Real-time RRT* with Signal Temporal Logic Preferences

Alexis Linard*, Ilaria Torre*, Ermanno Bartoli, Alex Sleat, Iolanda Leite, Jana Tumova

***Abstract*— Signal Temporal Logic (STL) is a rigorous specification language that allows one to express various spatio-temporal requirements and preferences. Its semantics (called robustness) allows quantifying to what extent are the STL specifications met. In this work, we focus on enabling STL constraints and preferences in the Real-Time Rapidly Exploring Random Tree (RT-RRT*) motion planning algorithm in an environment with dynamic obstacles. We propose a cost function that guides the algorithm towards the asymptotically most robust solution, i.e. a plan that maximally adheres to the STL specification. In experiments, we applied our method to a social navigation case, where the STL specification captures spatio-temporal preferences on how a mobile robot should avoid an incoming human in a shared space. Our results show that our approach leads to plans adhering to the STL specification, while ensuring efficient cost computation.**

***Index Terms*— Signal Temporal Logic, Real-Time Planning, Sampling-based Motion Planning.**

I. Introduction

Recent research has broadened the scope of motion planning to handle advanced goals, requirements, and preferences that are defined using temporal logics: for instance, Linear Temporal Logic (LTL) can, among others, express various safety, reachability, sequencing, request-response requirements as well as their arbitrary combinations. Signal Temporal Logic (STL) [1] extends LTL with explicit temporal and spatial constraints for continuous signals. When viewing robot trajectories as signals, STL can express properties, such as that a robot should always keep a safe distance to people, or that it should move slow when in a narrow passage. A key benefit of STL is its quantitative semantics – robustness – which provides a score indicating to what extent a signal satisfies or violates a given specification.

Maximizing STL robustness has become a goal of a number of motion planning algorithms based on a variety of principles, from reformulation to a Mixed Integer Linear Programming problem [2], [3], to leveraging STL spatial robustness as part of a cost function in RRT* [4], [5], [6]. For instance, Karlsson et al. [4] define the cost function as a compromise between the STL spatial robustness and trajectory duration. This cost function was also utilized in [5], where the authors enhance the autonomous exploration planner with STL preferences to increase the exploration performance. Finally, the approach in [6] is particularly related to this work: the goal is to synthesise motion plans that maximize the spatial robustness of STL specifications. They define quantitative satisfaction of partial trajectories (i.e., trajectories for which there is not enough data to compute robustness) as *robust satisfaction intervals* [7], and guide the exploration process accordingly. The existing approaches have, however, certain limitations: they do not consider the presence of moving obstacles, and the structure of the used cost function prevents its efficient use in real time.

In this work, we address these limitations by extending the Real-Time RRT* (RT-RRT*) algorithm [8] to handle STL specifications. In contrast to other real-time variants of RRTs that either regrow the entire tree or prune infeasible branches [9], [10], RT-RRT* retains the whole tree in the environment and rewires its nodes based on the location of the tree root (which moves with the robot) and changes in dynamic obstacles. This allows for possible changes in the goal point and efficient rewiring of the nodes around the moving obstacles. Another algorithm, RRT$^{\text{X}}$ [11], operates in a similar fashion, but concentrates rewiring operations on sub-trees around dynamic obstacles to ensure asymptotic optimality. However, it may fail to avoid collisions with obstacles that move significantly faster than the robot.

In this work, we define an STL-based robustness function computed recursively along the nodes of the RRT* tree, which enables time efficiency for its online use. We handle a fragment of STL including *timed* and *untimed always* and *eventually* operators but restricting the *until* operator. Unlike in [6] and [7], our cost function returns a real number instead of robust satisfaction intervals. Indeed, RT-RRT* needs real values for the inclusion of the STL robustness in a cost function.

Our contributions are as follows:

- we define a cost function that includes a derivation of the robustness of an STL specification, that can be updated in real time and that we include in RT-RRT*.
- we integrate our real-time motion planning with STL preferences in simulations and real-world experiments and show that our approach leads to plans closely adhering to the STL specification.

II. 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 $[a, b] \subset \mathbb{N}$, $a, b \in \mathbb{N}, a \leq b$. Further, we denote $t + [a, b]$, $t \in \mathbb{N}$ by $[t + a, t + b]$. An n-dimensional,

*contributed equally. 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 the Vinnova Competence Center for Trustworthy Edge Computing Systems and Applications at KTH Royal Institute of Technology. We also thank José Pedro for his contribution. {linard,ilariat,bartoli,iolanda,tumova}@kth.se

finite, discrete-time signal σ is defined as a sequence $\sigma : \sigma(t_0)\sigma(t_1)\sigma(t_2)\dots\sigma(t_{\|\sigma\|})$, where $\sigma(t_i) \in \mathbb{R}^n$ is the value of signal σ at time $t_i \in \mathbb{N}$, $t_i < t_j$ if $i < j$, and $\|\sigma\|$ is the length of signal σ. The set of all signals with values taken from $\mathbb{R}^n$ is denoted by Σ. The syntax of STL is defined as follows [1]:

$$\phi ::= \top \mid \pi \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \mathcal{U}_{[a,b]} \phi_2$$

where $\top$ is the Boolean *True* constant, π a predicate over $\mathbb{R}^n$ in the form of $f(x) \sim \mu$, $f : \mathbb{R}^n \to \mathbb{R}$, $\mu \in \mathbb{R}$, and $\sim \in \{\leq, >\}$; $\neg$ and $\wedge$ are the Boolean operators for negation and conjunction, respectively; and $\mathcal{U}_{[a,b]}$ is the temporal operator *until* over bounded interval $[a, b]$. Other Boolean operations are defined using the conjunction and negation operators to enable the full expression of propositional logic. Additional temporal operators *eventually* and *always* are defined as $\Diamond_{[a,b]}\phi \equiv \top \mathcal{U}_{[a,b]}\phi$ and $\Box_{[a,b]}\phi \equiv \neg\Diamond_{[a,b]}\neg\phi$, respectively. In this paper, we consider the following fragment of STL:

$$\phi ::= \top \mid \pi \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \Diamond_{[a,b]}\phi \mid \Box_{[a,b]}\phi \mid \Diamond\phi \mid \Box\phi \quad (1)$$

where $\Diamond$ and $\Box$ denote the untimed *eventually* and *always* operators, respectively. This fragment enables the specification of conjunctions/disjunctions of preferences having to *eventually* or *always* hold within given time intervals ($\Diamond_{[a,b]}$, $\Box_{[a,b]}$) or indefinitely ($\Diamond$, $\Box$).

