Temporal Logic Motion Planning for Mobile Robots*

Georgios E. Fainekos, Hadas Kress-Gazit and George J. Pappas
GRASP Laboratory, Departments of ESE and CIS
University of Pennsylvania
Philadelphia, PA 19104, USA
{*fainekos,hadaskg,pappasg*}*@grasp.upenn.edu*

***Abstract*— In this paper, we consider the problem of robot motion planning in order to satisfy formulas expressible in temporal logics. Temporal logics naturally express traditional robot specifications such as reaching a goal or avoiding an obstacle, but also more sophisticated specifications such as sequencing, coverage, or temporal ordering of different tasks. In order to provide computational solutions to this problem, we first construct discrete abstractions of robot motion based on some environmental decomposition. We then generate discrete plans satisfying the temporal logic formula using powerful model checking tools, and finally translate the discrete plans to continuous trajectories using hybrid control. Critical to our approach is providing formal guarantees ensuring that if the discrete plan satisfies the temporal logic formula, then the continuous motion also satisfies the exact same formula.**

***Index Terms*— Motion planning, temporal logics, model checking, discrete abstractions, hybrid control.**

I. INTRODUCTION

Robot motion planning problem has historically focused on generating trajectories which reach a goal configuration while avoiding obstacles [1], [2]. Mathematically formulating specifications such as motion sequencing, synchronization, or temporal ordering of different motions present new challenges for motion planning, as they require not only novel formulations, but also powerful computational approaches due to the inherent problem complexity.

Formally defining such specifications can be achieved using temporal logics, such as linear temporal logic (LTL) and computation tree logic (CTL), developed in concurrency theory. The applicability of temporal logics in robotics was advocated as far back as [3]. Over the years, the formal methods community has developed very sophisticated model checking tools such as SPIN [4] and NUSMV [5], which verify whether a discrete transition system satisfies a temporal logic formula. More recently, model checking approaches have been used for discrete planning in order to satisfy temporal logic specifications. This research has led to planning algorithms and tools such as MBP [6], TLPLAN [7] and UMOP [8]. These tools generate high-level, discrete plans that do not take into consideration the dynamic model of the robot, resulting in potentially infeasible plans.

*This work is partially supported by NSF EHS 0311123, NSF ITR 0324977, and ARO MURI DAAD 19-02-01-0383.

This paper addresses the novel problem of generating *continuous* trajectories for mobile robots while satisfying formulas in temporal logic. Our approach first lifts the problem to the discrete level by partitioning the environment into a finite number of equivalence classes. A variety of partitions are applicable, in particular the cellular decomposition in [9] or the triangular decomposition in [10]. The partition results in a natural discrete abstraction for robot motion which is used then for planning using model checking tools, in particular SPIN and NUSMV.

In order to ensure that the discrete plan is feasible at the continuous level, the decomposition must satisfy the so-called bisimulation property [11]. Bisimulations allow us to prove that if the abstract, discrete robot model satisfies the LTL formula, then the continuous robot model also satisfies the same formula. To ensure this critical property we utilize the hybrid control framework of [10], even though the framework of [9] is equally applicable but computationally more demanding.

Related work can be found in the hybrid systems community, and in particular the recent work of [12] which focuses on designing controllers for discrete-time control systems in order to satisfy temporal logic specifications. In [13], controllers are designed for satisfying LTL formulas by composing controllers using navigation functions [14]. In [15], the UPPAAL model checking tool for timed automata has been used for multi-robot motion planning using CTL formulas, but without taking into account the dynamics of the robots. This paper differentiates itself from all previous approaches by building upon the framework proposed in [10] which has, comparatively, the best computational properties, is fully automated, and is ideally suited for interfacing with model checking tools.

In addition to addressing this novel problem, we believe that this direction of research is important for at least three reasons. First, this work formally connects high-level planning with low-level control, resulting in a mathematically precise interface between discrete AI planning and continuous motion planning. Second, the mapping from temporal logic to physical motion is the first important step in the mapping from natural language to physical motion in a compositional manner. Finally, this work can be extended to multi-agent environments where formal specifications and computational solutions will result in verified coordination logic for cooperating robots.

