Reactive motion planning for temporal logic tasks without workspace discretization

Ashkan Zehfroosh and Herbert G. Tanner

***Abstract*— The curse of dimensionality is a challenge in many applications of Linear Temporal Logic robot motion planning, and is linked to the discretization of the robot's workspace. The discretization aggravates the problem by introducing a multitude of atomic propositions. This paper argues that a large portion of these atomic propositions is unnecessary. It demonstrates this point by introducing local navigation functions within a temporal logic planning framework, and utilizing register automata for reactive motion planning without explicit, high-resolution workspace discretization. Motivation for this approach comes from applications in pediatric motor rehabilitation involving play-based social child-robot interactions, where the appropriate robot behavior in response to child actions is best described in temporal logic terms. A simulation example drawn from the aforementioned pediatric rehabilitation studies illustrates the advantages of the approach.**

I. Introduction

Linear Temporal Logic (LTL) has expressive power to describe a wide variety of high-level robotic missions. Robotic planning using LTL has traditionally been based on model checking LTL specification in automata [1]. This approach generally involves two steps [2], [3]: the construction of (i) a Buchi automaton encoding the LTL task, and (ii) a discrete abstraction of the plant dynamics in the form of a deterministic transition system. The complexity of this process is exponential in the size of the LTL formula [1], and it is implicitly assumed that the valuation of the atomic propositions in the task specification are under the control of the robot. This problem can be solved by reactive LTL synthesis [4], with doubly exponential complexity in the size of the LTL formula. The latter can make the solution intractable for some applications.

Generalized Reactivity(1) (GR(1)) synthesis appears to have addressed the complexity issue [5]. The price comes in terms of expressivity: GR(1) covers only a special subclass of LTL —although the restriction does not seem to result in considerable loss [6]. Along the lines of GR(1) synthesis one would be given sensor information about environment actions (truth value of atomic propositions *not* under the control of the robot), and be asked to realize some desired behavior as a reaction (set the truth value of atomic propositions under robot's control), either in the form of motion or an inanimate response [7]. The solution provided is in the form of a game strategy, continuous execution of which allows timely reaction to changes in the environment in addition to satisfaction of the given specification [7].

GR(1) synthesis is computationally efficient in general; yet, the size of the state-space in this game is exponential in the number of atomic propositions in the LTL specification, a challenge that is typically referred to as *state-explosion* [8]. Multiple mitigating methods have been suggested, including a receding horizon approach [9], [10]. The problem is exacerbated in multi-robot planning cases, and decentralized solutions are desirable [11].

Most LTL planning formulations discretize the robot's workspace, therefore increasing the number of atomic propositions, and complicating further the expression of task specifications. However, many of the atomic propositions introduced are not central to expressing the specification; rather they are artifacts of the discretization process itself. The main idea of this paper therefore is to perform GR(1) synthesis without high-resolution workspace discretization. The workspace is still partitioned, but this decomposition is dictated primarily by the specifications given. Rather than thinking of robot motion as transition from cell to cell, gross robot motion is driven by globally convergent *navigation function* [12] controllers. Linking the continuous robot space to the discrete "mind" of the planner is facilitated by the adaptation and utilization of a computation model known as a *register automaton* [13]–[16].

Motivation for this approach comes from applications in early pediatric rehabilitation, in which robots try to socially interact with children who have motor disabilities. This play-based interaction between robot and child is designed to encourage physical activity as a means of assisting and triggering new motor developmental transitions [17]–[19], since motor skills and mobility has been shown to promote cognition [20]–[22]. Preliminary work [23] has offered some evidence that in this context, appropriate and effective robot reactions to children behavior are better modeled in LTL.

II. Technical Preliminaries

Given a set of atomic propositions AP, LTL formulas are recursively defined as

$$\varphi ::= \text{True} \mid a \in \text{AP} \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \bigcirc \varphi \mid \varphi \,\mathfrak{u}\, \varphi'$$

Given negation ($\neg$) and conjunction ($\wedge$), one defines disjunction ($\vee$), implication ($\implies$) and equivalence ($\iff$); given temporal operators "until" ($\mathfrak{u}$) and "next" ($\bigcirc$), one obtains "finally" ($\Diamond\varphi = \text{True}\ \mathfrak{u}\ \varphi$) and "always" ($\Box\varphi = \neg\Diamond\neg\varphi$).