The *robustness* of an STL formula is a function $\rho : \Sigma \times \Phi^{\Sigma} \times \mathbb{N} \to \mathbb{R}$, recursively defined as [1]:

$$\begin{aligned}
\rho(\sigma, (f(\sigma) \sim \mu), t) &= \begin{cases} \mu - f(\sigma(t)) & \sim = \leq \\ f(\sigma(t)) - \mu & \sim = \geq \end{cases}, \\
\rho(\sigma, \neg\phi, t) &= -\rho(\sigma, \phi, t), \\
\rho(\sigma, \phi_1 \vee \phi_2, t) &= \max(\rho(\sigma, \phi_1, t), \rho(\sigma, \phi_2, t)), \\
\rho(\sigma, \phi_1 \wedge \phi_2, t) &= \min(\rho(\sigma, \phi_1, t), \rho(\sigma, \phi_2, t)), \\
\rho(\sigma, \Diamond_{[a,b]}\ \phi, t) &= \max_{t' \in t+[a,b]} \rho(\sigma, \phi, t'), \\
\rho(\sigma, \Box_{[a,b]}\ \phi, t) &= \min_{t' \in t+[a,b]} \rho(\sigma, \phi, t') \\
\rho(\sigma, \Diamond\ \phi, t) &= \max_{\forall t'} \rho(\sigma, \phi, t'), \\
\rho(\sigma, \Box\ \phi, t) &= \min_{\forall t'} \rho(\sigma, \phi, t').
\end{aligned}$$

where f is a function over signal σ and $\mu \in \mathbb{R}$. A signal *satisfies* an STL specification at a certain time t iff $\sigma(t) \models \phi \Leftrightarrow \rho(\sigma, \phi, t) \geq 0$. In previous work [12], we used STL as a suitable formalism for social navigation, to learn specifications of human-robot encounters from data. In a real-world scenario, we will reuse these specifications describing 2-D trajectories (in a space with dimensions x and y, see Sect. V-A).

III. Problem formulation

Let $\mathcal{X} \subseteq \mathbb{R}^n$ be the *bounded* workspace that contains (dynamic) obstacles $\mathcal{X}_{\text{obs}} \subset \mathcal{X}$, and the free space is denoted by $\mathcal{X}_{\text{free}} = \mathcal{X} \setminus \mathcal{X}_{\text{obs}}$. The current position of the robot is denoted by $\mathbf{x}_a$ and it is assumed to be known, as well as the position of the obstacles.

The RT-RRT* tree is denoted by $\mathcal{T}$ and each node is denoted by $\mathbf{x}_i \in \mathcal{X}$. The current tree root is denoted by $\mathbf{x}_0$, and $\mathcal{T}$ is rerooted to the position of the robot $\mathbf{x}_a$, when the position is updated (with a given refreshing frequency). The Euclidean distance d between nodes $\mathbf{x}_i$ and $\mathbf{x}_j$ is denoted by $d(\mathbf{x}_i, \mathbf{x}_j)$. The parent node of $\mathbf{x}_i$ is denoted by $\texttt{parent}(\mathbf{x}_i)$, and its children as $\texttt{children}(\mathbf{x}_i)$. $\varpi = (\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{\text{goal}})$ is a sequence returned by the motion planning algorithm, where $\mathbf{x}_{\text{goal}}$ is the position of the goal. We denote by σ_ϖ the trajectory of the robot tracking ϖ and this is the signal over which we evaluate an STL specification.

Since the planning algorithm involves rerooting of the tree $\mathcal{T}$ with the updated robot's position, we want to store, for each node $\mathbf{x}_i \in \mathcal{X}$ – and for monitoring purposes – the succession of nodes the robot has visited so far since the beginning of the planning process, plus the nodes in the path from the robot's actual position to $\mathbf{x}_i$. We denote by ξ_i such a trajectory taken by the robot until node $\mathbf{x}_i$. As an example, if $\mathbf{x}_0$ is the tree root, ξ_0 is the trajectory taken by the robot until $\mathbf{x}_0$ since the initialization of the algorithm. We denote the concatenation of 2 sequences $\varpi' = (\mathbf{x}_i, \dots, \mathbf{x}_j)$ and $\varpi'' = (\mathbf{x}_k, \dots, \mathbf{x}_l)$ as $\varpi' + \varpi'' = (\mathbf{x}_i, \dots, \mathbf{x}_j, \mathbf{x}_k, \dots, \mathbf{x}_l)$.

We formulate the problem of real-time motion planning under STL specifications, and in a dynamic environment as 2 sub-problems. Given a robot operating in a workspace $\mathcal{X}$, and a specification ϕ, we want to:

Problem 1: Find a suitable cost function J that assigns a cost to each possible sequence $\varpi = (\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_i)$ and takes into consideration the total completed distance, from $\mathbf{x}_0$ to each node, as well as the degree of satisfaction of the specification ϕ.

Problem 2: Extend RT-RRT* to find a motion plan $\varpi = (\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{\text{goal}})$, from the robot's current position to the goal, such that the cost of ϖ is minimized.

IV. Method

A. Cost function

In this section, we define a cost function J that includes a derivation of the robustness of an STL specification. We later use this function in STL-RT-RRT*. Since the real-time aspect of the algorithm is crucial, the calculation of the cost function is deemed efficient. To that end, after each rerooting of the tree $\mathcal{T}$, the computation of the cost of the nodes has to be as least time-consuming as possible. This drives the design of the cost function to be recursive, to enable fast computation of the updated costs.

