEARN($\mathscr{S}$,φ^{path})

```
    Input: $\mathscr{S}$ – set of labelled signals
    $\varphi^{path}$ – set of STL formulae on the path to curr. node
    $\mathcal{P}$ – set of PSTL primitives
    $stop$ – stopping criterion
    Output: $\phi$ – an STL formula
1 if $stop$ then return $leaf(\arg\max_{(\sigma,C)\in\mathscr{S}} C)$
2 $\phi^* = \arg\max_{p^*\in\mathcal{P},\theta^*\in\Theta} G(\mathscr{S}, p^*(\theta^*))$
3 $\mathscr{S}_+ \leftarrow \{(\sigma,C)\in\mathscr{S} \mid \sigma\models\phi^*\}$
4 $\mathscr{S}_- \leftarrow \{(\sigma,C)\in\mathscr{S} \mid \sigma\not\models\phi^*\}$
5 $node.right \leftarrow multiclass\_STL\_learn(\mathscr{S}_+, \varphi^{path}\cup\{\phi^*\})$
6 $node.left \leftarrow multiclass\_STL\_learn(\mathscr{S}_-, \varphi^{path}\cup\{\phi^*\})$
7 return $node$
```

In the remainder of the paper, we will consider the set of parametric primitives $\mathcal{P} = \{p_1, p_2, p_3\}$ as defined in (2a), (2b) and (2c). Also, we will consider formulae belonging to the following fragment of STL:

$$\phi ::= \top \mid p \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \tag{3}$$

where p is an STL formula of the form of one of the STL formulae in $\mathcal{P}$ with defined parameters. We claim that this fragment is rich enough to capture specifications of trajectories, by combining conjunctions and disjunctions of satisfiability zones over time.

We aim to find an STL specification in the form of (3) that describes a set of 2-dimensional trajectories $\mathscr{S}$ discretized by sampling into discrete-time signals $\sigma_i(t_0)\sigma_i(t_1)\sigma_i(t_2)\cdots$, $i \in \{1,\ldots,|\mathscr{S}|\}$. We consider the setting where trajectories can be labelled by 1 to $|\mathcal{C}|$ classes where $\mathcal{C} = \{c_1, c_2 \ldots\}$, $|\mathcal{C}| \geq 2$, and where $\mathcal{C}$ denotes the classes a trajectory can belong to. We denote by $C \subseteq \mathcal{C}$ the labels of a signal.

Problem 1: Given a set of *labelled* discrete-time signals $\mathscr{S}$, learn $|\mathcal{C}|$ STL formulae ϕ_c, $\forall c \in \mathcal{C}$ such that:

$$\forall(\sigma,C)\in\mathscr{S},\ \forall c\in C,\ \sigma\models\phi_c \wedge \forall c'\in\mathcal{C}\backslash C, \sigma\not\models\phi_{c'} \tag{4}$$

In other words, we want to learn as many sub-formulae as classes, and that each of the trajectories in the dataset satisfies the class(es) it belongs to.

V. METHOD

We propose two algorithms to identify STL formulae describing multi-labelled signals in a multi-class context. The first algorithm is a modification of state-of-the-art decision tree-based algorithms that account for optimization criteria reflecting the multi-label problem (Section V-A). The second is an extension of the previous that renders more concise STL formulae to increase the readability and explainability of the returned specifications (Section V-B).

A. Learning Multiclass STL

Algorithm 1 follows parts of [8] and recursively builds a tree, given a set of labelled signals $\mathscr{S}$, where each node is labelled by an STL formula locally separating the data. It takes as input the set of PSTL primitives $\mathcal{P}$ from which an STL formula is found to classify the data at each node, and stopping criteria (e.g., a maximum tree depth, maximum runtime, purity inside the set of labelled signals $\mathscr{S}$, etc.). Unless the stopping criteria are met, the optimal combination of parameters θ^* for each primitive $p \in \mathcal{P}$ is found (line 2) w.r.t. an optimization criterion G, which is expressed as the gain expressed in terms on multi-label entropy E:

$$G(\mathscr{S},\phi) = E(\mathscr{S}) - (E(\mathscr{S}_+)\frac{|\mathscr{S}_+|}{|\mathscr{S}|} + E(\mathscr{S}_-)\frac{|\mathscr{S}_-|}{|\mathscr{S}|})$$

$$E(\mathscr{S}) = -\sum_{c\in\mathcal{C}} P(c)\log(P(c)) + (1 - P(c)\log(1-P(c)))$$