II. Problem Formulation

We consider a fully actuated, planar model of robot motion operating in a polygonal environment P. The motion of the robot is expressed as

$$\dot{x}(t) = u(t) \quad x(t) \in P \subseteq \mathbb{R}^2 \quad u(t) \in U \subseteq \mathbb{R}^2 \quad (1)$$

where $x(t)$ is the position of the robot at time t, and $u(t)$ is the control input. The goal of this paper is to construct a control input $u(t)$ for system (1) so that the resulting trajectory $x(t)$ satisfies a formula in a temporal logic, such as the temporal logic LTL [16]. The formulas are built from a finite number of atomic propositions or observables which label areas of interest in the environment such as rooms or obstacles. Let $\Pi = \{\pi_1, \pi_2, \dots \pi_n\}$ be a set of such propositions. For system (1) we then associate an observation map

$$h_C : P \to \Pi \quad (2)$$

which maps the continuous states of the robot to the finite set of propositions.[1] Proposition $\pi_i \in \Pi$ represents an area of interest in the environment which can be characterized by a convex set of the form:

$$P_i = \{x \in \mathbb{R}^2 \mid \bigwedge_{1 \le k \le m} a_k^T x + b_k \le 0,\ a_k \in \mathbb{R}^2, b_k \in \mathbb{R}\}$$

In other words, the observation map $h_C : P \longrightarrow \Pi$ has the form $h_C(x) = \pi_i$ iff x belongs in the associated set P_i.

We first give some informal examples of LTL formulas and defer the formal syntax and semantics of LTL to Section III. Propositional logic is the traditional logic of *conjunction* ($\wedge$), *disjunction* ($\vee$), *negation* ($\neg$), *implication* ($\Rightarrow$), and *equivalence* ($\Leftrightarrow$). LTL is obtained from standard propositional logic by adding temporal operators such as *eventually* ($\Diamond$), *always* ($\Box$), *next* ($\bigcirc$) and *until* ($\mathcal{U}$). Some LTL examples that express interesting properties include:

- **Reach goal while avoiding obstacles:** The formula $\neg(o_1 \vee o_2 \vee \cdots \vee o_n)\mathcal{U}\pi$ expresses the property that eventually π will be true, and until π is reached, we must avoid all obstacles labeled as o_i, $i = 1, \dots, n$.
- **Sequencing:** The requirement that we must first visit π_1, π_2, and π_3 in this order is naturally captured by the formula $\Diamond(\pi_1 \wedge \Diamond(\pi_2 \wedge \Diamond\pi_3))$.
- **Coverage:** Formula $\Diamond\pi_1 \wedge \Diamond\pi_2 \wedge \cdots \wedge \Diamond\pi_m$ reads as the robot will eventually reach π_1 and eventually π_2 and ... eventually π_m, requiring the robot to eventually visit all regions of interest in any order.

More complicated specifications can be composed from more basic specifications using the logic operators. For such temporal logic formulas, in this paper we provide computational solution of the following problem.

[1]Uninteresting regions of the state space could be mapped to a dummy proposition or no proposition (resulting in a partial observation map). Furthermore, one can easily consider overlapping propositions resulting in non-deterministic observation maps.

Problem 1: [Temporal logic motion planning] Given robot model (1), observation map (2), initial condition $x(0) \in P$, and a LTL temporal logic formula φ, construct a control input $u(t)$ so that the resulting robot trajectory $x(t)$ satisfies the formula.

Example 1: In order to better explain the different steps in this paper, we will consider throughout the paper the following example. Consider a robot that is moving in a square environment with four areas of interest denoted by $\pi_1, \pi_2, \pi_3, \pi_4$. Initially, the robot is placed somewhere in the region labeled π_1 (see Figure 1). The desired specification for the robot given in natural language is: "Visit area π_2 then area π_3 then area π_4 and, finally, return to region π_1 while avoiding areas π_2 and π_3".

III. Linear Temporal Logic

In this section, we formally describe linear temporal logic (LTL) by giving its syntax and semantics.

