Hybrid Controllers for Path Planning: A Temporal Logic Approach

Georgios E. Fainekos, Hadas Kress-Gazit, and George J. Pappas

Abstract— **Robot motion planning algorithms have focused on low-level reachability goals taking into account robot kinematics, or on high level task planning while ignoring low-level dynamics. In this paper, we present an integrated approach to the design of closed–loop hybrid controllers that guarantee by construction that the resulting continuous robot trajectories satisfy sophisticated specifications expressed in the so–called Linear Temporal Logic. In addition, our framework ensures that the temporal logic specification is satisfied even in the presence of an adversary that may instantaneously reposition the robot within the environment a finite number of times. This is achieved by obtaining a Büchi automaton realization of the temporal logic specification, which supervises a finite family of continuous feedback controllers, ensuring consistency between the discrete plan and the continuous execution.**

I. Introduction

One of the main challenges in robotics is the development of mathematical frameworks that formally and verifiably integrate high level planning with continuous control primitives. Traditionally, the path planning problem for mobile robots has considered reachability specifications of the form "move from the Initial position I to the Goal position G while staying within region R". The solutions to this well-studied problem span a wide variety of methods, from continuous (like potential or navigation functions [1]) to discrete (like Canny's algorithm, Voronoi diagrams, cell decompositions and probabilistic road maps [1], [2]).

Whereas these methods solve basic path planning problems, they do not address high level planning issues that arise when considering a number of goals or a particular ordering of them. In order to manage such constraints, one should either employ an existing high level planning method [2] or attack the problem using optimization techniques like mixed integer linear programming [3]. Even though the aforementioned methods can handle partial ordering of goals, they cannot deal with temporally extended goals. For such specifications, planning techniques [4], [5] that are based on model checking [6] seem more natural choices.

Using temporally extended goals, one would sacrifice some of the efficiency of the standard planning methods for expressiveness in the specifications. Temporal logics [6] such as the Linear Temporal Logic (LTL) and computation tree logic (CTL) have the expressive power to describe a conditional sequencing of goals under a well defined formal framework. Such a formal framework can provide us with the tools for automated controller synthesis and code generation.

Research is partially supported by the Army Research Office MURI Grant DAAD 19-02-01-0383 and NSF EHS 0311123

The authors are with the GRASP Laboratory, University of Pennsylvania, Philadelphia, PA 19104, USA. E-mail: {fainekos,hadaskg,pappasg}@grasp.cis.upenn.edu

The applicability of temporal logics in robotics was advocated as far back as [7]. In more recent works, the authors in [8] design controllers that satisfy LTL formulas by composing controllers using navigation functions. In [9], the UPPAAL model checking toolbox for timed automata has been applied to the multi-robot motion planning using CTL formulas, but without taking into account the dynamics of the robots. In our previous work [10], we have presented a framework for the design of an open–loop hybrid controller that generates continuous trajectories that satisfy temporal specifications in LTL. Related approaches to motion planning using hybrid or symbolic methods include the design of controllers that satisfy temporal logic specifications [11], the maneuver automata [12], the motion description language [13], the control and computation language [14] and the control quanta [15].

In this paper, we extend our previous work by designing closed-loop hybrid controllers that guarantee the generation of continuous robot trajectories that satisfy temporal specifications in an adversarial environment under reasonable assumptions. 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 [16], but we focus on triangular decompositions and general cellular partitions as these have been successfully applied to the basic path planning problem in [17] and [18] respectively. The partition results in a natural discrete abstraction of the robot motion which is used then for planning with automata theoretic methods [5].

In particular, we use the automaton, which captures the temporal specification (LTL formula), to enforce a sequencing on the possible moves of the robot in the discrete abstraction of the environment. Out of all the possible discrete trajectories that satisfy the specification, we choose the shortest one. In order to ensure that the discrete plan is feasible at the continuous level, the decomposition must satisfy the so-called bisimulation property [19]. The feedback controllers that are presented in [17] and [18] satisfy this property. 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.

In real-life applications, the sequential composition of controllers (open-loop hybrid controller) might fail due to localization errors. This is especially true when the environment is partitioned into cells. The main contribution of this paper is the design of a closed-loop (at the specification level) hybrid controller that generates a new sequence of controllers every time the system moves out of the operational range of the initial open-loop hybrid controller.