Hereinafter, we define the robustness value $\bar{\rho}$ associated to a node $\mathbf{x}_i$ in $\mathcal{T}$. Since a trajectory until a given node in the RT-RRT* tree might be partially testable against the STL specification, we recursively define the robustness of (partial) trajectories until node $\mathbf{x}_i$. Consider for example the specification $\Diamond_{[4,5]}(x > 2)$. Until no observation of the trajectory for time t_4 has been made, it is not possible to assess the trajectory's robustness against the specification. Similar to [7], we use and maintain in memory the syntax tree of the STL formula, augmented with a robustness value $\bar{\rho}$ associated to the nodes in $\mathcal{T}$. As shown in Fig. 1, each vertex in the syntax tree corresponds to an STL operator ($\neg, \vee, \wedge, \Diamond_{[a,b]}, \Box_{[a,b]}, \Diamond, \Box$), and the leaves to a predicate π. We equip the temporal operators with the robustness values

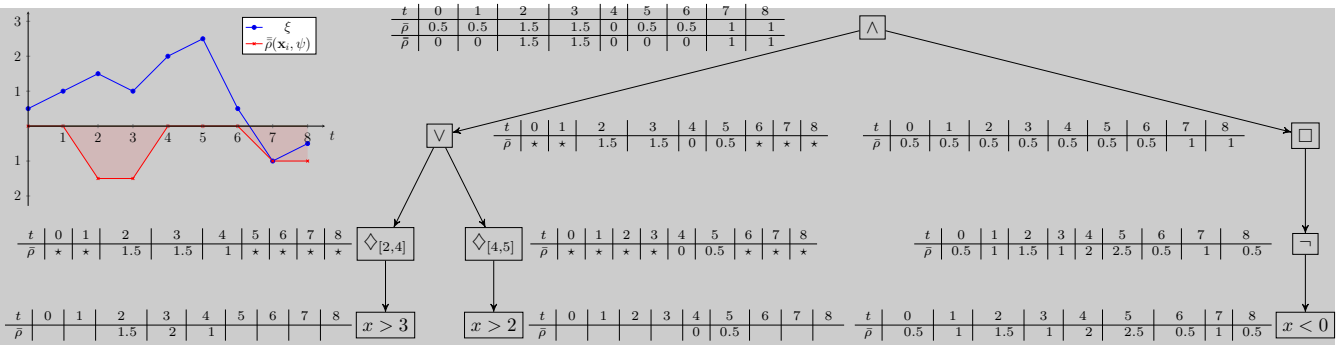


Fig. 1: Calculation of the $\bar{\rho}$ function for the specification $\psi = (\Diamond_{[2,4]}(x > 3) \vee \Diamond_{[4,5]}(x > 2)) \wedge \square\neg(x < 0)$ for 1-D nodes $\mathbf{x}_0 \dots \mathbf{x}_8$ (top left, blue), where ξ is the trajectory between nodes. The underlying data structure is the syntax tree of specification ψ, where for each temporal operator the nodes' values of $\bar{\rho}$ are stored. An entry marked '–' means that the value does not need to be computed. We also show the value of the $\bar{\bar{\rho}}$ function (top left, red) for the different nodes, as well as $\int_0^{\|\xi_i\|} \bar{\bar{\rho}}(\mathbf{x}_i, \psi, t)\, dt$ (top left, red area) used in the computation of $J_\psi(\mathbf{x}_i)$.

$\bar{\rho}$ for nodes $\mathbf{x}_i \in \mathcal{X}$, that are stored. Further, we define by $\bar{\rho}_{\text{parent}} : \mathcal{X} \times \Phi^\Sigma \to \mathbb{R} \cup \{\star\}$ the $\bar{\rho}$ value of a temporal operator in the syntax tree of the specification, for the parent of node $\mathbf{x}_i$ in the RT-RRT* tree, where $\star$ is a dummy symbol used to provide no real value to a formula, the robustness of which cannot be stated for a given trajectory (i.e., for trajectories shorter than the lower bound of the interval of a temporal operator). In the following, $\bar{\rho}_{\text{parent}}$ is called to compute the actual value of $\bar{\rho}$ for node $\mathbf{x}_i$ (since $\bar{\rho}$ depends on the value of $\bar{\rho}$ for the parent node of $\mathbf{x}_i$). The robustness value $\bar{\rho} : \mathcal{X} \times \Phi^\Sigma \times \mathbb{N} \to \mathbb{R} \cup \{\star\}$ associated to a node $\mathbf{x}_i$ in $\mathcal{T}$ is given by the recursive function:

$$\bar{\rho}(\mathbf{x}_i, (f(\xi_i) \sim \mu), t) = \begin{cases} \mu - f(\xi_i(t)) & \sim = \leq \\ f(\xi_i(t)) - \mu & \sim = \geq \end{cases}$$

$$\bar{\rho}(\mathbf{x}_i, \neg\phi, t) = \begin{cases} \star & \text{if } -\bar{\rho}(\mathbf{x}_i, \phi, t) = \star \\ -\bar{\rho}(\mathbf{x}_i, \phi, t) & \text{otherwise} \end{cases}$$

$$\bar{\rho}(\mathbf{x}_i, \phi_1 \vee \phi_2, t) = \begin{cases} \star & \text{if } \bar{\rho}(\mathbf{x}_i, \phi_1, t) \text{ and } \bar{\rho}(\mathbf{x}_i, \phi_2, t) = \star \\ \bar{\rho}(\mathbf{x}_i, \phi_1, t) & \text{if } \bar{\rho}(\mathbf{x}_i, \phi_2, t) = \star \\ \bar{\rho}(\mathbf{x}_i, \phi_2, t) & \text{if } \bar{\rho}(\mathbf{x}_i, \phi_1, t) = \star \\ \max(\bar{\rho}(\mathbf{x}_i, \phi_1, t), \bar{\rho}(\mathbf{x}_i, \phi_2, t)) & \text{otherwise} \end{cases}$$

$$\bar{\rho}(\mathbf{x}_i, \phi_1 \wedge \phi_2, t) = \begin{cases} \star & \text{if } \bar{\rho}(\mathbf{x}_i, \phi_1, t) \text{ and } \bar{\rho}(\mathbf{x}_i, \phi_2, t) = \star \\ \bar{\rho}(\mathbf{x}_i, \phi_1, t) & \text{if } \bar{\rho}(\mathbf{x}_i, \phi_2, t) = \star \\ \bar{\rho}(\mathbf{x}_i, \phi_2, t) & \text{if } \bar{\rho}(\mathbf{x}_i, \phi_1, t) = \star \\ \min(\bar{\rho}(\mathbf{x}_i, \phi_1, t), \bar{\rho}(\mathbf{x}_i, \phi_2, t)) & \text{otherwise} \end{cases}$$