Syntax: LTL formulas are interpreted over all trajectories of the system starting from some initial state $x(0)$ [16]. The atomic propositions of the logic are labels representing areas of interest in the environment such as rooms or obstacles. Let $\Pi = \{\pi_1, \pi_2, \dots\}$ be a set of such propositions. The LTL formulas are defined according to the following grammar:

$$\phi \quad ::= \quad \pi \mid \neg\phi \mid \phi \vee \phi \mid \phi\, \mathcal{U} \phi$$

As usual, the Boolean constants $\top$ and $\bot$ are defined as $\top = \pi \vee \neg\pi$ and $\bot = \neg\top$ respectively. Given negation ($\neg$) and disjunction ($\vee$), we can define conjunction ($\wedge$), implication ($\Rightarrow$), and equivalence ($\Leftrightarrow$). Furthermore, we can also derive additional temporal operators such as eventuality $\Diamond\phi = \top\mathcal{U}\phi$ and safety $\Box\phi = \neg\Diamond\neg\phi$. Note that our syntax does not contain the so-called next operator $\bigcirc\phi$.

Semantics: We define the continuous semantics of LTL formulas over robot trajectories. Let $x(t)$ for $t \ge 0$ denote the state of the robot at time t and let $x[t]$ be a possible robot trajectory starting at $x(t)$. That is $x[t] = \{x(s) \mid s \ge t \text{ and } \dot{x}(t) = u(t)\}$ or $x[t]$ denotes the flow of $x(s)$ under the input $u(s)$ for $s \ge t$.

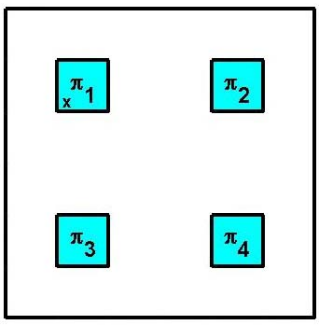

Fig. 1. Example 1. The 4 areas of interest and the initial position of the robot marked with x.

LTL formulas ϕ are interpreted over a trajectory $x[t]$. $x[t] \models_C \phi$ denotes the satisfaction of the formula ϕ over the trajectory $x[t]$ starting at $x(t)$. The semantics of any formula can be recursively defined as:

- $x[t] \models_C \pi$ iff $h_C(x(t)) = \pi$
- $x[t] \models_C \neg\phi$ if $x[t] \not\models_C \phi$
- $x[t] \models_C \phi_1 \vee \phi_2$ if $x[t] \models_C \phi_1$ or $x[t] \models_C \phi_2$
- $x[t] \models_C \phi_1 \mathcal{U} \phi_2$ if there exists $s \geq t$ such that $x[s] \models_C \phi_2$ and for all s' with $t \leq s' < s$ we have $x[s'] \models_C \phi_1$

Therefore, the formula $\phi_1 \mathcal{U} \phi_2$ intuitively expresses the property that over the trajectory $x[t]$ ϕ_1 is true until ϕ_2 becomes true. Formula $\Diamond\phi$ indicates that over the trajectory the formula ϕ becomes eventually true, whereas $\Box\phi$ indicates that ϕ is true over the trajectory $x[t]$ for all time $t' \geq t$.

Example 2: Coming back to Example (1), we can now formally define the specification using temporal logic formulas. Let π_i be the proposition that is true when the robot is in area i. Using LTL the precise specification is:

$$\phi = \Diamond(\pi_2 \wedge \Diamond(\pi_3 \wedge \Diamond(\pi_4 \wedge (\neg\pi_2 \wedge \neg\pi_3)\mathcal{U}\pi_1)))$$

IV. Temporal Logic Motion Planning

Our solution to generating continuous robot trajectories satisfying LTL formulas ϕ consists of the following three steps:

1) *Discrete Abstraction of Robot Motion:* Decompose the environment P into a finite number of equivalence classes resulting in a finite state model of robot motion.
2) *Temporal Logic Planning using Model Checking:* Construct plans for the discrete robot motion satisfying desired specifications using model checkers.
3) *Continuous Implementation of Discrete Plan:* Implement the discrete plan at the continuous level while preserving the satisfaction of the temporal formula.