Ashkan Zehfroosh, and Bert Tanner are with the Department of Mechanical Engineering, University of Delaware. {ashkanz, btanner}@udel.edu

This work was supported by NIH under grant # R01HD87133.

The truth value of an LTL formula is evaluated over infinite sequence σ of (temporal) truth assignments to the atomic propositions in AP. Denote $\sigma(i)$, the set of atomic propositions that are true at position i. The suffix of σ from position i is denoted $\sigma[i\ldots]$. Whether σ satisfies LTL formula φ, written $\sigma \models \varphi$, can be recursively defined as[1]

$$\begin{array}{ccc}
\sigma \models a & \text{iff} & a \in \sigma(0) \\
\sigma \models \varphi_1 \wedge \varphi_2 & \text{iff} & \sigma \models \varphi_1 \text{ and } \sigma \models \varphi_2 \\
\sigma \models \neg\varphi & \text{iff} & \sigma \not\models \varphi \\
\sigma \models \varphi_1 \vee \varphi_2 & \text{iff} & \sigma \models \neg(\neg\varphi_1 \wedge \neg\varphi_2) \\
\sigma \models \varphi_1 \Rightarrow \varphi_2 & \text{iff} & \sigma \models \neg\varphi_1 \vee \varphi_2 \\
\sigma \models \bigcirc\varphi & \text{iff} & \sigma[1\ldots] \models \varphi \\
\sigma \models \varphi_1 \mathfrak{u} \varphi_2 & \text{iff} & \left\{ \begin{array}{c} \exists j \geqslant 0,\ \ \sigma[j\ldots] \models \varphi_2 \text{ and} \\ \forall\ 0 \leqslant i < j,\ \ \sigma[i\ldots] \models \varphi_1 \end{array} \right\} \\
\sigma \models \Diamond\varphi & \text{iff} & \exists i \geqslant 0,\ \ \sigma[i\ldots] \models \varphi \\
\sigma \models \Box\varphi & \text{iff} & \forall i \geqslant 0,\ \ \sigma[i\ldots] \models \varphi
\end{array}$$

The "always-eventually" operator ($\Box\Diamond$) is also useful

$$\sigma \models \Box\Diamond\varphi \quad \text{iff} \quad \forall i \geqslant 0,\ \exists j \geqslant i,\ \ \sigma[j\ldots] \models \varphi$$

A formula free of temporal operators, is called *Boolean*.

A. Atomic propositions

Consider the finite set of atomic propositions $\text{AP} = E \cup Y$, where E is the set of atomic propositions controlled by the environment, and which the robot can react to after evaluating their truth value, and Y is the set of atomic propositions the truth value of which is under robot control.

B. Task specification

Specifications are expressed in a special class of LTL [7], according to which if the environment adheres to certain LTL assumptions φ_e, the robot should satisfy an LTL formula φ_r:

$$\varphi = \varphi_e \Rightarrow \varphi_r \tag{1}$$

A formula φ_α for $\alpha \in \{e, r\}$, is a conjunction of: (a) Boolean formulas φ_α^i that specify *initial conditions* of the environment and the robot. (b) *Environment safety guarantees* $\varphi_e^t = \bigwedge_{j=1} \Box B_i$ where B_i is a Boolean formula involving propositions in $Y \cup E \cup E'$, and E' containing all propositions in E prefixed by the next operator $\bigcirc$; and *robot safety guarantees* $\varphi_r^t = \bigwedge_{j=1} \Box A_j$, where A_j is a Boolean formula involving propositions in $Y \cup E \cup Y' \cup E'$, with Y' constructed from Y just like E' is from E. (c) *Goal specification* φ_α^g in the form $\bigwedge_{j=1} \Box\Diamond D_j$ where Boolean formulas D_j are constructed from propositions in $Y \cup E$.

A specification φ is said to be *unsatisfiable* if φ_r cannot be satisfied no matter what the behavior of the environment is. A specification φ is said to be *unrealizable* if the environment can force a violation of φ. A specification is *unsynthesizable* if it is either unsatisfiable or unrealizable. The environment plays *fair* if its behavior satisfies $\varphi_e^i \wedge \varphi_e^t$ [7].