where $P(c)$ refers to the frequency of class c in $\mathscr{S}$, $\mathscr{S}_+ = \{\sigma \in \mathscr{S} \mid \sigma \models \phi\}$ and $\mathscr{S}_- = \{\sigma \in \mathscr{S} \mid \sigma \not\models \phi\}$. The optimization problem is solved using particle swarm optimization [19], [8], as a successful technique to search the combination of PSTL primitives' parameters rendering the highest gain, especially considering the high dimensionality of our problem (6 variables for the primitives of (2a), (2b) and (2c)). The choice among all optimized primitives is given to the primitive with parameters that provide the highest gain. The node is then labelled by this candidate STL formula ϕ^*. The next step is to separate the signals in $\mathscr{S}$ according to the candidate STL formula ϕ^*, between signals $\mathscr{S}_+$ satisfying ϕ^* (line 3) and signals $\mathscr{S}_-$ violating ϕ^* (line 4). The algorithm is recursively called on $\mathscr{S}_+$ to build the successor right nodes of the current node (line 5) as well as on $\mathscr{S}_-$ to build the successor left nodes (line 6).

Note that the output decision tree can be easily translated into an STL formula [8, Algorithm 2].

Algorithm 2: MULTICLASS_STLDIFF($\mathscr{S}$, φ^{path})

```
    Input: $\mathscr{S}$ – set of labelled signals
    $\varphi^{path}$ – set of STL formulae on the path to curr. node
    $\mathcal{P}$ – set of PSTL primitives
    $stop$ – stopping criterion
    Output: $\phi$ – an STL formula
1  if $stop$ then return $leaf(\arg\max_{(\sigma,C)\in\mathscr{S}} C)$
2  $\phi' = \arg\max_{p^*\in\mathcal{P},\theta^*\in\Theta} G(\mathscr{S}, p^*(\theta^*))$
   // calculates $\phi'$ minus all elements in $\varphi^{path}$
3  $diff \leftarrow \{\phi'\}$
4  for $\phi_p \in \varphi^{path}$ do
5      $\Delta \leftarrow \emptyset$
6      for $\phi_{diff} \in diff$ do $\Delta \leftarrow \Delta \cup \{\phi_{diff}\backslash\phi_p\}$
7      $diff \leftarrow \Delta$
8  $\phi^* = \arg\max_{\phi_d\in diff} G(\mathscr{S}, \phi_d)$
9  $\mathscr{S}_+ \leftarrow \{(\sigma,C)\in\mathscr{S} \mid \sigma\models\phi^*\}$
10 $\mathscr{S}_- \leftarrow \{(\sigma,C)\in\mathscr{S} \mid \sigma\not\models\phi^*\}$
11 $node.right \leftarrow multiclass\_STLDiff(\mathscr{S}_+, \varphi^{path}\cup\{\phi^*\})$
12 $node.left \leftarrow multiclass\_STLDiff(\mathscr{S}_-, \varphi^{path}\cup\{\phi^*\})$
13 return $node$
```

B. STL-difference method

In this section, we explain how we extend Algorithm 1 with on-the-fly pruning of the node's STL formulae in order to render concise STL formulae with non-overlapping semantics. The algorithm as depicted by Algorithm 2 follows a similar structure except for the optimization step

Algorithm 3: $\phi \cap \phi'$

```
Input: $\phi$, $\phi'$ – STL formulae of the form of (2a), (2b) or (2c), with set parameters.
Pre-condition: $\phi$ and $\phi'$ both of the same form.
Output: $\phi_\cap = \phi \cap \phi'$ – the intersection of $\phi$ and $\phi'$.
1  $a_\cap \leftarrow \max(a, a')$ ; $b_\cap \leftarrow \min(b, b')$
2  $c_\cap \leftarrow \max(c, c')$ ; $d_\cap \leftarrow \min(d, d')$
3  $t_{1\cap} \leftarrow \max(t_1, t'_1)$ ; $t_{2\cap} \leftarrow \min(t_2, t'_2)$
   // if $\phi$ and $\phi'$ overlap
4  if $a_\cap < b_\cap \wedge c_\cap < d_\cap \wedge t_{1\cap} < t_{2\cap}$ then
5      if $\phi$ and $\phi'$ of the form of (2a) then
6          return $\diamond_{[t_{1\cap}, t_{2\cap}]}(a_\cap \leq x \leq b_\cap \wedge c_\cap \leq y \leq d_\cap)$
7      if $\phi$ and $\phi'$ of the form of (2b) then
8          return $\square_{[t_{1\cap}, t_{2\cap}]}(a_\cap \leq x \leq b_\cap \wedge c_\cap \leq y \leq d_\cap)$
9      if $\phi$ and $\phi'$ of the form of (2c) then
10         return $\square_{[t_{1\cap}, t_{2\cap}]}\neg(a_\cap \leq x \leq b_\cap \wedge c_\cap \leq y \leq d_\cap)$
11 return $\emptyset$
```