A. Discrete Abstraction of Robot Motion

We first partition the workspace P of the robot into a finite number of equivalence classes (or cells). Clearly, we can use many efficient cell decomposition methods for polygonal environments [2]. In this paper, we chose to triangulate P for two main reasons. First, there exist several efficient triangulation algorithms which can partition very complicated environments [17]. Second, the choice of controllers used in Section IV-C is proven to exist and be efficiently computable on triangles [10]. Despite this choice, many of the results in this section can be easily adapted to similar decompositions, such as the decomposition described in [9].

Let $T : P \longrightarrow Q$ denote the map which sends each state $x \in P$ to the finite set $Q = \{q_1, \ldots, q_n\}$ of all equivalence classes (triangles in this paper). In other words, $T^{-1}(q)$ contains all states $x \in P$ which are contained in the triangle labeled by q and $\{T^{-1}(q_i) \mid q_i \in Q\}$ is a partition of the state space. Given such a partition of P, we can naturally abstract the robot motion by defining a finite transition system

$$D = (Q, q(0), \rightarrow_D, h_D) \tag{3}$$

where Q is the finite set of states, and $q(0) \in Q$ is the cell containing the initial robot state $x(0) \in P$, that is $q(0) = T(x(0))$. The dynamics are captured by the transition relation $\rightarrow_D \subseteq Q \times Q$, defined as $q_i \rightarrow_D q_j$ iff the cells labeled by q_i, q_j are topologically adjacent, that is triangles $T^{-1}(q_i)$ and $T^{-1}(q_j)$ have a common line segment. The transition relation $\rightarrow_D$ is also known as the dual graph of the triangulation and can be easily computed. Having defined transitions $\rightarrow_D$ for transition system D, we can define trajectories p of D as sequences of the form $p[i] = p_i \rightarrow_D p_{i+1} \rightarrow_D p_{i+2} \rightarrow_D \ldots$, where $p_i = p(i) \in Q$.

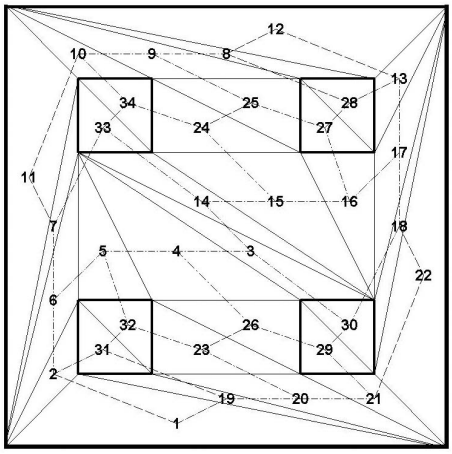

Fig. 2. The triangulation of the workspace of Example 1 appears with solid lines. The edges of the dual graph appear with dashes. The numbers denote the nodes of the undirected graph.

In addition to defining the transition relation, we also define the observation map $h_D : Q \longrightarrow \Pi$, as $h_D(q) = \pi$, if there exists $x \in T^{-1}(q)$ such that $h_C(x) = \pi$. In order to ensure that h_D is well defined, we must impose the requirement that the decomposition is proposition or *observation preserving*, that is for all $x_1, x_2 \in P$ and all $\pi \in \Pi$, $T(x_1) = T(x_2) \Rightarrow h_C(x_1) = h_C(x_2)$. In other words, states that belong in the same equivalence class or cell, map to the same observations.

Example 3: Revisiting Example 1, we can now triangulate the environment (see [18] for the algorithm used) and construct the dual graph of the triangulation (Figure 2). The resulting undirected graph has 34 states and 49 transitions.

The transition system D will serve as an abstract model of robot motion. We must now lift our problem formulation from the continuous to the discrete domain. In the previous section we defined the semantics of LTL formulas over continuous trajectories. We keep the LTL syntax exactly

the same, but we reformulate the semantics of the temporal logic formula to be interpreted over the discrete trajectories generated by transition system D.