[1]For a comprehensive exposition of LTL semantics, see [24].

C. GR(1) solutions

Checking for synthesizablity of, and designing a strategy to satisfy a given task specification φ, can be done using GR(1) [5]. This framework considers the problem as a two-player game $G = (V, E, Y, \varphi_e^i, \varphi_r^i, \varphi_e^t, \varphi_r^t, \varphi^g)$ between the robot and the environment, where $V = Y \cup E$. The game states are the elements of 2^V.

The game starts at initial states $s_0 \in S_0 \subset 2^V$ such that $s_0 \models \varphi_e^i \wedge \varphi_s^i$. Labeling functions $\gamma_E : 2^V \to 2^E$ and $\gamma_Y : 2^V \to 2^Y$ single out the atomic propositions $e \subseteq E$ and $y \subseteq Y$, respectively, that are true in the particular game state. In every state $s \in 2^V$, the environment chooses a next state having atomic propositions $e' \in 2^E$ such that $(s, e') \models \varphi_e^t$, in the sense that atomic propositions E and Y are evaluated based on $\gamma_E(s)$ and $\gamma_Y(s)$ respectively, while atomic propositions $\bigcirc E$ are evaluated based on e'. Once the environment has "played" the robot makes a move picking atomic propositions $y' \in 2^Y$ such that $(s, e', y') \models \varphi_r^t$; similarly here, atomic propositions E and Y are evaluated based on $\gamma_E(s)$ and $\gamma_y(s)$, while those in $\bigcirc E$ and $\bigcirc Y$ are evaluated based on e' and y', respectively. The robot wins if $\varphi^g = \varphi_e^g \Rightarrow \varphi_r^g$.

There is a set of (winning) states $W \subseteq 2^V$ from which the robot can force its winning condition [5]. If $S_0 \subseteq W$, then the specification is *synthesizable* [25].

The winning strategy is a transition function [25]

$$\delta_s : W \times 2^E \to W \tag{2}$$

which prescribes its next winning robot "move." All state sequences $\sigma = [s_0, s_1, s_2, \ldots]$ generated based on δ_s, will satisfy the specification (1).

III. Problem Statement

Let $R = \{r_1, r_2, \ldots, r_n\}$ be a set of *region propositions* defining n disjoint subsets of the robot's workspace that have to be visited, and thus necessary to encode the task specification. Let $A = \{a_1, a_2, \ldots, a_m\}$ be a set of Boolean (action) propositions which the robot can render true or false at any time, and set $Y = R \cup A$. Then with reference to game G (Section II-C), define $\gamma_A : 2^V \to 2^A$ as the labeling function that identifies the robot actions licensed by the specification at each state of the game, and $\gamma_R : 2^V \to R$ the labeling function that projects to the robot's region associated with the game state.

The robot evolves with single integrator dynamics

$$\dot{x}(t) = u(t),\ \ x(t) \in X \subset \mathbb{R}^2,\ \ u(t) \in U_r \subset \mathbb{R}^2 \tag{3}$$

Proposition r_i is true if x belongs in the corresponding region. Proposition a_i becomes true if the robot is performing the corresponding action. In what follows, hopefully without causing confusion, the term *action* is used to refer to both the atomic proposition, and the physical event the robot triggers. The robot can "multitask," and $\text{act}(t) \subseteq A$ denotes the set of actions performed at time t, meaning that $a_i \in \text{act}(t) \iff a_i = 1$ (True) at time t.

Problem 1: Given robot model as (3) and a synthesizable task φ as (1), we want to find the control input $u(t)$ as well as $\mathrm{act}(t)$ at each time t, such that the robot's behavior satisfies the task φ in a fair environment.

IV. Technical approach

A. Overview

The approach followed here combines two components hierarchically: a planner sits at the top and determines a series of waypoints that need to be hit, and actions from $\mathrm{act}(t)$ to be triggered in sequence for specification (1) to be satisfied; at the bottom we have a continuous feedback controller u responsible for taking the system through these waypoints in the order prescribed. The two components are interlinked, and communication flows bidirectionally at the frequency of a discrete clock that runs at the time-scale of the continuous dynamics.