$$\bar{\rho}(\mathbf{x}_i, \Diamond_{[a,b]}\phi, t) = \begin{cases} \star & \text{if } t < a \text{ or } t > b \\ \bar{\rho}(\mathbf{x}_i, \phi, t) & \text{if } t = a \\ \max(\bar{\rho}(\mathbf{x}_i, \phi, t), \bar{\rho}_{\text{parent}}(\mathbf{x}_i, \Diamond_{[a,b]}\phi)) & \text{otherwise} \end{cases}$$

$$\bar{\rho}(\mathbf{x}_i, \square_{[a,b]}\phi, t) = \begin{cases} \star & \text{if } t < a \text{ or } t > b \\ \bar{\rho}(\mathbf{x}_i, \phi, t) & \text{if } t = a \\ \min(\bar{\rho}(\mathbf{x}_i, \phi, t), \bar{\rho}_{\text{parent}}(\mathbf{x}_i, \square_{[a,b]}\phi)) & \text{otherwise} \end{cases}$$

$$\bar{\rho}(\mathbf{x}_i, \Diamond\phi, t) = \max(\bar{\rho}(\mathbf{x}_i, \phi, t), \bar{\rho}_{\text{parent}}(\mathbf{x}_i, \Diamond\phi))$$

$$\bar{\rho}(\mathbf{x}_i, \square\phi, t) = \min(\bar{\rho}(\mathbf{x}_i, \phi, t), \bar{\rho}_{\text{parent}}(\mathbf{x}_i, \square\phi))$$

For timed temporal operators, the design of $\bar{\rho}$ only uses the evaluation of predicates for the relevant time intervals. Outside of these, predicates are not evaluated, hence are not reflected in the calculation of the cost function. Also, the definition of $\bar{\rho}$ as such enables an easy computation of the min and max in the case of temporal operators: for a given node $\mathbf{x}_i$, the whole trajectory until node $\mathbf{x}_i$ doesn't need to be tested, but only the spatial coordinates of node $\mathbf{x}_i$ and the value of $\bar{\rho}$ of the temporal operator for the parent of $\mathbf{x}_i$, which leads to a lower computational complexity. We now define $\bar{\bar{\rho}} : \mathcal{X} \times \Phi^\Sigma \times \mathbb{N} \to \mathbb{R}$, that will be directly called by the RT-RRT* cost function:

$$\bar{\bar{\rho}}(\mathbf{x}_i, \phi, t) = \begin{cases} 0 & \text{if } \bar{\rho}(\mathbf{x}_i, \phi, t) = \star \\ \min(\bar{\rho}(\mathbf{x}_i, \phi, t), 0) & \text{otherwise} \end{cases} \quad (2)$$

Note that, by construction, $\bar{\bar{\rho}}$ is upper-bounded by 0. We now define the cost function J:

$$J(\mathbf{x}_i) = J_d(\mathbf{x}_i) + J_\phi(\mathbf{x}_i) \quad (3)$$

where $J_d(\mathbf{x}_i)$ is the total completed distance of trajectory until node $\mathbf{x}_i$, and $J_\phi(\mathbf{x}_i)$ the STL-based cost of $\mathbf{x}_i$ given specification ϕ. Further, we define $J_d(\mathbf{x}_i)$ and $J_\phi(\mathbf{x}_i)$ as:

$$J_d(\mathbf{x}_i) = d(\mathbf{x}_i, \texttt{parent}(\mathbf{x}_i)) + J_d(\texttt{parent}(\mathbf{x}_i)) \quad (4)$$

$$J_\phi(\mathbf{x}_i) = -\int_0^{\|\xi_i\|} \bar{\bar{\rho}}(\mathbf{x}_i, \phi, t)\, dt \quad (5)$$

where the integral is evaluated over discrete observations using the trapezoidal rule.

Lemma 4.1: Given $\bar{\bar{\rho}}(\mathbf{x}_i, \phi, t)$ in (2), $J(\mathbf{x}_i)$ in (3) and $J_\phi(\mathbf{x}_i)$ in (5) are monotonically increasing.

Lemma 4.2: $J(\mathbf{x}_i)$ in (3) is Lipschitz continuous.

Proof: (Sketch) Since $\bar{\bar{\rho}}$ in (2) is bounded (by most violating value of robustness in a bounded workspace) and called on the closed and bounded interval $[0, \|\xi_i\|]$, then integral in (5) is Lipschitz continuous. Hence, $J_\phi(\mathbf{x}_i)$ and $J(\mathbf{x}_i)$ are Lipschitz continuous. For the extensive proof, we redirect the reader to our Appendix[1]. ■

[1]See https://www.diva-portal.org/smash/record.jsf?pid=diva2:1746700

Algorithm 1: STL_RT_RRT*

Input: $\mathbf{x}_a$ – initial position, $\mathcal{X}_{\text{obs}}$ – set of obstacles, $\mathbf{x}_{\text{goal}}$ – goal position, ϕ – STL formula in form of (1).

1 `initialize` $\mathcal{T}$
2 **while** *goal is not reached* **do**
3 `update` $\mathbf{x}_a$, $\mathbf{x}_{\text{goal}}$, $\mathcal{X}_{\text{obs}}$, $\mathcal{X}_{\text{free}}$
4 `reroot` $\mathcal{T}$ to $\mathbf{x}_a$
5 `update_traj_until_node`$(\mathbf{x}_0, \xi_0 + \mathbf{x}_a)$
6 `update_stl_costs`$(\mathbf{x}_0, \phi)$
7 **while** *time left for tree extension* **do** `expand` $\mathcal{T}$
8 **while** *time left to rewire* **do** `rewire` nodes in $\mathcal{T}$
9 `plan` $\varpi = (\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\text{goal}})$
10 `move robot to` $\mathbf{x}_0, \mathbf{x}_1, \ldots$

Algorithm 2: UPDATE_TRAJ_UNTIL_NODE

Input: $\mathbf{x}$ – node in $\mathcal{T}$, ξ_{new} – trajectory until node $\mathbf{x}$

1 $\xi \leftarrow \xi_{\text{new}}$;
2 **for** $\mathbf{x}_i \in$ *children*$(\mathbf{x})$ **do**
3 `update_traj_until_node`$(\mathbf{x}_i, \xi + \mathbf{x}_i)$

B. Algorithm

We propose STL-RT-RRT* (Algorithm 1), an extension to RT-RRT* using the cost function J listed in (3). It introduces an online tree rewiring strategy that allows the tree root to move with the robot while updating the position of the obstacles with a given frequency.