II. Problem Formulation

We consider a fully actuated, planar model of robot motion operating in a polygonal environment P. The environment P may have holes, but these must be enclosed by a single polygonal chain that does not intersect itself. The motion of the robot is expressed as:

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

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 closed-loop hybrid controller that generates control inputs $u(t)$ for system (1) so that the resulting trajectory $x(t)$ for a set of initial conditions X_0 satisfies a formula–specification ϕ in the linear temporal logic LTL$_{-X}$ [6] in an adversarial environment. By saying adversarial environment, we mean the following:

Definition 1 (Adversary): We assume the existence of an adversary with the following properties. The adversary is allowed to reposition the robot only: (i) a finite number of times, (ii) within the connected workspace P and (iii) to positions that do not falsify the specification ϕ.

In our framework, temporal formulas are built upon a finite number of atomic propositions which label areas of interest in the environment (rooms, obstacles, buildings, etc). Let $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ be a set of such propositions. For system (1) we then associate an observation map

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

which maps the continuous states of the robot to the finite set of propositions. Note that regions of the state space of no interest to the user are mapped to a dummy proposition. Even though one can easily consider overlapping propositions resulting in non-deterministic observation maps, here we consider only sets of disjoint atomic propositions. Each proposition $\pi_i \in \Pi$ represents an area of interest in the environment which can be characterised by a convex set of the form:

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

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.

In order to make more apparent the use of LTL$_{-X}$ for the composition of temporal specifications, we first give an informal description of the traditional and temporal operators. The formal syntax and semantics of LTL$_{-X}$ are presented in Section IV. LTL$_{-X}$ contains the traditional logic operators of *conjunction* ($\wedge$), *disjunction* ($\vee$), *negation* ($\neg$), *implication* ($\Rightarrow$), and *equivalence* ($\Leftrightarrow$). The main temporal operators are usually *eventually* ($\Diamond$), *always* ($\Box$) and *until* ($\mathcal{U}$). Some LTL formulas that express interesting properties in the context of robot motion planning are the following. **Recurrence:** the formula $\Box(\Diamond\pi_1 \wedge \Diamond\pi_2 \wedge \cdots \wedge \Diamond\pi_m)$ requires that the robot visits the areas π_1 to π_m infinitely often without any particular ordering. **Fairness properties:** *"whenever the robot visits π_1, it should also eventually visit area π_2"* as $\Box(\pi_1 \rightarrow \Diamond\pi_2)$. Some further examples can be found in [10]. For such temporal logic formulas, we provide a computational solution to the following problem.

Problem 1 (Temporal Logic Motion Planning): Given robot model (1), observation map (2), a set of initial conditions $X_0 \subseteq P$ and an LTL$_{-X}$ temporal logic formula ϕ, construct a hybrid controller $H(x, t, \phi)$ so that the control input is $u(t) = H(x, t, \phi)$ and the resulting robot trajectory $x(t)$ satisfies the formula ϕ in the adversarial environment of Definition 1.

III. Discrete Abstraction of Robot Motion

In order to use discrete logics to reason about continuous systems, we need a finite partition of the continuous state space P. Clearly, we can use many efficient cell decomposition methods for polygonal environments [1]. In this paper, we follow the approach presented in [17] and [10], that is we chose to triangulate P and create a finite number of equivalence classes (each triangle). This choice was mainly made for two reasons. First, there exist several efficient triangulation algorithms which can partition complicated polygonal environments [16]. Second, the choice of controllers used in Section V is proven to exist and be efficiently computable on triangles [17]. Despite this choice, the results in this paper can be easily adapted to similar decompositions, such as the decomposition described in [18].

In the following paragraphs, we present how the undirected graph resulting from the triangulation of the polygonal environment can be converted to a finite transition system that serves as an abstract model of the robot motion. 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 cell labelled by q, and $\{T^{-1}(q) \mid q \in Q\}$ 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.