Consider first of the continuous control u that drives the robot with dynamics $\dot{x} = u$ between the different regions of interest: from r_i to $r_{i'}$. A navigation function $F_{ii'}$ is constructed, parameterized by its goal configuration x_g (selected as a point inside $r_{i'}$ and treating regions r_j for $i' \neq j \neq i$ as obstacles), and control is assigned as $u = -\nabla F_{ii'}$.

The high-level planner is a particular register automaton $T = \{Q, Q_0, F, \Sigma, D, h, \tau, \Delta\}$ having

Q	a finite set of states
$Q_0 \subseteq Q$	the set of initial states
F	set of final states
Σ	the finite alphabet
D	a infinite set of vector data[α]
h	a (vector) register[β]
τ	the register assignment map[γ]
Δ	the transition function[δ]

[α] examples of which can be elements of $\mathbb{R}^n$.

[β] the register is essentially a static memory buffer.

[γ] $\tau : \{1, 2, \ldots, h\} \to D \cup \{\#\}$ assigns data to registers; symbol $\#$ represents an empty register. Given input data atom $(\sigma, d) \in \Sigma \times D \cup \{\#\}$, a test $\mathrm{Test}(\tau)$ is performed on the registers by evaluating Boolean formulas constructed from $\mathrm{AP} \cup \{\tau(i) \preceq d\}$ with $i \in \{1, \ldots, h\}$, where $\preceq$ denotes element-wise inequality.

[δ] Assume that the automaton is at state $q \in Q$, the register content is given by τ, and the machine reads $(\sigma, d) \in \Sigma \times D \cup \{\#\}$ at its input. Then there can be two types of transitions

$$\big([q, \tau], \phi_r\big) \xrightarrow{(\sigma,\mathrm{d})} [q', \tau] \tag{4}$$

$$\big([q, \tau], \phi_w^i\big) \xrightarrow{(\sigma,\mathrm{d})} [q', \tau'] \tag{5}$$

with the following semantics: In (4), the register test ϕ_r is performed and if ϕ_r evaluates true then the automaton jumps to state q'; this is referred to as a *read transition*. In (5), the register test ϕ_w^i is performed and if it evaluates true then the automaton jumps to state q' and overwrites the registers with d; this is referred to as a *write transition*.

Thus the high-level planner monitors the evolution of the lower-level continuous dynamics and stores it in its vector registers. The register conditions $\mathrm{Test}(\tau)$ then inform it when it is time to trigger a transition and licensed by the task specification. This transition determines a new navigation and Boolean actions ($\mathrm{act}(t)$) for the robot.

B. Particularization to the motivating application

This section instantiates the register automaton of Section IV-A in a way that serves the purposes of the motivating application described in Section I and sets the stage for the numerical analysis of Section V. The mathematical constructions of Sections II and III are now recalled.

The set of *states* is $Q = 2^{Y \cup E}$ —same as those of the game. Recalling that $Y = R \cup A$ and $V = Y \cup E$, a (region membership) function $r : X \to R$ identifies which element in R contains a given $x \in X$.

The *initial states* of the register automaton Q_0 are the same as the initial states of the game structure G. The set of *final states* is set to $F = \varnothing$; this register automaton has essentially the structure of a *semiautomaton* [26]. Alternatively, one can think of all states being final. The *alphabet* of this register automaton is a copy of its state set $\Sigma = Q$. The *data* space of this machine is $D = X \subset \mathbb{R}^2$. The *register* matches the size of the data: it is a two dimensional array, which is essentially used to hold the current position of the robot.