First, we need to initialize a tree $\mathcal{T}$ (line 1, either with an empty tree or an existing tree computed offline) that will contain nodes sampling the workspace. We follow the same fixed-time loop consisting of first updating the robot's position, rerooting the tree to the current robot's position, and recursively updating the nodes' costs (lines 3–6). In addition to [8], to keep track of the robot's completed trajectory since the start of the algorithm, we update the trajectories until a given node so that the prefix of all ξ_i's for each node $\mathbf{x}_i$ is the robot's completed trajectory so far (Algorithm 1, line 5; Algorithm 2). Then, we can recursively update the cost of each node using the cost function J (line 6). Note that while updating the obstacle's position, we set the cost of the nodes blocked by the obstacle (i.e. nodes falling in the radius of the obstacle – as well as all the descendants of blocked nodes) to infinity. Further, after an obstacle has moved and "freed" some previously blocked nodes, the cost of the "freed" nodes is again set to J, and priority rewiring is performed around the "freed" nodes. To minimize the computational complexity of the re-rooting operations, we use a grid-based spatial indexing for the nodes in $\mathcal{T}$ to enable fast rewiring of neighbouring nodes [8].

The rest of Algorithm 1 is as follows: within the fixed computation time, we dedicate some time to expand the tree $\mathcal{T}$ (line 7). According to Algorithm 3 of [8], but using our cost function J instead, tree expansion in $\mathcal{X}_{\text{free}}$ is performed either by sampling the environment uniformly, creating a node inside an ellipsis containing the path from $\mathbf{x}_0$ to $\mathbf{x}_{\text{goal}}$ [13], or creating a node in the line between $\mathbf{x}_{\text{goal}}$ and the node of the tree that is the closest to $\mathbf{x}_{\text{goal}}$. Finally, we dedicate the rest of the computation time of the iteration to rewire the tree from the root [8, Algorithm 5] (that is, for a given node in the tree, find a potential better parent node regarding cost function J). The focus of rewiring is around the robot's position; rewiring operations are successively executed from the tree root to its children, and so recursively until rewiring the leaves of $\mathcal{T}$. Rewiring continues over iterations until all the leaves of the tree are reached. When no rewiring operation is left, we start rewiring from the tree root again at the next iteration.

The planned motion $\varpi = (\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\text{goal}})$ is a set of nodes starting from the tree root to the goal node (line 9). Note that if the goal node is blocked by an obstacle (or has infinity cost due to blockage of one of its ancestors), then the algorithm will plan from the tree root to the closest node to the goal that is not blocked itself (or has infinity cost due to blockage of one of its ancestors). Also, note that if an obstacle directly blocks the robot, no motion plan is returned (and the robot is commanded to stop). At each iteration, we move the robot to the next waypoint in the plan (line 10). When motion planning is done, and the robot is at the tree root, we change the tree root to the next immediate node after $\mathbf{x}_0$ in the planned path; hence we enable the robot to move along the planned path on the tree towards the goal.

The loop of line 2 is run until the goal is reached or a new goal is set. To dedicate more time to rewiring operations, a warm start is possible. We start by growing a large tree offline and then only dedicate the online computation to updating the obstacles, node costs, and rewiring of nodes. This is an advantage when the robot is deployed in a controlled environment (when, e.g., little sampling is needed).

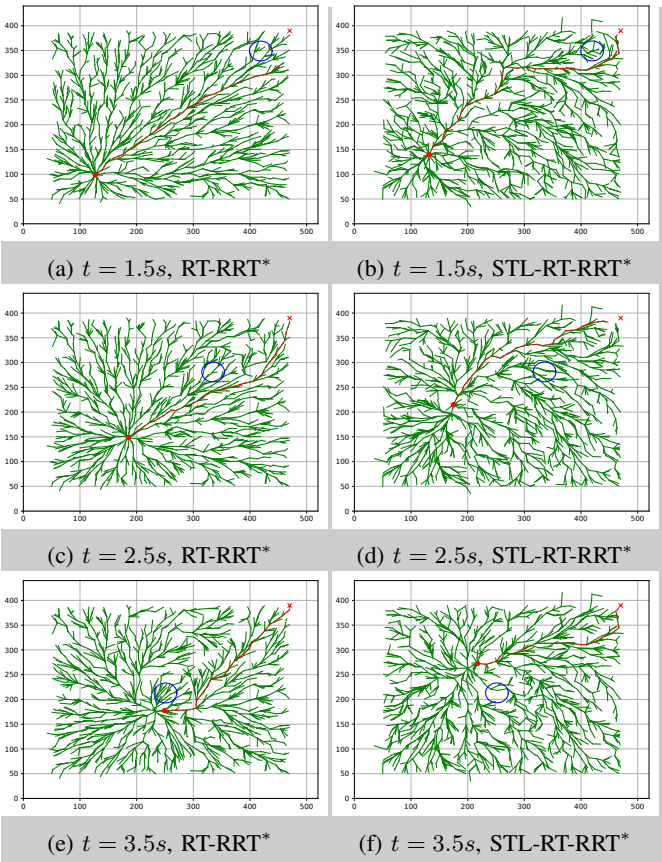
C. Analysis

a) Asymptotic optimality: The structure of Algorithm 1 and cost function J in (3) suggest that STL-RT-RRT* is asymptotically optimal. Given the fact that tree expansion is done in the same way as in RRT*, the proof follows from the asymptotic optimality of RRT*. Further, since our alternative optimality criterion is monotonically increasing and Lipschitz continuous (Lemmas 4.1 and 4.2), the returned trajectory is indeed optimal with respect to this criterion, provided that the refreshing frequency leaves enough time for the nodes' cost update, and tree rewiring.

b) Time complexity: The computational complexity of Algorithm 1 is $O(|\phi| \log(n))$ per iteration, where $|\phi|$ is the size of the formula. This follows from the computational complexity of RRT*, which is in $O(n \log(n))$ [14]. The computational complexity of our cost function is in $O(|\phi|)$, for the computation of the values for $\bar{\rho}$ for the temporal operators for trajectories shorter than the lower bound of the interval of a temporal operator, following results in [6], [7].