Definition 2 (FTS): A Finite Transition System is a tuple $D = (Q, Q_0, \rightarrow_D, h_D, \Pi)$ where:

- Q is the finite set of states
- $Q_0 \subseteq Q$ is the set of the possible initial robot states in the planar environment
- $\rightarrow_D \subseteq Q \times Q$ captures the dynamics of the system and it is defined as $q_i \rightarrow_D q_j$ iff the cells labelled by q_i, q_j are topologically adjacent, i.e. cells $T^{-1}(q_i)$ and $T^{-1}(q_j)$ share a common edge
- $h_D : Q \longrightarrow \Pi$ is the observation map defined as $h_D(q) = \pi$, if there exists $x \in T^{-1}(q)$ such that $h_C(x) = \pi$
- Π is the set of propositions defined in Section II

An infinite sequence of states $p = p_0, p_1, p_2, \ldots$ such that $p_0 \in Q_0$, $\forall k. p_k \in Q$ and $p_i \rightarrow_D p_{i+1}$ is called an *execution*, whereas the infinite sequence $tr(p) = h_D(p_0), h_D(p_1), h_D(p_2), \ldots$ is called a *trace* of the transition system D. Let $p[i]$ denote the suffix of the execution p that starts from state p_i, i.e. $p[i] = p_i, p_{i+1}, p_{i+2}, \ldots$ and

$p = p[0]$. The language of D, i.e. $L(D)$, is the set of all possible traces.

In order to ensure that the observation map h_D is well defined, we must impose the requirement that the decomposition is proposition or *observation preserving*, that is for all $x_i, x_j \in P$,

$$T(x_i) = T(x_j) \Rightarrow h_C(x_i) = h_C(x_j)$$

In other words, states that belong in the same equivalence class or cell, map to the same observations.

IV. The Linear Temporal Logic LTL$_{-X}$

Logic can be a powerful tool for describing concretely formal statements and, more importantly, for reasoning over them. Temporal logics can provide us with even more flexibility as they allow reasoning over time. In the context of path planning for a single robot, we present and use a subclass of temporal logics commonly known as linear temporal logic LTL$_{-X}$ [6].

A. LTL$_{-X}$ Syntax

LTL is a temporal logic whose syntax contains path formulas, i.e. the specification that the temporal formula describes is validated over a trajectory (or trace) of the robot (discrete system). Usually, these traces have infinite length as temporal formulas may describe non–terminating properties. As mentioned earlier, the atomic propositions Π of the logic are labels representing cells in the environment. The LTL$_{-X}$ formulas are defined according to the following grammar:

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

where ϕ is a path formula, $\pi \in \Pi$ and $\mathcal{U}$ is the (strong) until operator. 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 $\Diamond\phi = \top \mathcal{U} \phi$
- Safety $\Box\phi = \phi \mathcal{U} \bot$

B. LTL$_{-X}$ Continuous Semantics

We define the continuous semantics of LTL$_{-X}$ formulas over the robot continuous trajectories. Let $x(t)$ for $t \geq 0$ denote the state of the system (robot) at time t. Let $x[t]$ denote the flow of $x(s)$ under the input $u(s)$ for $s \geq t$ and $x[t, t']$ denote the flow of $x(s)$ under the input $u(s)$ for $t \leq s \leq t'$.

The formulas ϕ are interpreted over a trajectory $x[t]$ of the system. $x[t] \models_C \phi$ denotes the satisfaction of the formula ϕ over the trajectory $x[t]$. 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 path 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. The formula $\Diamond\phi$ indicates that over the trajectory $x[t]$ the subformula ϕ becomes eventually true, whereas $\Box\phi$ indicates that ϕ is always true over $x[t]$.

C. LTL$_{-X}$ Discrete Semantics

Let $p[i]$ be an execution of the discrete transition system D starting from state $p_i \in Q$. Hence, $p[0]$ is an execution of D starting at state $p_0 \in Q_0$. 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 $j \geq i$ such that $p[j] \models_D \phi_2$, and for all j' with $i \leq j' < j$ we have $p[j'] \models_D \phi_1$

The relationship between the continuous LTL semantics ($x[0] \models_C \phi$) and the discrete LTL semantics ($p[0] \models_D \phi$ for $p_0 = T(x(0))$) was discussed in Section IV-C of our previous work [10].

D. From LTL to Büchi Automata

It has been proven that any LTL formula can be converted to an equivalent Büchi automaton (for a discussion see [6]). A Büchi automaton differs from the usual notion of automata in that it accepts infinite traces. The conversion from LTL formulas to Büchi automata is a well studied problem that has resulted in many efficient algorithms.