The *transition function* Δ processes inputs to the register automaton that are produced by Algorithm 1 and the current observed position of the robot. Algorithm 1 gets the current environment state $e(t)$ and current state/register value of the automaton $[q, \tau]$ and returns the next state that need to be visited s (according to δ_s) and the intermediate goal point x_g that the robot should head toward (in order to be able to make transition to state s). The current observed position x_c takes the place of the data atom d in the automaton's input (σ, d); the symbolic part of the input is generated by Algorithm 1 which is the next state of game G to be visited (as explained above) i.e., $\sigma = s$. In other words, game G through Algorithm 1 attempts to push T to a new state, while the continuous dynamics periodically updates the automaton's register. The transition in T occurs if the register test condition licenses it. Specifically, upon receiving input $(\sigma, d) = (s, x_c)$ at state q with register content τ, with s provided by the game and x_c by the continuous dynamics, the register automaton runs the test $r(d) = \gamma_R(s)$; if true, it takes a write transition

$$\big([q, \tau], [r(x_c) = \gamma_R(s)]\big) \xrightarrow{(s,x_c)} [s, x_c]$$

and if the test fails, the automaton takes the write transition

$$\big([q, \tau], [r(x_c) \neq \gamma_R(s)]\big) \xrightarrow{(s,x_c)} [q, x_c]$$

Concurrently, Algorithm 1 informs the continuous dynamics of the current goal position x_g; that information, together with the robot's current position determine the navigation function utilized: If $x_c \in r_i$ and $x_g \in r_j$, pick F_{ij} (Fig. 1). At the same time, the register automaton T keeps track of

the visiting states and decides which immediate actions (i.e. $\mathrm{act}(t)$) should be activated at each time to ensure that the overall robot behavior satisfies the LTL specification.

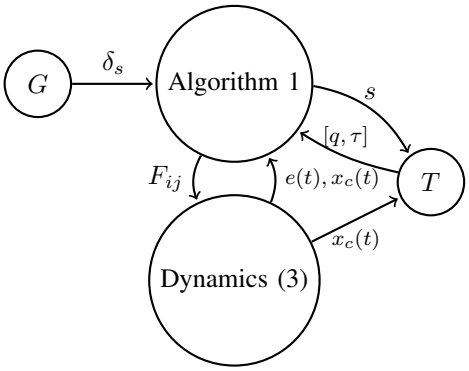

Fig. 1. Diagram showing the flow of information between the three key components of the planner: game G, Algorithm 1, and the register automaton T.

Fig. 2. Robot workspace in a child-robot interaction scenario. Two robots interact with the subject: NAO and Dash.

Input: current register automaton state q, current robot's position $x_c(t)$, observed (sensed) environment signal $e(t)$, last step goal point x'_g, transition function δ_s, arbitrary small neighborhood ϵ around a point that indicates converging to that point.

1) next state $s = \delta_s(q, e(t))$
2) intermediate goal region $r_g = \gamma_R(s)$
3) randomly choose a point $x_g \in r_g$
4) if $(r(x'_g) = r_g) \wedge (x_c(t) \notin \epsilon(x'_g))$ then $x_g = x'_g$

Return s, x_g
End

Algorithm 1: Algorithm of finding intermediate goal positions.

V. Validation

In the pediatric rehabilitation example that motivated the approach outlined in this paper (Fig. 2), infants with motor disabilities are socially interacting with robots in play-based scenarios that involve physical activity. The play-based intervention is intended to incite infant mobility. If this form of human-robot interaction (HRI) is to be automated, the robot has to have a way of deciding what is the intervention-appropriate response to child behavior in order to keep her engaged in play-based physical activity.

Similar to other instances of HRI application reported in literature [27]–[29], a Markovian model is used to model the interaction at the high level. The parameters of this Markovian model are learned through observations in sessions with human subjects [23], [30]. Initial approaches to encoding high-level robot actions in this play-based intervention were based on describing simple atomic responses like "approach infant," "step back from infant," "stand still," "make a sound," etc. Analysis of mobility data from rehabilitation sessions, however, revealed that robot behaviors that appeared to trigger infant responses were rarely atomic; they usually involved some combination of proximity conditions with a temporal pattern of cycling between atomic behaviors. For example, as a social non-verbal cue to "follow me," the robot would typically succeed if it initially approached the child up to about a meter, stood still for a short time interval, and then attempted to increase the distance slightly.