of the PSTL primitives. We start by finding the optimal combination of parameters θ^* for each primitive $p \in \mathcal{P}$ according to optimization criterion G (line 2). We gather ϕ' as the best candidate primitive with values. The following is unprecedented and is one of the contributions of this paper.

Indeed, candidate ϕ' does not account for possible semantic overlappings between ϕ' and predecessor STL formulae in φ^{path} already discovered in the decision tree. Take, as an example, two STL formulae $\varphi' = \square_{[15,30]}(0 \leq x \leq 2)$ and $\varphi_p = \square_{[10,20]}(1 \leq x \leq 3)$, where φ_p is one of the STL formulae already found in the decision tree. One sees that, semantically, φ' overlaps with φ_p for times $[15, 20]$ and $1 \leq x \leq 2$. In this case, since φ_p is already discovered and present in the decision tree structure on the one hand, and φ' optimized for the current node, the "gain" brought by φ' can only be obtained in the part of φ' that does not overlap with φ_p. Therefore, we claim that the only valuable information laying in φ' is, in fact, in the difference between φ_p and φ', that is, in $\varphi' \backslash \varphi_p$, either $\square_{[15,20]}(0 \leq x \leq 1)$ or $\square_{[20,30]}(0 \leq x \leq 2)$.

Therefore, once the best candidate primitive and parameters are optimized into a candidate STL formula ϕ' (line 2), we calculate the difference between ϕ' and all STL formulae in φ^{path} already in the decision tree structure (lines 3–7). In case ϕ' and an STL formula ϕ_p in φ^{path} overlap (i.e., $\phi_p \cap \phi' \neq \emptyset$ – see Algorithm 3), we calculate $\phi' \backslash \phi_p$ and do this repeatedly on all STL formulae in φ^{path}. This operation is provided by Algorithm 4, which is a geometrical-based calculation of the difference between 2 STL formulae. It takes as input 2 formulae of the same structure (either (2a), (2b) or (2c)) to preserve semantic coherence), and conceptually transform these into convex 3D polytopes. The algorithm returns a set of convex polytopes describing the difference between 2 formulae. Note that each element in the difference (by construction of Algorithm 4, a cuboid) can be converted into an STL formula. The result is then processed in a second optimization process (line 8), where the gain of

Algorithm 4: $\phi \setminus \phi'$

```
Input: $\phi$, $\phi'$ – STL formulae of the form of (2a), (2b) or (2c), with set parameters.
Pre-condition: $\phi$ and $\phi'$ both of the same form.
Output: diff – set of STL formulae for $\phi \setminus \phi'$.
1  if $\phi \cap \phi' = \emptyset$ then return $\phi$
2  $diff \leftarrow \emptyset$
3  $x_s \leftarrow \{a, b\}$ ; $y_s \leftarrow \{c, d\}$ ; $t_s \leftarrow \{t_1, t_2\}$
4  if $a < a' < b$ then $x_s \leftarrow x_s \cup \{a'\}$
5  if $a < b' < b$ then $x_s \leftarrow x_s \cup \{b'\}$
6  if $c < c' < d$ then $y_s \leftarrow y_s \cup \{c'\}$
7  if $c < d' < d$ then $y_s \leftarrow y_s \cup \{d'\}$
8  if $t_1 < t'_1 < t_2$ then $t_s \leftarrow t_s \cup \{t'_1\}$
9  if $t_1 < t'_2 < t_2$ then $t_s \leftarrow t_s \cup \{t'_2\}$
10 for $(x_1^*, x_2^*), (y_1^*, y_2^*), (t_1^*, t_2^*) \in pairwise(x_s) \times pairwise(y_s) \times pairwise(t_s)$ do
11     if $\phi$ and $\phi'$ of the form of (2a) then
12         $diff \leftarrow diff \cup \{\diamond_{[t_1^*, t_2^*]}(x_1^* \leq x \leq x_2^* \wedge y_1^* \leq y \leq y_2^*)\}$
13     if $\phi$ and $\phi'$ of the form of (2b) then
14         $diff \leftarrow diff \cup \{\square[t_1^*, t_2^*](x_1^* \leq x \leq x_2^* \wedge y_1^* \leq y \leq y_2^*)\}$
15     if $\phi$ and $\phi'$ of the form of (2c) then
16         $diff \leftarrow diff \cup \{\square[t_1^*, t_2^*]\neg(x_1^* \leq x \leq x_2^* \wedge y_1^* \leq y \leq y_2^*)\}$
17 return diff
```