V. Evaluation

We implemented and tested our algorithm in Python 3.8, both in simulation (Sect. V-B), and in real-world experiments in a Motion Capture (MoCap) lab on Softbank Robotics Pepper, which is a commonly used social robot [15]. In both cases, we conducted experiments to highlight differences



(a) $t = 1.5s$, RT-RRT* (b) $t = 1.5s$, STL-RT-RRT*

(c) $t = 2.5s$, RT-RRT* (d) $t = 2.5s$, STL-RT-RRT*

(e) $t = 3.5s$, RT-RRT* (f) $t = 3.5s$, STL-RT-RRT*

Fig. 2: Simulations of RT-RRT* (left column) and STL-RT-RRT* (right column) in a 520x440cm room, at $t = 1.5s$, $t = 2.5s$ and $t = 3.5s$ from the initialization of the algorithm. The obstacle is represented by the blue circle, the robot by the red dot, and its planned trajectory at a given iteration by the red-dotted segments. The underlying tree $\mathcal{T}$ is in green.

between RT-RRT* and STL-RT-RRT* in terms of computational time (to expand the tree, update the costs of the nodes, and rewire the nodes), as well as the number of collisions between the robot and the dynamic obstacles [2].

A. Case study

Our case study considers social robot navigation and, more precisely, human-robot encounters in $\mathcal{X} \subseteq \mathbb{R}^2$. In this case, a human and a robot walk towards each other on a collision course in a square environment, and the robot should avoid the human in, preferably, a socially acceptable way. The preferences for avoiding the human "obstacle" are encapsulated in STL. In our earlier work [12], we focused on learning STL formulas that capture human preferences in such human-robot encounters. In this work, we use the formulas learned therein. The specification is defined in centimeters, in the human's coordinate system (i.e., where the human has coordinates (0,0) – so that the specification captures *how* the robot has to avoid the human) as $\phi = \varphi_{\text{left}} \vee \varphi_{\text{right}}$ with:

$$\varphi_{\text{left}} = \Diamond(-90 \leq x \leq -80 \wedge -90 \leq y \leq 0)$$
$$\varphi_{\text{right}} = \Diamond(70 \leq x \leq 85 \wedge -60 \leq y \leq 50)$$

[2]implemented at https://github.com/KTH-RPL-Planiacs/STL-RT-RRTstar

TABLE I: Nb. of trials the robot stopped during navigation ($\#R_{\text{stops}}$); nb. of trials with collisions (#col). Average time spent in the human's personal space (t_{hzone}); smallest distance between human and robot (min_d); robot completion time (t_{compl}); robot completed distance (d_{compl}); nb. of pairs of nodes in $\mathcal{T}$ tested for rewiring (#rewire); computation time for cost updates ($t_{\text{update } J_d}$ and $t_{\text{update } J_\phi}$) over 1000 trials.

Condition	RT-RRT*	STL-RT-RRT*
$\#R_{\text{stops}}$	286	4
#col	513	0

Condition	RT-RRT*				STL-RT-RRT*			
	min	*max*	avg	*stdev*	*min*	*max*	avg	*stdev*
t_{hzone} (s)	*1.1*	*1.9*	1.3	*0.15*	*0.8*	*1.6*	1.1	*0.12*
min_d (cm)	*0.1*	*43.5*	27.8	*8.7*	*35.3*	*94.9*	66.4	*9.9*
t_{compl} (s)	*7.2*	*8.6*	7.5	*0.26*	*6.6*	*8.2*	7.0	*0.21*
d_{compl} (cm)	*548*	*613*	562	*8.17*	*562*	*690*	589	*13.96*
#rewire	*5097*	*45952*	32180	*4964*	*91*	*39002*	2430	*2740*
of which successful	*0*	*87*	24	*11*	*0*	*2110*	113	*82*
$t_{\text{update } J_d}$ (ms)	*1.0*	*34.3*	1.7	*1.2*	*1.2*	*16.7*	1.4	*1.3*
$t_{\text{update } J_\phi}$ (ms)	-	-	-	-	*2.1*	*31.2*	9.2	*4.8*

B. Simulations

We ran our experiments on an Intel i7-8665U CPU and 32GB RAM. In a simulated environment, we recreated human-robot encounters where a human and a robot had to swap positions – i.e. the robot had to go from point A to point B, and the human from point B to point A. The simulated workspace corresponded to our 520x440cm MoCap facility (see Sect. V-C). Both RT-RRT* and STL-RT-RRT* were configured to handle the human as an obstacle of radius 25cm and the human and robot speeds were set to 1.1m/s and 0.55m/s respectively. The robot was commanded to stop if the human directly blocked the robot. Samples were drawn from a subset of the workspace up to 50cm away from the walls. The underlying tree contained a maximum of 2,000 nodes (to sample up to 2,000 nodes, computation times of 419ms for RT-RRT*; 589ms for STL-RT-RRT*). The duration of each iteration was set to 100ms. At each iteration, the human's position was randomized $\pm$ 10cm along the line leading to the human's goal, and the position of the robot $\pm$ 2cm from the first waypoint of the trajectory returned by the last iteration of the 2 planning algorithms. An execution of (STL-)RT-RRT* in shown in Fig. 2.

We measured, for each iteration of 100ms, how much time is spent to update the cost of each node using the cost function J (Algorithm 1, line 6). We also measured, for the remaining time, how many rewiring checks were performed (that is, for a given node in the tree, find a potentially better parent node than its current one regarding J). The number of rewiring operations (Algorithm 1, line 8) gives a good indication of the performance of the algorithm to update the underlying tree, hence to be able to keep on finding a motion plan to the goal that avoids the dynamic obstacle. From the results we obtained in Table I over 1000 trials, we can see that over 2,000 nodes, an iteration of 100ms can test up to 2,430 pairs of nodes for rewiring operations. On average, STL-RT-RRT* can successfully rewire up to 113 nodes per iteration of 100ms, which is more than RT-RRT* (merely 24).