We thus conjecture that robot responses modeled in an LTL framework may be more effective in triggering the desired subject responses. Toward this end, consider the workspace of Dash (blue robot in Fig. 2) featured in Fig. 3. The workspace consists of three labeled regions $R = \{r_1, r_2, r_3\}$, with the robot initialized in region r_2, while and the human subject is still. In plain English, we would encode our task specification related to getting the human subject to follow the robot as follows:

> Infinitely often visit region r_3 (reduce distance), and infinitely often visit r_2 (back and forth movements), and infinitely often make a sound or flash a light. Never visit region r_1 (do not increase distance beyond its initial value). If the child starts to follow you, do not remain inside or visit r_3 (to encourage more mobility).

When the human subject changes her state from "not moving" to "moving," the high-level plan is considered successful, the system resets, and the robot waits for the decision-making algorithm to decide a new optimal high-level behavior to implement.

Let us see now how the task specification can be encoded in LTL. Define first the atomic propositions $\mathrm{AP} = E \cup Y$:

$E = \{m\}$ is the environment signal that shows if the child is moving or not. If m is true the infant is moving, and false otherwise.

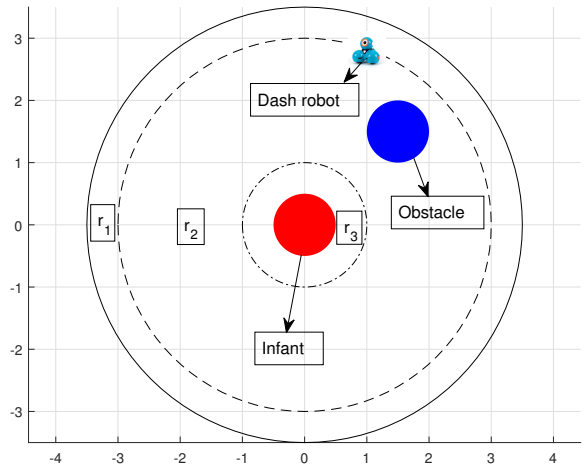

Fig. 3. Schematic of Dash's robot workspace.

$Y = R \cup A = \{r_1, r_2, r_3, a^{\text{sound/light}}\}$ are the robot's atomic propositions. Predicate r_i being true means the robot is in region i. Similarly, if Boolean action $a^{\text{sound/light}}$ evaluates true the robot is making a sound and flashing a light.

Consider now the following robot and environment formulae for expressing (1):

$\varphi_e^i = \neg m$: the subject is not moving when the action is introduced.

$\varphi_e^t = \square(\text{True})$: no specific environmental assumptions.

$\varphi_e^g = \square\lozenge(\text{True})$: no specific environment goal.

$\varphi_r^i = r_2 \wedge \neg a^{\text{sound/light}}$: the robot initiates the action in region r_2 without making a sound or flashing lights.

$$\begin{aligned}\varphi_r^t = \; & \square[r_1 \implies (\bigcirc r_1 \vee \bigcirc r_2)] \\ & \wedge \square[r_2 \implies (\bigcirc r_1 \vee \bigcirc r_2 \vee \bigcirc r_3)] \\ & \wedge \square[r_3 \implies (\bigcirc r_2 \vee \bigcirc r_3)] \\ & \wedge \textstyle\bigwedge_j \square[\bigcirc r_j \implies \neg \bigwedge_{i \neq j} \bigcirc r_i] \\ & \wedge \square[\neg \bigcirc r_1] \\ & \wedge \square[\bigcirc m \implies \neg \bigcirc r_3]\end{aligned}$$

The first part of φ_r^t encodes workspace topology, the second part captures the mutual exclusion region constraint (robot in one region only), and the third specifies the desired robot behavior.

$$\varphi_r^g = \square\lozenge[r_3 \vee m] \;\; \wedge \;\; \square\lozenge[a^{\text{sound/light}}] \;\; \wedge \;\; \square\lozenge[r_2 \vee m]$$

With the exception of partitioning into regions of interest, the approach above does not involve any uniform high resolution workspace discretization. For comparison purposes, consider a typical convex polygonal workspace discretization shown in Fig. 4, with resolution dictated by the physical size of Dash, which is approximately 20 cm $\times$ 20 cm. This discretization alone brings the length of the specification and the size of the game machine to 9×10^4 and 2^{207}, respectively, with complexity being exponential in these parameters. In contrast, the reported approach affords a task specification of length 92 and a game machine with 32 states.