each element in the difference is evaluated. Finally, the best element ϕ^* is extracted and chosen to label the node. The remainder of the algorithm is similar to the baseline, except for the recursive call that appends ϕ^* to φ^{path} (line 11– 12).

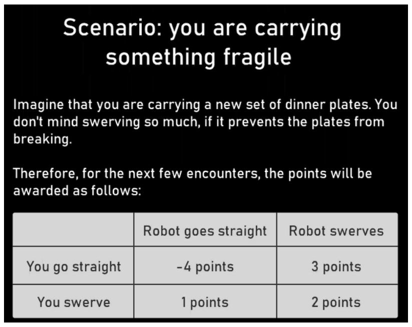
VI. Experiments

We implemented and tested our learning algorithm in Python 3.8[1]. We ran our experiments on an Intel i7-8665U CPU and 32GB RAM. We ran experiments on both a synthetic dataset and trajectories where, in a simulation, participants had to deviate their path to avoid collision with an incoming robot. We also compare the predictive power of our approaches to classical neural networks.

A. Synthetic Dataset

In this experiment, we generated 500 trajectories from 7 STL specifications. The purpose of this experiment is to see, whether in a controlled environment (we know which STL specification trajectories were generated from), we could learn equivalently good STL formulae to correctly classify the trajectories. We considered the following STL formulae:

$$\phi_1 = \square_{[0,100]}\neg(-2 \leq x \leq 2 \wedge -1 \leq y \leq 1) \tag{5a}$$
$$\phi_2 = \square_{[10,15]}(-6 \leq x \leq -2 \wedge -4 \leq y \leq -3) \tag{5b}$$
$$\phi_3 = \square_{[25,30]}(6 \leq x \leq 8 \wedge -6 \leq y \leq -2) \tag{5c}$$
$$\phi_4 = \diamond_{[40,80]}(-6 \leq x \leq -4 \wedge 3 \leq y \leq 4) \tag{5d}$$
$$\phi_5 = \diamond_{[50,70]}(1 \leq x \leq 4 \wedge 0 \leq y \leq 4) \tag{5e}$$
$$\phi_6 = \square_{[85,90]}(-3 \leq x \leq -1 \wedge 5 \leq y \leq 8) \tag{5f}$$
$$\phi_7 = \square_{[95,100]}(1 \leq x \leq 3 \wedge 5 \leq y \leq 8) \tag{5g}$$



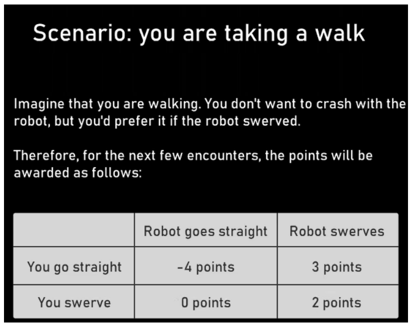
Scenario: you are carrying something fragile

Imagine that you are carrying a new set of dinner plates. You don't mind swerving so much, if it prevents the plates from breaking.

Therefore, for the next few encounters, the points will be awarded as follows:

	Robot goes straight	Robot swerves
You go straight	-4 points	3 points
You swerve	1 points	2 points

(a) *Safety* motivation.



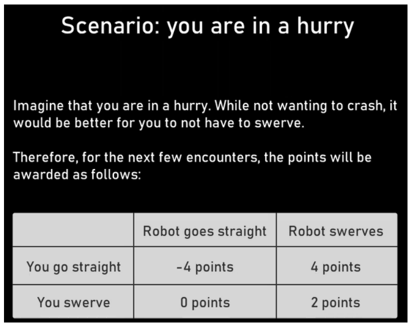
Scenario: you are taking a walk

Imagine that you are walking. You don't want to crash with the robot, but you'd prefer it if the robot swerved.