Discrete LTL Semantics: Path formulas ϕ are interpreted over an execution $p[i]$, denoted as $p[i] \models_D \phi$. The semantics of any path formula can be recursively defined as:

- $p[i] \models_D \pi$ iff $h_D(p(i)) = \pi$
- $p[i] \models_D \neg\phi$ if $p[i] \not\models_D \phi$
- $p[i] \models_D \phi_1 \vee \phi_2$ if $p[i] \models_D \phi_1$ or $p[i] \models_D \phi_2$
- $p[i] \models_D \phi_1 \mathcal{U} \phi_2$ if there exists $j \geq i$ s. t. $p[j] \models_D \phi_2$, and for all j' with $i \leq j' < j$ we have $p[j'] \models_D \phi_1$

We are interested in understanding the relationship between the continuous robot model satisfying formula $x[0] \models_C \phi$ with continuous LTL semantics and the transition system D satisfying formula $p[0] \models_D \phi$, where $p(0) = T(x(0))$, but with the discrete LTL semantics.

B. Temporal Logic Planning using Model Checking

In a nutshell, model checking is the algorithmic procedure for testing whether a specification formula holds over some semantic model [19]. The model of the system is usually given in the form of a discrete transition system like the one described in Section IV-A. The specification formula is usually given in the form of temporal logics such as LTL.

As mentioned earlier, we are looking for computation paths $p[i]$ that satisfy the temporal formula $p[0] \models_D \phi$. In the model checking community, this is known as the generation of *witnesses*. Unfortunately, the current versions of the model checking software tools do not support the construction of witnesses as they are mainly analysis tools. Hence, we have to employ the algorithms that solve the dual problem, i.e. the generation of counterexamples. In this case, when the model checker determines that a formula ϕ is false, it constructs a finite trace $p[0]$ which demonstrates that the negation of ϕ is true, i.e. $p[0] \models_D \neg\phi$.

Let ϕ be the formula that the system should satisfy. Assume now that we give as input to our model checking algorithm the LTL formula $\neg\phi$, representing the negation of the desired behavior. If the formula is false in our discrete model of the environment, then the model checker will return a finite trace $p[0]$ that satisfies the formula $\neg(\neg\phi) \equiv \phi$ and, thus, we are done as we have found a finite path that satisfies the original LTL formula ϕ.

Out of the variety of model checking tools that have been developed over the years, we chose the most dominant ones, that is, NuSmv [5] which is based on symbolic model checking techniques and is mainly targeted for CTL (but it can also handle LTL) model checking problems, and Spin [4] which uses an automaton approach to the model checking problem and accepts only LTL formulas. Both toolboxes support hierarchy and composition, multiple agents, generation of counterexamples in case the temporal formula is invalidated and nondeterministic environments. Of course, there are also several differences between the two toolboxes mainly concerning the way they deal with the model checking problem, the user interface and the expressive power of the underlying logic. Spin only supports asynchronous communication among agents, but it gives us the option for the generation of traces that are optimal in the sense of minimum number of transitions (trace length). The conversion of the discrete transition system of Section IV-A to the input language of NuSmv or to the input language of SPIN is straightforward and it is automated.

Example 4: Using NuSmv for our example, we get the following witness trace p = {33, 34, 24, 25, 27, 16, 15, 14, 3, 4, 5, 32, 23, 26, 29, 30, 3, 14, 33}, which satisfies our specification.

C. Continuous Implementation of Discrete Trajectory

Our next task is to utilize the discrete trajectory $p[0]$ in order to construct a control input $u(t)$ for $t \geq 0$ and, therefore, a continuous trajectory $x[0]$ that satisfies exactly the same path formula. We achieve this desired goal by simulating (or implementing) at the continuous level each discrete transition of $p[0]$. This means that if the discrete system D makes a transition $p_i \rightarrow_D p_j$, then the continuous system must match this discrete step by moving the robot from states in triangle $T^{-1}(p_i)$ to states in triangle $T^{-1}(p_j)$.