Definition 3 (Büchi automaton): A Büchi automaton is a tuple $B = (S, S_0, \Sigma, \rightarrow_B, F)$ where:

- S is a finite set of states
- S_0 is the set of the possible initial states
- Σ is the input alphabet of the automaton
- $\rightarrow_B \subseteq S \times \Sigma \times 2^S$ is a nondeterministic transition function
- $F \subseteq S$ is the set of accepting states

In our case, the input alphabet Σ of the automaton B is the same as the set of propositions Π of the finite transition system D ($\Sigma = \Pi$). An *infinite word* w is a member of Σ^ω, which means that we concatenate an infinite number of symbols from Σ (i.e. $w = w_0, w_1, w_2, \ldots$ with $w_i \in \Sigma$). A *run* r of B is the sequence of states $r = r_0, r_1, r_2, \ldots$ with $r_i \in S$ that occurs under the input word w. Let $lim(\cdot)$ be the function that returns the set of states that are encountered infinitely often in the run r of B. The language of B, i.e. $L(B)$, consists of all the input words that have a run that is accepted by B.

Definition 4 (Büchi acceptance): A Büchi automaton B accepts an infinite input word w iff the run $r = r_0, r_1, r_2, \ldots$ such that $r_i \xrightarrow{w_i}_B r_{i+1}$ with $r_0 \in S_0$, $r_i \in S$ and $w_i \in \Sigma$, satisfies the relationship $lim(r) \cap F \neq \emptyset$.

Finding the existence of accepting runs is an easy problem. First, we convert the Büchi automaton to a directed graph and, then, we find the strongly connected components (SCC) in the graph. If at least one SCC that contains a final state ($s \in F$) is reachable from some state in the set of initial

states S_0, then the language $L(B)$ of B is non-empty (for more details see [20]).

V. Open-loop hybrid controller

The design of the open-loop hybrid controller consists of two parts. First, the creation of a trajectory on the discrete abstraction of the robot workspace (presented in this section), and second, the conversion of the latter trajectory to a continuous path for the robot (see Section IV-C in [10]).

First, we extend the finite transition system D, which models the basic structure of the environment, with a dummy state labelled as "Start" that has a transition to every other state in the set of initial states Q_0. The addition of this dummy state is necessary in the case that some initial state of D already satisfies partially the temporal specification.

Let D' be the extended finite transition system, then $D' = (Q', Q'_0, \rightarrow_{D'}, h_{D'}, \Pi)$ where:

- $Q' = Q \cup \{Start\}$
- $Q'_0 = \{Start\}$
- $\rightarrow_{D'} = \rightarrow_D \cup \rightarrow_d$ where $\rightarrow_d$ is defined as $Start \rightarrow_d q_j$ for all $q_j \in Q_0$
- $h_{D'} : Q' \longrightarrow \Pi$ is the same observation map as h_D but extended by mapping the "Start" state to the dummy observation

In the context of formula satisfiability, we can use the finite transition systems D and D' interchangeably.

Proposition 1: p is an execution of D iff $p' = \{Start\}, p$ is an execution of D'. Also, $p[0] \models_D \phi$ iff $p'[1] \models_{D'} \phi$.

Now, consider the automaton A that derives from the product of the finite transition system D' that describes the dynamics of our system and the Büchi automaton B that represents the temporal logic specification (formula ϕ). The design of an open-loop hybrid controller for motion planning using temporal logic specifications reduces to the problem of finding the accepting executions of the automaton A. Informally, the Büchi automaton B restricts the behaviour of the system D' by permitting only certain acceptable transitions.

Definition 5 (Product automaton): The product automaton A of D' and B ($A = D' \times B$) that accepts finite executions is a tuple $A = (S_A, S_{A0}, Q, \rightarrow_A, F_A)$ where:

- $S_A = Q' \times S$
- $S_{A0} = Q'_0 \times S_0$
- $\rightarrow_A \subseteq S_A \times Q \times 2^{S_A}$ such that $(q_i, s_j) \xrightarrow{q_{i'}}_A (q_{i'}, s_{j'})$ iff $q_i \rightarrow_{D'} q_{i'}$ and $s_j \overset{h_{D'}(q_{i'})}{\longrightarrow}_B s_{j'}$
- $F_A = Q' \times F$ is the set of accepting states

The automaton A permits a transition to a state $(q_{i'}, s_{j'})$ iff the finite transition system D' can take a transition to state $q_{i'}$ and the automaton B has a transition with an input symbol that is the observation of state $q_{i'}$. That is, we allow a transition if and only if we can observe an acceptable proposition $h_{D'}(q_{i'})$ at the next state.