Therefore, for the next few encounters, the points will be awarded as follows:

	Robot goes straight	Robot swerves
You go straight	-4 points	3 points
You swerve	0 points	2 points

(b) *Taking a "normal" walk* motivation.



Scenario: you are in a hurry

Imagine that you are in a hurry. While not wanting to crash, it would be better for you to not have to swerve.

Therefore, for the next few encounters, the points will be awarded as follows:

	Robot goes straight	Robot swerves
You go straight	-4 points	4 points
You swerve	0 points	2 points

(c) *Speed* motivation.

Fig. 1: Payoff matrices and task description as displayed to the MTurk participants, for the motivations they had to describe.

Further, we generated trajectories given 3 classes:

$$c_1 = \phi_1 \wedge \phi_2 \wedge \phi_6 \tag{6a}$$

$$c_2 = \phi_1 \wedge \phi_3 \wedge \phi_4 \wedge \phi_7 \tag{6b}$$

$$c_3 = \phi_2 \wedge \phi_5 \wedge \phi_7 \tag{6c}$$

Since we consider the multi-class and multi-label case, trajectories could be labelled as $\{c_1\}$, $\{c_2\}$, $\{c_3\}$, $\{c_1, c_2\}$, $\{c_1, c_3\}$, $\{c_2, c_3\}$ and $\{c_1, c_2, c_3\}$. We generated trajectories from specifications of different (combinations of) classes using an MILP approach [20] and the Gurobi optimizer. We generated 100 trajectories for each of the classes $\{c_1\}$, $\{c_2\}$ and $\{c_3\}$, and 50 trajectories for each of the remaining classes $\{c_1, c_2\}$, $\{c_1, c_3\}$, $\{c_2, c_3\}$ and $\{c_1, c_2, c_3\}$.

TABLE I: Cross-validation results (average of the 5 folds) for the synthetic data, and for our multi-class (dt) and STL-difference ($dt_\triangle$) methods compared to a classical neural networks approach (nn).

H			A			time (s)		
nn	dt	$dt_\triangle$	nn	dt	$dt_\triangle$	nn	dt	$dt_\triangle$
0.001	0.001	0.000	0.999	0.998	1.000	65	174	465

We evaluated our method over these classes by a stratified shuffle split cross-validation (with five stratified randomized folds – made so that we preserve the percentage of trajectories in the data, for each class) and could get the results presented in Table I, where H represents the results in terms of hamming loss (7), and A the results in terms of example accuracy (8):

$$H(\mathscr{S}) = \frac{1}{|\mathscr{S}||\mathcal{C}|} \sum_{\sigma \in \mathscr{S}} \sum_{c \in \mathcal{C}} [I(l_\sigma^c \neq \hat{l}_\sigma^c)] \tag{7}$$

where l_σ^c is the *true* label of signal σ for class c, $\hat{l}_\sigma^c$ is the *predicted* label of signal σ for class c. In other words, in the case of multilabel classification, H is the percentage of incorrectly predicted labels to the total number of labels.

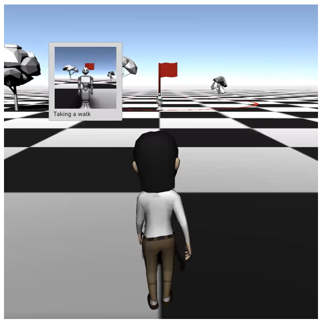
$$A(\mathscr{S}) = \frac{1}{|\mathscr{S}|} \sum_{\sigma \in \mathscr{S}} \frac{|l_\sigma \wedge \hat{l}_\sigma|}{|l_\sigma \vee \hat{l}_\sigma|} \tag{8}$$

where l_σ denotes the *true* labels of signal σ and $\hat{l}_\sigma$ the *predicted* labels of signal σ. In other words, A is the proportion of correctly predicted labels to the total number of labels. For comparison purposes of the predictive power of our decision-tree based STL formulae, we also trained on the multi-label trajectories and used the Keras deep learning Python library, a multi-layer perceptron model taking as input the trajectories and providing as output the probability of membership for each class of the trajectories.