We define a transition relation $\rightarrow_C \subset P \times P$ between continuous robot states in P. Formally, there is a transition $x \rightarrow_C x'$ if x and x' belong to adjacent triangles, and it is possible to construct a trajectory $x(t)$ for $0 \leq t \leq T$ with $x(0) = x$ and $x(T) = x'$, and, furthermore, for all $0 \leq t \leq T$ we have $x(t) \in (T^{-1}(T(x)) \cup T^{-1}(T(x')))$. Informally, $x \rightarrow_C x'$ if we can steer the robot from x to x' without visiting any triangle other than the triangle containing x or the neighboring triangle containing x'. Having defined $\rightarrow_C$ allows us to formally define a transition system $C = (P, x(0), \rightarrow_C, h_C)$.

In order to ensure that the continuous system can implement *any* discrete plan obtained by the model checker, we require that the decomposition of P satisfies the so called bisimulation property [11].

Definition 1 (Bisimulations): A partition $T : P \longrightarrow Q$ is called a bisimulation if the following properties hold for all $x, y \in P$:

- (Observation preserving) If $T(x) = T(y)$, then $h_C(x) = h_C(y)$
- (Reachability preserving) If $T(x) = T(y)$, then if $x \rightarrow_C x'$ then $y \rightarrow_C y'$ for some y' with $T(x') = T(y')$.

In other words, the triangulation is a bisimulation if the whole triangle is mapped to the same observation, and furthermore, if one state x can move to the adjacent triangle

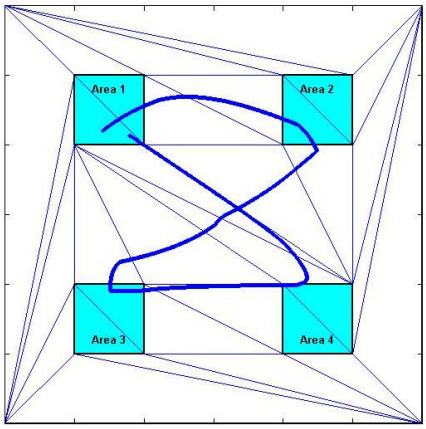

Fig. 3. Example 1: Continuous trajectory implementation

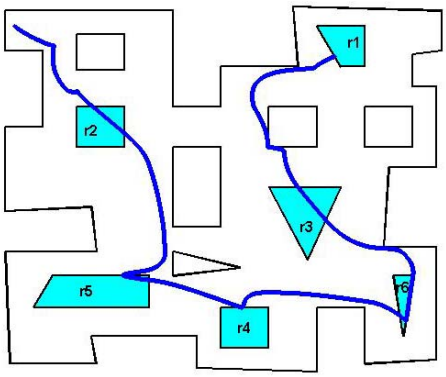

Fig. 4. Example 6: Visit all the rooms

to some state x', then all states y in the same triangle with x can also move to the same triangle with x'.

Assuming that this property is satisfied by the partition with respect to transitions we just defined, it is straightforward to show the following proposition.

Proposition 1: Let ϕ be an LTL path formula, and let $T : P \longrightarrow Q$ be a bisimulation. If $p[0] \models_D \phi$, then for every $x(0) \in T^{-1}(p(0))$ there exists a trajectory $x[0]$ satisfying $x[0] \models_C \phi$.

It remains to design controllers that satisfy the so-called bisimulation property. There are several recent approaches for generating such controllers, such as [9], [10] and [20]. We use the framework developed in [10] due to its computational properties in triangular environments. In this approach, an affine vector field is created in each triangle that drives the robot to the desired adjacent triangle, while taking into consideration any velocity bounds the robot might have. For a description of this controller design, we refer the reader to [10].

Note however, that by satisfying the bisimulation property using feedback controllers, the temporal logic formula is *robustly* satisfied not only by the initial state $x(0)$, but also by all other states in the same triangle. Furthermore, the design of the controllers in [10] can guarantee the continuity of the vector fields at the common edges of adjacent triangles.

Example 5: Figure 3 shows the continuous trajectory corresponding to our example, which was created using the triangulation from Example 3 and the discrete path generated by NuSMV in example 4.

V. SIMULATIONS

In order to test our approach to the problem of motion planning, we ran several simulations. We started with simple environments and continued by increasing the complexity of both the environment and the specification in order to make sure our approach scales well. In this section, we describe the process of creating a solution to the motion planning problem, and we show examples of non-trivial behaviors in complex environments, which may include holes and regions of interest.