This shows how the algorithm can keep on finding updated collision-free paths over time, while always updating STL costs in a reasonable time.

Compared to the baseline RT-RRT*, collisions were easily avoided (over 1000 trials, no collisions were reported STL-RT-RRT*, instead of 513 for RT-RRT*), which suggests that the STL specification could drive the robot to avoid the human with a sufficient distance. Further, we could observe that a robot driven by STL-RT-RRT* had to come to a full stop fewer times to let the human pass (over 1000 trials, only 4 times with STL-RT-RRT*, in contrast to 286 times with RT-RRT*). We can also note that the time the robot spent in the human's personal space (i.e., within a 1.2m radius) [16] is lower in the case of STL-RT-RRT*. This suggests that our algorithm could anticipate better how to avoid the incoming obstacle (see Fig. 2d and 2f), while maintaining low completion time and distance. This suggests also that the trajectory adheres well to the STL preferences on how to avoid the human.

C. Real-world setup

In a real-world experiment, we implemented and tested our algorithm on Pepper. For the evaluation, we asked human participants to walk from one end of a room to the other, in a straight line, while the robot walked in the opposite direction. Both with STL-RT-RRT* and RT-RRT*, the robot should avoid the human; given the simulation results, we hypothesize that our algorithm will perform better in terms of social acceptance, while not decreasing performance.

(a)

$$
\begin{aligned}
v_x &= k_1 \cos(\text{atan2}((y_{t+1} \quad y_t), \\
&\quad (x_{t+1} \quad x_t)) \quad \theta_t) \\
v_y &= k_1 \sin(\text{atan2}((y_{t+1} \quad y_t), \\
&\quad (x_{t+1} \quad x_t)) \quad \theta_t) \\
\omega &= \gamma
\end{aligned}
$$

(b)

Fig. 3: (a) MoCap lab, with disposition of the participant's starting point and robot's starting point; they are approximately 3.5 meters away from each other. (b) Feedback control law with linear velocities v_x and v_y, angular velocity ω and $k_1 > 0$ a defined constant.

The in-person evaluation was conducted in a 520x440cm MoCap lab designed to measure and digitally record the movement of the robot and the human participant (Fig. 3). The robot avoids the human according to the RT-RRT* and STL-RT-RRT* algorithms. Every 100ms, given the updated positions of the robot and the human, the algorithm provides an updated plan leading to the robot's goal point, while avoiding the human. The resulting plan ϖ consists in a set of waypoints, and at each iteration t the robot has a position $(x_t,\ y_t)$ and an orientation θ_t with respect to the global reference frame of the MoCap room. Considering that the next waypoint in the plan has coordinates $(x_{t+1},\ y_{t+1})$, we can define the pointing error $\gamma = \text{atan2}((y_{t+1} - y_t), (x_{t+1} - x_t)) + \pi - \theta_t$. We consider the robot as a nonholonomic wheeled mobile robot, and command it through the feedback control law in Fig. 3b to reach the next waypoint of the trajectory. Note that, since in our case planning is updated every 100ms, there is no guarantee that the robot will have reached the next waypoint before a new plan is returned. Further, if the trajectory is updated, so is the next waypoint.

TABLE II: Nb. of trials where the robot ($\#R_{\text{stops}}$) and human ($\#H_{\text{stops}}$) stopped during navigation; nb. of trials with collisions (#col). Average time spent in the human's personal space (t_{hzone}); smallest distance between human and robot ($\min_d$); robot completion time (t_{compl}) and robot completed distance (d_{compl}) in the two conditions.

Condition	RT-RRT*	STL-RT-RRT*
$\#R_{\text{stops}}$	36	8
$\#H_{\text{stops}}$	30	5
#col	10	1

Condition	RT-RRT*				STL-RT-RRT*			
	min	*max*	avg	*stdev*	*min*	*max*	avg	*stdev*
t_{hzone} (s)	1.2	24.5	5.5	4.5	1.6	10.2	3.4	1.3
$\min_d$ (cm)	11.9	59.0	35.5	11.1	29.2	77.1	52.0	10.1
t_{compl} (s)	5.6	39.6	20.4	9.3	5.6	31.4	22.0	5.0
d_{compl} (cm)	12	427	211	114	29	431	240	103

a) Results: We recruited a total of 16 participants. To study the statistical variability between STL-RT-RRT* and RT-RRT*, we conducted repeated measures analysis of variance (ANOVA) for all objective and subjective outcomes. Details on the procedure, the user study design, as well as the objective [17], [18] and subjective metrics [19] on social acceptance can be found in Appendix[1].

We found that the robot in our STL-RT-RRT* condition kept an overall larger distance from the human than the robot in the baseline RT-RRT* condition ($F(1, 22334) = 441.80, p < .001$) as prescribed by the STL preference; the minimum distance kept at each trial was also larger in the STL-RT-RRT* condition ($F(1, 79) = 70.67, p < .001$); the time spent within the human's personal space (< 1.2 metres) was shorter in the STL-RT-RRT* condition ($F(1, 79) = 9.34, p < .005$); there were more collisions and stops in the RT-RRT* condition than in the STL-RT-RRT* condition (see Table II); but the time taken by the robot to reach its goal was not statistically different between the two conditions ($F(1, 79) = 1.29, p = .26$).

For the subjective measures, the robot in the STL-RT-RRT* condition was perceived as more intelligent ($F(1, 30) = 13.63, p < .001$), predictable ($F(1, 30) = 4.10, p = .05$), and as having more social competence ($F(1, 30) = 4.51, p = .04$) than the robot in the RT-RRT* condition. There were no statistically significant differences between the two conditions on the perceived safety scale.