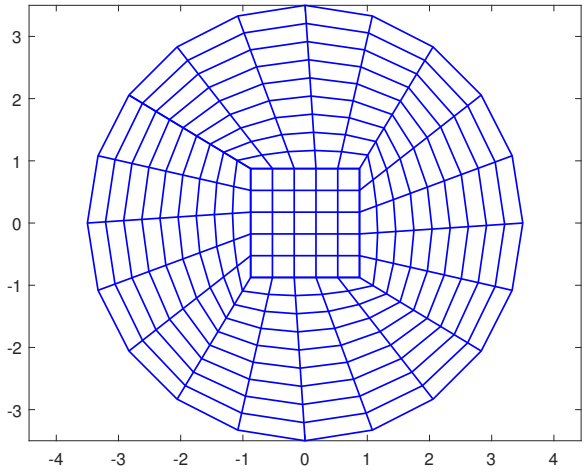

Fig. 4. A convex polygonal workspace discretization.

Implementing now the high-level planner as a combination (Fig. 1) of game G + Algorithm 1 + register automaton T, and populating the set of required navigation functions allows one to automate the behavior of the robot as it interacts with the subject. Figure 5 illustrates a possible scenario run in simulation.

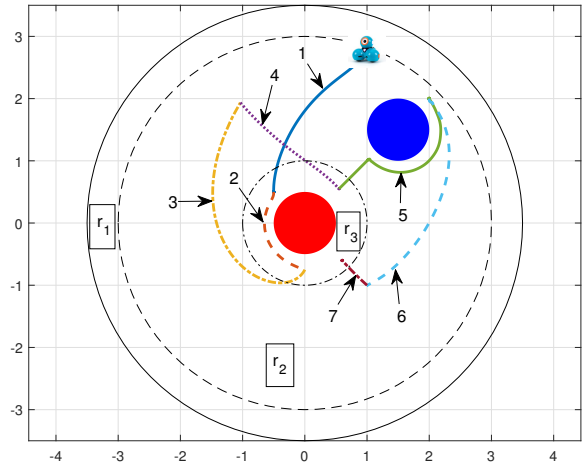

Fig. 5. Simulated HRI scenario with the robot steered by the planner. The interaction between robot and subject in this scenario consists of seven distinct stages.

The simulated HRI scenario of Fig. 5 includes 7 interaction stages. In stages 1 through 4, the robot senses $\neg m$, and so it performs random back and forth motion between regions r_2 and r_3, while running $a^{\text{sound/light}}$ attempting to attract the subject's attention. In stages 5 and 6, the robot senses m, and

returns to region r_2 where it waits. Stage 7 brings back $\neg m$ measurements, and the robot resumes its attention-grabbing routine of stages 1–4.

VI. Conclusion

Reactive LTL robot motion planning robot is feasible without resorting to relatively high-resolution discretization of the robot's physical workspace. In fact, a workspace partitioning that identifies the regions of interest with regards to problem constraints and task specification, is sufficient. The workspace discretization that would otherwise be necessary to track the evolution of the concrete continuous dynamics, and capture them in discrete form, can be circumvented with the introduction of an interface between the concrete continuous dynamics and the very high-level planner, that takes the form of a register automaton and a collection of navigation functions for the robot. The introduction of such an interface promises substantial computational savings.

References

[1] Moshe Y Vardi and Pierre Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences*, 32(2):183–221, 1986.

[2] Georgios E Fainekos, Hadas Kress-Gazit, and George J Pappas. Temporal logic motion planning for mobile robots. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2020–2025. IEEE, 2005.

[3] Geogios E Fainekos, Hadas Kress-Gazit, and George J Pappas. Hybrid controllers for path planning: A temporal logic approach. 2005.

[4] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 179–190. ACM, 1989.

[5] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive (1) designs. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 364–380. Springer, 2006.

[6] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Where's waldo? sensor-based temporal logic motion planning. In *2007 IEEE International Conference on Robotics and Automation*, pages 3116–3121. IEEE, 2007.

[7] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

[8] Hadas Kress-Gazit, Tichakorn Wongpiromsarn, and Ufuk Topcu. Correct, reactive, high-level robot control. *IEEE Robotics & Automation Magazine*, 18(3):65–74, 2011.