Notice however that the strongly connected components of automaton A can be singletons with nodes without outgoing transitions, for example in the case where the temporal formula is $\Diamond\Box\pi$. In order to use the Büchi acceptance definition for automaton A, we need to extend the definition of automaton A so as its language consists only of infinite accepting executions. For this reason, we use the stutter extension rule [20], that is, we add on the blocking states a self transition. Let r be a run of automaton A, then we define the projection functions $pr_1 : 2^{S_A} \longrightarrow 2^Q$ and $pr_2 : 2^{S_A} \longrightarrow 2^S$ such that if $r = (q_{i_0}, s_{j_0})(q_{i_1}, s_{j_1})(q_{i_2}, s_{j_2}) \dots$ then $pr_1(r) = q_{i_0} q_{i_1} q_{i_2} \dots$ and similarly for function $pr_2(r) = s_{j_0} s_{j_1} s_{j_2} \dots$.

Definition 6 (Stutter Extension): For all the states $s \in S_A$, if there do not exist some $s' \in S_A$ and $q \in Q$ such that $s \xrightarrow{q}_A s'$, then $\rightarrow_A = \rightarrow_A \cup (s \overset{pr_1(s)}{\longrightarrow}_A s)$.

By construction, the following theorem is satisfied.

Theorem 1 (Adapted from [5]): An execution p of FTS D' that satisfies the specification ϕ exists iff the language of A is non-empty ($L(A) \neq \emptyset$).

The non-emptiness problem of the language $L(A)$ can be solved as described in Section IV-D. Due to the fact that the robot is fully actuated, the structure of the FTS D' is such that all the states in D' are reachable (as long as there do not exist any disconnected areas). Hence, the language $L(A)$ can only be empty in the case that there exist logical inconsistencies in the temporal logic formula ϕ.

Corollary 1: A word w in the language $L(A)$ is an execution of D that satisfies the temporal specification ϕ, that is, $w[0] \models_D \phi$.

For theoretical bounds on the complexity of the above problem as well as for extensions on planning under partial observability see the work of Giacomo and Vardi [5].

Algorithm 1 The Open-Loop Hybrid Controller

```
1: procedure OpenLoopController(P, φ, x_0)
2:     Δ ← Triangulate(P)
3:     D' ← TriangulationToFTS(Δ)
4:     B ← LTLtoBuechi(φ)
5:     A ← Product(D', B)
6:     SCC_A ← StronglyConnectedComponents(A)
7:     return Controllers(A, SCC_A, Δ, (T(x_0), s_0))
8: end procedure
```

Algorithm 2 Subroutine called by the Open-Loop and Closed-Loop Hybrid Controller Algorithms

```
1: procedure Controllers(A, SCC_A, Δ, (q_0, s_0)))
2:     Plans ← BreadthFirstSearch(A, (q_0, s_0))
3:     r ← ShortestPathToFinalSCC(Plans, SCC_A)
4:     if r is empty then return false
5:     else return GenerateControllers(r, Δ, q_0)
6:     end if
7: end procedure
```

VI. Closed-Loop Hybrid Controller

In real life situations, though, an open-loop hybrid controller at the level of specification cannot guarantee the completion of the task. For example, we could have insufficient sampling frequency which leads to unobservable

events (missed triangle transitions) and sensor and actuation noise. In this, paper we are not interested in the low level issues involved in the previous examples, but in the high level planning. Hence, we consider the existence of an adversary with the properties described in Section II.

Even though there exist general planners based on symbolic methods [4] and on the manipulation of the temporal formulas explicitly [21], they are not well suited for the specific structure of our problem. The typical planners generate plans that are either a sequence of actions (if there do not exist any observable predicates) or a tree of actions (in the case of conditional planning) that guarantee the achievement of the goal. They can also model non–determinism, in the sense that the outcome of an action may result in many different states, and/or partial observability, which occurs when the high-level controller cannot distinguish between different states of the system unless it performs a set of "sensing" actions. The nature of the planning problem that we consider here cannot be handled by the aforementioned planning methods. We do not care about partial observability and, also, the fact that the adversary can teleport the robot to any position cannot be modeled by adding non-determinism in the discrete model of the robot motion.