VI. Conclusion and Future Work

We developed a real-time motion-planning algorithm under STL preferences. Our results showed that adhering to STL preferences in real-time is tractable, despite a slightly longer time update of the cost of the nodes, as well as a lower amount of rewiring operations. We applied our

algorithm to a social robot, where the STL specifications encapsulate desired ways for a robot to perform encounters with an incoming human. The resulting plan leads to fewer collisions and stops than the baseline algorithm. Further, in a user study, our results showed that the adherence to STL preferences learned from trajectories in simulations [12] can lead to improved performance of the robot from a human's perspective, while not sacrificing performance (the robots in both conditions took, on average, the same amount of time to reach their destination). Our approach improves the social acceptance of the robot, both with objective and subjective measures. An interesting line of work to pursue would be to analyze the impact of the STL preference on human behaviour and to see whether a new specification could be inferred from the adapted behaviour of people (and eventually observe convergence in the long run).

A limitation of our method when dealing with multiple obstacles – or when in an environment cluttered with obstacles – would be the lack of guarantee, in the case of timed temporal operators, that completion of part of the specifications (i.e., *timed eventually* or *always*) would be satisfied on time. Indeed, in case the robot is blocked by an obstacle (e.g., a human in a narrow passage), the robot would have to stop, which might incur delays until the robot can move again to complete the rest of the plan. The algorithm, instead, at every iteration and given the context, finds the best motion plan given the cost function J and the position of the obstacles.

In future work, we will include human motion prediction [20], [21] in our planning algorithm and study how different human internal states and goals influence social navigation [22]. We will also look into integrating STL to broader social navigation scenarios [23]. For instance, we would like to consider drone navigation around humans under STL preferences on how to perform 3D navigation. Our method can be easily applied to cases with higher dimensions since expanding the underlying tree and handling signals in STL with extra dimensions requires minor changes in our implementation. Further, STL has recently been used to guarantee the functioning of robotic interactions with humans and perceived safety [24]: we wish to study, as a future direction, how to align STL preferences with human perception of robots.

References

[1] 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.

[2] 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 IEEE Conference on Decision and Control*. IEEE, 2014, pp. 81–87.

[3] A. Rodionova, L. Lindemann, M. Morari, and G. J. Pappas, "Time-robust control for stl specifications," in *2021 60th IEEE Conference on Decision and Control (CDC)*, 2021, pp. 572–579.

[4] J. Karlsson, F. S. Barbosa, and J. Tumova, "Sampling-based motion planning with temporal logic missions and spatial preferences." *21st IFAC World Congress*, vol. 53, no. 2, pp. 15 537–15 543, 2020.

[5] F. S. Barbosa, D. Duberg, P. Jensfelt, and J. Tumova, "Guiding autonomous exploration with signal temporal logic," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3332–3339, 2019.

[6] C.-I. Vasile, V. Raman, and S. Karaman, "Sampling-based synthesis of maximally-satisfying controllers for temporal logic specifications," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 3840–3847.

[7] J. V. Deshmukh, A. Donzé, S. Ghosh, X. Jin, G. Juniwal, and S. A. Seshia, "Robust online monitoring of signal temporal logic," *Formal Methods in System Design*, vol. 51, no. 1, pp. 5–30, 2017.

[8] K. Naderi, J. Rajamäki, and P. Hämäläinen, "Rt-rrt*: A real-time path planning algorithm based on rrt*," in *Proceedings of the 8th Conference on Motion in Games*, ser. MIG '15. New York, USA: Association for Computing Machinery, 2015, p. 113–118.

[9] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *IEEE/RSJ international conference on intelligent robots and systems*, vol. 3. IEEE, 2002, pp. 2383–2388.

[10] B. D. Luders, S. Karaman, E. Frazzoli, and J. P. How, "Bounds on tracking error using closed-loop rapidly-exploring random trees," in *Proceedings of the 2010 american control conference*. IEEE, 2010, pp. 5406–5412.

[11] M. Otte and E. Frazzoli, "Rrtx: Asymptotically optimal single-query sampling-based motion planning with quick replanning," *International Journal of Robotics Research*, vol. 35, no. 7, pp. 797–822, 2016.

[12] A. Linard, I. Torre, I. Leite, and J. Tumova, "Inference of multi-class stl specifications for multi-label human-robot encounters," in *International Conference on Intelligent Robots and Systems*, 2022.

[13] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, "Informed sampling for asymptotically optimal path planning," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 966–984, 2018.

[14] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[15] 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.

[16] E. T. Hall and E. T. Hall, *The hidden dimension*. Anchor, 1966, vol. 609.

[17] I. Kostavelis, A. Kargakos, D. Giakoumis, and D. Tzovaras, "Robot's workspace enhancement with dynamic human presence for socially-aware navigation," in *international conference on computer vision systems*. Springer, 2017, pp. 279–288.

[18] C. Mavrogiannis, P. Alves-Oliveira, W. Thomason, and R. A. Knepper, "Social momentum: Design and evaluation of a framework for socially competent robot navigation," *ACM Transactions on Human-Robot Interaction (THRI)*, vol. 11, no. 2, pp. 1–37, 2022.

[19] M. Rubagotti, I. Tusseyeva, S. Baltabayeva, D. Summers, and A. Sandygulova, "Perceived safety in physical human–robot interaction—a survey," *Robotics and Autonomous Systems*, vol. 151, p. 104047, 2022.

[20] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social lstm: Human trajectory prediction in crowded spaces," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 961–971.

[21] C. S. Swaminathan, T. P. Kucner, M. Magnusson, L. Palmieri, S. Molina, A. Mannucci, F. Pecora, and A. J. Lilienthal, "Benchmarking the utility of maps of dynamics for human-aware motion planning." *Frontiers in Robotics and AI*, vol. 9, 2022.

[22] 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.

[23] C. Mavrogiannis, F. Baldini, A. Wang, D. Zhao, P. Trautman, A. Steinfeld, and J. Oh, "Core challenges of social robot navigation: A survey," *arXiv preprint arXiv:2103.05668*, 2021.

[24] 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.

[25] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," *Robotics Science and Systems VI*, vol. 104, no. 2, 2010.

[26] C. Bartneck, D. Kulić, E. Croft, and S. Zoghbi, "Measurement instruments for the anthropomorphism, animacy, likeability, perceived intelligence, and perceived safety of robots," *International Journal of Social Robotics*, vol. 1, no. 1, pp. 71–81, 2009.