[9] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M Murray. Receding horizon temporal logic planning for dynamical systems. In *Proceedings of the 48h IEEE Conference on Decision and Control CDC held jointly with 2009 28th Chinese Control Conferenc*, pages 5997–6004. IEEE, 2009.

[10] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M Murray. Receding horizon control for temporal logic specifications. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, pages 101–110. ACM, 2010.

[11] Jana Tumova and Dimos V Dimarogonas. Decomposition of multi-agent planning under distributed motion and task ltl specifications. In *54th IEEE Conference on Decision and Control CDC*, pages 7448–7453. IEEE, 2015.

[12] Elon Rimon and Daniel E Koditschek. Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation*, 8(5):501–518, 1992.

[13] Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.

[14] Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Transactions on Computational Logic (TOCL)*, 5(3):403–435, 2004.

[15] Yael Cohen-Sygal and Shuly Wintner. Finite-state registered automata for non-concatenative morphology. *Computational Linguistics*, 32(1):49–82, 2006.

[16] Jie Fu and Herbert G Tanner. Optimal planning on register automata. In *2012 Annual American Control Conference ACC*, pages 4540–4545. IEEE, 2012.

[17] Karen Adolph. Motor development. *Handbook of child psychology and developmental science*, 2:114–157, 2015.

[18] Laura A Prosser, Laurie B Ohlrich, Lindsey A Curatalo, Katharine E Alter, and Diane L Damiano. Feasibility and preliminary effectiveness of a novel mobility training intervention in infants and toddlers with cerebral palsy. *Developmental Neurorehabilitation*, 15(4):259–66, 2012.

[19] Karina Pereira, Renata Pedrolongo Basso, Ana Raquel Rodrigues Lindquist, Louise Gracelli Pereira da Silva, and Eloisa Tudella. Infants with Down syndrome: percentage and age for acquisition of gross motor skills. *Research in Developmental Disabilities*, 34(3):894–901, 3 2013.

[20] J. J. Campos, D. I. Anderson, M. A. Barbu-Roth, E. M. Hubbard, M. J. Hertenstein, and D. Witherington. Travel broadens the mind. *Infancy*, 1(2):149–219, 2000.

[21] M. W. Clearfield. The role of crawling and walking experience in infant spatial memory. *Journal of Experimental Child Psychology*, 89:214–241, 2004.

[22] Eric A Walle and Joseph J Campos. Infant language development is related to the acquisition of walking. *Developmental Psychology*, 50(2):336–348, 2014.

[23] A. Zehfroosh, E. Kokkoni, H. G. Tanner, and J. Heinz. Learning models of human-robot interaction from small data. In *2017 25th Mediterranean Conference on Control and Automation*, pages 223–228, July 2017.

[24] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[25] Kai Weng Wong, Rüdiger Ehlers, and Hadas Kress-Gazit. Correct high-level robot behavior in environments with unexpected events. In *Robotics: Science and Systems*, 2014.

[26] Jie Fu, Herbert G. Tanner, Jeffrey N. Heinz, Konstantinos Karydis, Jane Chandlee, and Cesar Koirala. Symbolic planning and control using game theory and grammatical inference. *Engineering Applications of Artificial Intelligence*, 37:378—391, 2015.

[27] Frank Broz, Illah Nourbakhsh, and Reid Simmons. Planning for Human-Robot Interaction in Socially Situated Tasks: The Impact of Representing Time and Intention. *International Journal of Social Robotics*, 5(2):193–214, 2013.

[28] Tirthankar Bandyopadhyay, Kok Sung Won, Emilio Frazzoli, David Hsu, Wee Sun Lee, and Daniela Rus. Intention-Aware Motion Planning. In E. Frazzoli et al., editor, *Algorithmic Foundations of Robotics X*, volume 86, pages 475–491. Springer-Verlag, 2013.

[29] Catharine LR McGhan, Ali Nasir, and Ella M Atkins. Human intent prediction using markov decision processes. *Journal of Aerospace Information Systems*, 5(12):393–397, 2015.

[30] A. Zehfroosh, H. G. Tanner, and J. Heinz. Learning option mdps from small data. In *2018 Annual American Control Conference ACC*, pages 252–257. IEEE, 2018.