The first step consists of specifying the environment. The environment is described as a set of vertices which define the outer contour, inner holes and inner regions of interest (such as rooms). We specify these vertices either by using a MATLAB based graphical user interface which allows the user to select points on a grid or by writing MATLAB functions that create vertices in a desired pattern. Next, we triangulate the polygonal environment using the software developed in [18] and we create the input code for the model checker which we augment with the temporal logic formula. The required path is generated as a counter example trace using a model checker. The final step is to create the control law $u(t)$ for $t \geq 0$ and to simulate the robot path. This step is performed in MATLAB and the control law is generated according to the method developed in [10] using linear programming.

Example 6: Figure 4 is an example of a trajectory, generated by NuSMV, satisfying a coverage requirement. In this example the desired behavior was to visit each of the rooms (shaded areas) in no particular order. The LTL formula that captures the specification is: $\diamond r_1 \wedge \diamond r_2 \wedge \diamond r_3 \wedge \diamond r_4 \wedge \diamond r_5 \wedge \diamond r_6$. For problems of this size, the generation of the discrete path is almost instant and the controller synthesis in MATLAB takes less then 15 seconds.

Example 7: This is an example of a trajectory satisfying a more complex requirement. In this example the desired behavior is "Visit room r_2, then room r_1 and then cover rooms r_3, r_4, r_5 - all this while avoiding obstacles o_1, o_2, o_3". Figure 5 depicts the path generated by SPIN.

Example 8: Figure 6 is an example of a very large environment. This environment includes 1156 areas of interest (rooms) and its discrete abstraction consists of 9250 triangles. The specification for this example was "Start in the white room and go to both black rooms". Even though this environment is very large, the computation time was a few seconds for the triangulation, about 55 seconds for the path generation in NuSMV and around 90 seconds for the

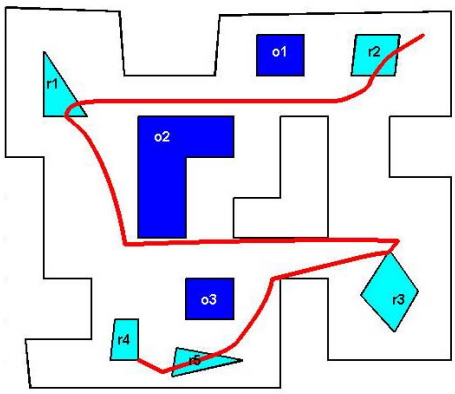

Fig. 5. Example 7: While avoiding the obstacles go to room 2, then to room 1 and then go to rooms 3, 4, 5 (in any order)

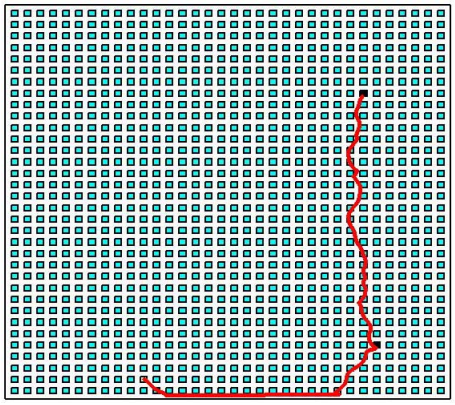

Fig. 6. Example 8: Complex environment - Visit the two square areas in black color

controller synthesis of a path of 145 triangles in MATLAB.

VI. Conclusions - Future work

In this paper, we have described our approach to the problem of motion planning, which begins at a high level of behavior specification, expressed in temporal logic, and ends in creating continuous control inputs for the robot that satisfy those requirements. We have shown that this approach is computationally feasible, that complex environments can be handled easily and that many complex robot behaviors can be expressed and satisfied.

We find this approach to be very promising and there are several directions in which we are planning to proceed, such as, extending this framework to multiple robots, incorporating natural language as a higher level specification (which will be automatically translated into temporal logic), and looking at different cell decomposition techniques.

Currently, we are investigating the extension of the presented approach to the design of hybrid controllers that would guarantee the satisfaction of a path formula ϕ in the presence of localization and actuation errors, in the presence of observable predicates (sensory input) and under a set R of initial conditions (i.e. $\forall x(0) \in R \Rightarrow x[0] \models_C \phi$). Furthermore, we plan to run experiments testing our approach using ACTIVMEDIA mobile robots as a testbed.