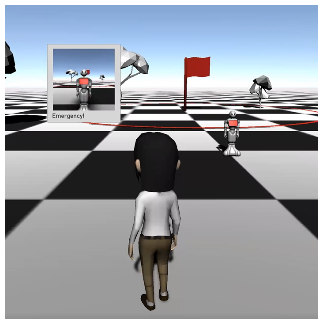
[1]Software, data and learned STL specifications available at https://github.com/KTH-RPL-Planiacs/stl_multiclass

B. Human-robot Encounters

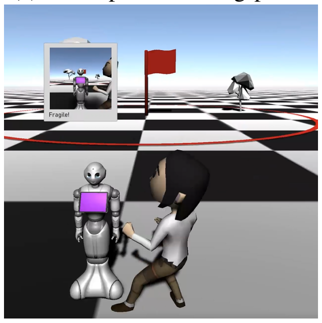
In this experiment, we collected human trajectories where, in a simulation, participants played a game where they had to reach a target zone while navigating around a robot moving towards them. We designed three different 'motivations': 1. "carrying something fragile" 2. "taking a walk" and 3. "in a hurry", each linked to a different payoff matrix based on the game-theoretic Game of Chicken (Fig. 1). Each of these makes for three different trajectory classes that we denote f, w and h. Participants completed 18 trials. For each trial, they were told what their motivation was, and what the



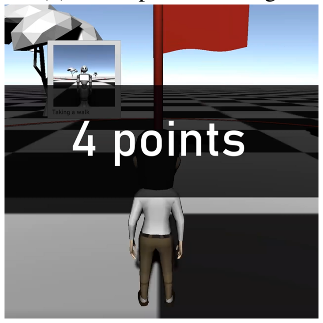
(a) Participant's starting point.



(b) Participant deviating.



(c) Collision.



(d) Participant's reward.

Fig. 2: Screenshots from the experiment: robot's and participant's avatars walking towards each other.

robot's motivation was. The three human motivation blocks were counterbalanced across participants, and the robot's motivations were randomised within these blocks, so that each participant played the same number of combinations of own and robot motivations.

At each trial, participants walked towards a goal, represented by a red flag, by means of an avatar (Fig. 2). Participants could use the 4 keyboard arrows to start moving and accelerate the speed of the avatar, to decelerate and 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, a widely-used social robot. 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. The game was developed in Unity version 2019.4.16f1 and exported to WebGL. Screenshots from the game are shown in Fig. 2. We recruited 50 participants (30 males, 20 females, median age 34 years old). Of these, 15 reported having interacted with a robot before, and 9 reported having played the Game of Chicken before. The experiment was conducted on Amazon Mechanical Turk (AMT) and lasted approximately 10 minutes. Participants were paid a base fee plus an additional reward based on how many points they earned in the experiment.

We collected a total of 900 trajectories (50 participants $\times$ 6 trials $\times$ 3 motivations) recorded with a frequency of 10 Hz, i.e. time steps of 100ms. The initial x/y-position of the participants was set in the robot's referential, that is, the robot's coordinates are at all times (0,0). In order to retain trajectories of participants who adopted homogeneous behaviors, we filtered out outliers as follows:

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 1.82, with a standard deviation of 1.67. We filtered out trajectories containing at least one data point with absolute value of x greater than $1.82 + 2 \times 1.67 = 5.16$.
2) the trajectories where participants completed the task too slowly. The minimum completion was done in 66 time steps, the maximum in 237 time steps, with an average of 91 and a standard deviation of 25.19. We removed trajectories completed with a number of time steps greater than $\lfloor 91 + 2 \times 25.19 \rfloor = 141$.
3) the trajectories where participants collided with the robot, as a 4th class of trajectories col.

Algorithm 5: MULTILABEL_TRAJECTORIES($\mathscr{S}$)

```
Input: $\mathscr{S}$ – set of single-label signals $(\sigma, c)$
       $\eta$ – Euclidean distance threshold
Output: $\mathscr{S}'$ – set of multi-label signals $(\sigma, \{c_1 \dots\})$
1 $\mathscr{S}' \leftarrow \emptyset$
2 for $(\sigma, c) \in \mathscr{S}$ do
3     $labels \leftarrow \{c\}$
      // d : Euclidean distance function
4     if $\exists (\sigma', c') \in \mathscr{S} \setminus \{\sigma\}\ s.t.\ \forall t,\ d(\sigma(t), \sigma'(t)) < \eta$
      then $labels \leftarrow labels \cup \{c'\}$
5     $\mathscr{S}' \leftarrow \mathscr{S}' \cup \{(\sigma, labels)\}$
6 return $\mathscr{S}'$
```