The answer to Problem 1 lies in the automata construction of Section V. The only additional mechanism that is required is tracking which parts of the temporal specification have been completed. This is an easy thing to do on the product automaton A by monitoring the current state of the Büchi automaton B. The supervisory controller monitors the continuous trajectory of the robot and verifies whether it follows the proper execution of the discrete path. In case it doesn't, it halts the robot and checks whether the trajectory has violated the temporal logic specification. If it has not done any violation, it generates a new discrete trajectory and monitors the resulting continuous path. The high level description of the closed-loop hybrid controller is presented in Algorithm 3. The Algorithms 1 and 2 need to be modified so as to return the run r of automaton A and additional data structures in order to be used in Algorithm 3. The Proposition 2 below is a straightforward corollary of Proposition 1 in [10], the if-check at line 6 in Algorithm 3 and the if-check at line 4 in Algorithm 2 under the following set of assumptions.

Assumptions 1: (i) The system (1) is operating within a connected workspace P. (ii) The specification ϕ is not a tautology or unsatisfiable. (iii) $\{C_i\}_m = C_1, C_2, \ldots, C_m$ is the finite sequence of controllers (open-loop hybrid controller $H(x, t, \phi)$) that was generated using the open-loop algorithm for the temporal specification ϕ. Without loss of generality, C_m is a controller that halts the robot at the centroid of the triangle.

Proposition 2 (Main Loop Invariance): If the set of Assumptions 1 holds, then at line 3 of Algorithm 3 the temporal specification ϕ is not false.

Remark 1: In terms of implementation, the check that the temporal specification ϕ has been violated can be easily done by maintaining a list for each state s' of the Büchi automaton

Algorithm 3 The Closed-Loop Hybrid Controller

```
1:  procedure ClosedLoopController(P, φ, x_0)
2:      [{C_i}, r, ...] ← OpenLoopController(P, φ, x_0)
3:      i ← 1 and m ← |r|
4:      while i ≤ m do
5:          while T(x) = pr_1(r_i) do
6:              x ← ApplyController(C_i, x)
7:          end while
8:          if T(x) = pr_1(r_{i+1}) then i ← i + 1
9:          else
10:             if φ is violated then return false
11:             else
12:                 if pr_2(r_i) ≠ pr_2(r_{i+1}) and h_C(x) =
                        h_D(pr_1(r_{i+1})) then s ← pr_2(r_{i+1})
13:                 else s ← pr_2(r_i)
14:                 end if
15:                 [{C_i}, r] ← Controllers(..., (T(x), s))
16:                 i ← 1 and m ← |r|
17:             end if
18:         end if
19:     end while
20: end procedure
```

B with the states q of the FTS D' for which

$$s \xrightarrow{\neg h_{D'}(q)}_B s'$$

The following proposition is the main result of this paper (the proof is omitted due to space limitations, but it is a straightforward case by case analysis):

Proposition 3 (Termination): If the set Assumptions 1 and Definition 1 hold, then Algorithm 3 terminates with a trajectory $x_f = \bigcup_{j=1}^{k} x_{j-1}[t_{j-1}, t_j] \cup x_k[t_k]$ for some $k \in \mathbb{N}$ and for $x_j = H_j(x_j, t, \phi)$ such that $x_f[0] \models_C \phi$.

VII. Implementation and Simulations

For the conversion of the LTL formulas to Büchi automata we use the algorithm that is available at [http://www.liafa.jussieu.fr/~oddoux/], and for the triangulation of the environment we use the code available at [http://www.cs.unc.edu/~dm/CODE/GEM/chapter.html]. The rest of the code was developed in house using the MATLAB programming platform.

Example 1 (Simulation results): In this simulation, a polygonal environment with holes was created, as depicted in Fig. 1. This environment contains 4 areas, highlighted in the figure, that are to be visited by the robot. The formula specifying the coverage requirement of visiting all areas is $\phi = \Diamond area_1 \wedge \Diamond area_2 \wedge \Diamond area_3 \wedge \Diamond area_4$.

In order to create the hybrid controller, first the environment was triangulated and abstracted to a finite transition system (FTS) containing 98 states (triangles). Next, the specification formula was translated into a Büchi automaton containing 16 states, and finally, the supervisory controller was created by taking the product of the FTS and the Büchi automaton (1569 states).