Acknowledgments

The authors would like to thank the anonymous reviewers for their comments.

References

[1] S. M. LaValle, "Planning Algorithms", [Online, Available at http://msl.cs.uiuc.edu/planning/], 2004

[2] H. Choset, K. M. Lynch, L. Kavraki, W. Burgard, S. A. Hutchinson, G. Kantor, and S. Thrun. *Robotic Motion Planning: Foundations and Implementation*. 2004. In preparation.

[3] M. Antoniotti and B. Mishra, "Discrete Event Models + Temporal logic = Supervisory Controller: Automatic Synthesis of Locomotion Controllers", *IEEE International Conference on Robotics and Automation*, 1995.

[4] G.J. Holzmann, "The Spin Model Checker Primer and Reference Manual", Addison-Wesley, Reading Massachusetts, 2004.

[5] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani and A. Tacchella, "NuSMV 2: An OpenSource Tool for Symbolic Model Checking", In Proceeding of *International Conference on Computer-Aided Verification* (CAV 2002), Copenhagen, Denmark, July 27-31, 2002.

[6] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso, "MBP: A Model Based Planner", *In Proc. IJCAI'01 Workshop on Planning under Uncertainty and Incomplete Information*, 2001.

[7] F. Bacchus and F. Kabanza, "Using Temporal Logics to Express Search Control Knowledge for Planning", *Artificial Intelligence*, vol 116, 2000.

[8] R.M. Jensen and M. M. Veloso, "OBDD-based Universal Planning for Synchronized Agents in Non-Deterministic Domains", *Journal of Artificial Intelligence Research*, 2000, Volume 13, 189-226.

[9] D.C. Conner, A. Rizzi, and H. Choset, "Composition of local potential functions for global robot control and navigation", *Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS 2003), IEEE, Vol. 4, October, 2003, pp. 3546-3551.

[10] C. Belta and L.C.G.J.M. Habets, "Constructing decidable hybrid systems with velocity bounds", *43rd IEEE Conference on Decision and Control*, Bahamas, Dec 2004.

[11] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas. "Discrete abstractions of hybrid systems", *Proceedings of the IEEE*, 88:971984, 2000.

[12] P. Tabuada and G. J. Pappas. Model checking LTL over controllable linear systems is decidable. *Hybrid Systems : Computation and Control*. volume 2623 of *Lecture Notes in Computer Science*. Springer-Verlag, Prague, 2003.

[13] S. Loizou and K. Kyriakopoulos, "Automatic Synthesis of Multi-Agent Motion Tasks Based on LTL Specifications", *43rd IEEE Conference on Decision and Control*, Bahamas, Dec 2004.

[14] E. Rimon and D. E. Kodischek, "Exact robot navigation using artificial potential functions", *IEEE Transactions on Robotics and Automation*, 8(5):501–518, 1992.

[15] M.M. Quottrup, T. Bak, and R. Izadi-Zamanabadi, "Multi-Robot Planning: A Timed Automata Approach", *Proc. 2004 IEEE Int. Conf. on Robotics and Automation*, New Orleans, LA.

[16] A. E. Emerson, "Temporal and Modal Logic", in: Van Leeuwen (ed) *Handbook of Theoretical Computer Science*, Vol. B, pp. 997-1072, Elsevier Science Publishers, 1990.

[17] M. de Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf, "Computational Geometry: Algorithms and Applications", 2nd rev. ed. 2000.

[18] A. Narkhede, and D. Manocha, "Fast Polygon Triangulation based on Seidel's Algorithm", [Online at http://www.cs.unc.edu/ dm/CODE/GEM/chapter.html#Seidel91].

[19] E. M. Clarke, O. Grumberg and D. A. Peled, "Model Checking", The MIT Press, Cambrige, MA, 1999.

[20] L.C.G.J.M. Habets and J.H. van Schuppen. A control problem for affine dynamical systems on a full-dimensional polytope. *Automatica*, 40:21–35, 2004.