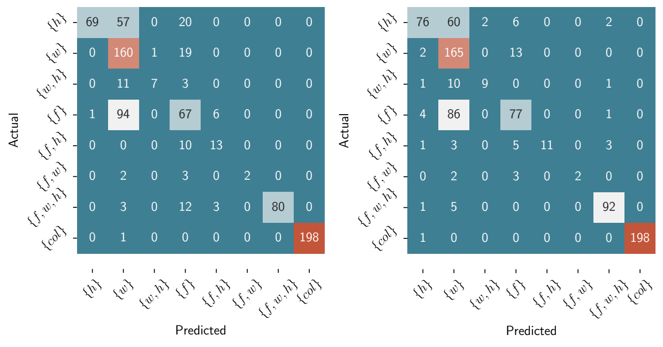
As a result, we used 842 trajectories, which we refer to as $\mathscr{S}$. The set of classes is denoted by $\mathcal{C} = \{f, w, h, col\}$. Since participants may have different conceptions and descriptions of the different behaviors, we notice some overlaps in the trajectories they depict. Indeed, some participant's trajectories for the "carrying something fragile" behavior might overlap the behavior of participants "taking a normal walk". Therefore, we associate these multiple labels to such trajectories (see Algorithm 5). The distribution of the trajectories is as follows: $\{f\} = 168$, $\{w\} = 180$, $\{h\} = 146$, $\{f, w\} = 7$, $\{f, h\} = 23$, $\{w, h\} = 21$, $\{f, w, h\} = 98$ and $\{col\} = 199$.

TABLE II: Cross-validation results of our baseline (dt) and STL-difference (dt_Δ) methods compared to a classical neural networks approach (nn), for the user study data, and a max tree height of 5.

fold	H			A			time (s)		
	nn	dt	dt_Δ	nn	dt	dt_Δ	nn	dt	dt_Δ
1	0.280	0.269	0.231	0.346	0.489	0.548	83	1853	2585
2	0.297	0.249	0.215	0.272	0.528	0.591	92	1935	2689
3	0.306	0.223	0.237	0.094	0.577	0.548	155	2019	2539
4	0.287	0.203	0.232	0.220	0.634	0.559	160	2139	2846
5	0.327	0.219	0.219	0.002	0.582	0.588	120	1905	2702
	0.299	*0.233*	*0.227*	*0.187*	*0.562*	*0.567*	*122*	*1970*	*2672*

TABLE III: Cross-validation results of our baseline (dt) and STL-difference (dt_Δ) methods compared to a classical neural networks approach (nn), for the user study data, and a max tree height of 10.

fold	H			A			time (s)		
	nn	dt	dt_Δ	nn	dt	dt_Δ	nn	dt	dt_Δ
1	0.280	0.244	0.286	0.346	0.533	0.472	83	3750	2250
2	0.297	0.215	0.249	0.272	0.590	0.529	92	3764	4394
3	0.306	0.249	0.237	0.094	0.532	0.557	155	3656	4289
4	0.287	0.243	0.213	0.220	0.539	0.592	160	3654	3514
5	0.327	0.225	0.207	0.002	0.567	0.605	120	3493	4210
	0.299	*0.235*	*0.238*	*0.187*	*0.552*	*0.551*	*122*	*3663*	*3731*



(a) mutli-class method.
(b) STL-difference method.

Fig. 3: Confusion matrices of the different classes of the user study, for the specifications learned on the entire data.

We evaluated our methods over the user study trajectories following the same procedure as in Sect. VI-A, and could get the results presented in Tables II and and III (maximum tree heights of 5 and 10, respectively), where H represents the results in terms of hamming loss, and A the results in terms of example accuracy. The purpose of varying the maximum tree height is to measure the influence of the size of rendered specifications on the predictive power of the models. In other words, Tables II and III show the trade-off between predictive power and interpretability (since a smaller specification is rather easier to interpret). The confusion matrices of the different classes of the user study, for specifications learned over the entire data, are displayed in Fig. 3. Rows represent the trajectories for the *true* sets of labels, while columns represent the trajectories in the *predicted* sets of labels.

C. Discussion

Considering the experiment with the synthetic dataset, we can observe that we perform nearly exact classification of the trajectories in the multi-label context. Further, the depth of the decision trees obtained remains small ($h = 3$), which allows us to render interpretable formulae as a result (for access and visualization of the models we learned, we refer the reader to our GitHub repository[1]). The results we obtained for the user study data do not reach exact identification of the STL formulae: this is due to the less "separable" problem at hand: indeed, some trajectories for one (pair of) user motivations are very close to other trajectories for other (pairs of) motivations. Also, a close look at the confusion matrices shows that most of the errors made are missing one of the labels in the prediction. We claim that finding at least one or 2 out of the 3 labels is already a fair outcome for our methods. Overall, our methods also perform better than classical neural networks designed to be trained on the multi-label case. Considering the STL-difference method, we can see that our extension of the multi-class STL learning algorithm renders semantically more interpretable specifications. Also, one of the pruning's side effects is that the learned specifications are less prone to overfitting, which explains why we obtain better cross-validation results. Finally, the primitives we work with are easily interpretable: they consist of 2D spatial specifications with time bounds, which can be well projected onto a map for visualization. Also, note that our work is extensible to richer fragments of STL, or the *first-level* primitives as defined by [8]: Algorithm 1 would require no modification, while Algorithm 2 would require to set finite bounds on the spatio-temporal space to calculate the difference between existing STL formulae in the decision-tree and a candidate STL formulae.

VII. Conclusion

We developed a technique to infer STL formulae from data labelled with multiple classes. We collected data in a simulation that places human users in a virtual environment with a robot and where users described different behaviors due to different goals. From the trajectories recorded from the study, we learned specifications of social navigation given different human internal states. The specifications learned per human preferences will, in the future, enable us to generate robot trajectories depending on human preferences or wishes, and perceived safety. As future work, we will look into the inclusion of *black-box* models (e.g., neural networks) to improve the learning of STL formulae from data, as well as the development of path planning techniques given our multi-class STL model.

References

[1] K. Charalampous, I. Kostavelis, and A. Gasteratos, "Recent trends in social aware robot navigation: A survey," *Robotics and Autonomous Systems*, vol. 93, pp. 85–104, 2017.

[2] 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.

[3] 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.

[4] 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.

[5] A. Linard, I. Torre, A. Steen, I. Leite, and J. Tumova, "Formalizing trajectories in human-robot encounters via probabilistic stl inference," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[6] H. Kress-Gazit, K. Eder, G. Hoffman, H. Admoni, B. Argall, R. Ehlers, C. Heckman, N. Jansen, R. Knepper, J. Křetínskỳ, *et al.*, "Formalizing and guaranteeing human-robot interaction," *Communications of the ACM*, vol. 64, no. 9, pp. 78–84, 2021.

[7] I. Torre, A. Linard, A. Steen, J. Tumová, and I. Leite, "Should robots chicken? how anthropomorphism and perceived autonomy influence trajectories in a game-theoretic problem," in *Proceedings of the 2021 ACM/IEEE International Conference on Human-Robot Interaction*, ser. HRI '21, 2021, p. 370–379.

[8] G. Bombara and C. Belta, "Offline and online learning of signal temporal logic formulae using decision trees," *ACM Trans. Cyber-Phys. Syst.*, vol. 5, no. 3, mar 2021.

[9] 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.

[10] 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.

[11] Z. Kong, A. Jones, A. Medina Ayala, E. Aydin Gol, and C. Belta, "Temporal logic inference for classification and prediction from data," in *Proceedings of the 17th International conference on Hybrid systems: computation and control (HSCC)*. ACM, 2014, pp. 273–282.

[12] P. Vaidyanathan, R. Ivison, G. Bombara, N. A. DeLateur, R. Weiss, D. Densmore, and C. Belta, "Grid-based temporal logic inference," in *56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 5354–5359.

[13] M. Vazquez-Chanlatte, J. V. Deshmukh, X. Jin, and S. A. Seshia, "Logical clustering and learning for time-series data," in *International Conference on Computer Aided Verification*, 2017, pp. 305–325.

[14] 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.

[15] 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.

[16] E. Aasi, C. I. Vasile, M. Bahreinian, and C. Belta, "Inferring temporal logic properties from data using boosted decision trees," *arXiv preprint arXiv:2105.11508*, 2021.

[17] E. Aasi, M. Cai, C. I. Vasile, and C. Belta, "Time-incremental learning from data using temporal logics," *preprint arXiv:2112.14300*, 2021.

[18] 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.

[19] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, vol. 4. IEEE, 1995, pp. 1942–1948.

[20] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," in *53rd Conference on Decision and Control (CDC)*. IEEE, 2014, pp. 81–87.