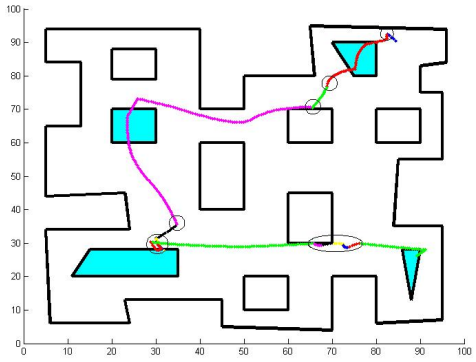

Fig. 1. Trajectory created by the close-loop hybrid controller. Each divergence from the planned discrete path is circled.

Fig. 1 shows the trajectory generated for the specification ϕ by the close-loop hybrid controller. In this case, the sampling interval has been intentionally set too long allowing many discrete transitions to be missed. For each such divergence, which is circled in the figure, the close-loop hybrid controller generates a new continuous trajectory that completes the temporally extended goal without repeating the whole specification. We would like to point out that each such re-planning usually takes place in the areas of the workspace where there is a high number of triangles (not visible in this figure).

VIII. Conclusions – Discussion

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 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.

Most of the assumptions we have made in this paper (a powerful adversary, the structure of the environment and the terminal controller C_m) can be easily overcome (not explained here due to space limitations). Note that the approach presented here can be also used in the case where obstacles enter dynamically in the environment making some discrete transitions unavailable (similarly to the D^* algorithm [1]). Also, we would like to point out that the off-line design of a single hybrid controller that can provide feedback both at the low and the high (specification) levels is possible.

We are currently extending the results presented in this paper in order to reason on how robustly is the temporal formula satisfied by the continuous robot trajectory. Another direction of on-going research is the implementation and application of the above algorithms to our UGV experimental test–beds. Finally, we are investigating the concurrency issues involved in the transition of the current framework to the multi–agent case.

Acknowledgments: The first author would like to thank Stanislav Angelov and Boulos Harb for the many useful discussions.

References

[1] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms and Implementations*. MIT Press, March 2005.

[2] S. M. LaValle, *Planning Algorithms*, 2005. [Online]. Available: http://msl.cs.uiuc.edu/planning/

[3] A. Richards and J. P. How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," in *Proceedings of the 2002 IEEE American Control Conference*, May 2002, pp. 1936–1941.

[4] F. Giunchiglia and P. Traverso, "Planning as model checking," in *Proceedings of the 5th European Conference on Planning, LNCS*, vol. 1809, 1999, pp. 1–20.

[5] G. D. Giacomo and M. Y. Vardi, "Automata-theoretic approach to planning for temporally extended goals," in *Proceedings of the 5th European Conference on Planning, LNCS*, vol. 1809, 1999, pp. 226–238.

[6] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, Massachusetts: MIT Press, 1999.

[7] M. Antoniotti and B. Mishra, "Discrete event models + temporal logic = supervisory controller: Automatic synthesis of locomotion controllers," in *Proceedings of the 1995 IEEE International Conference on Robotics and Automation*, May 1995.

[8] S. G. Loizou and K. J. Kyriakopoulos, "Automatic synthesis of multi-agent motion tasks based on ltl specifications," in *Proceedings of the 43rd IEEE Conference on Decision and Control*, Dec. 2004.

[9] M. M. Quottrup, T. Bak, and R. Izadi-Zamanabadi, "Multi-robot planning: A timed automata approach," in *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, New Orleasn, LA, April 2004, pp. 4417–4422.

[10] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for mobile robots," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, April 2005.

[11] P. Tabuada and G. J. Pappas, "From discrete specifications to hybrid control," in *Proceedings of the 42nd IEEE Conference on Decision and Control*, December 2003.

[12] E. Frazzoli, "Robust hybrid control for autonomous vehicle motion planning," Ph.D. dissertation, Massachusetts Institute of Technology, May 2001.

[13] D. Hristu-Varsakelis, M. Egerstedt, and P. S. Krishnaprasad, "On the complexity of the motion description language mdle," in *Proceedings of the 42nd IEEE Conference on Decision and Control*, December 2003, pp. 3360–3365.

[14] E. Klavins, "A language for modeling and programming cooperative control systems," in *Proceedings of the International Conference on Robotics and Automation*, New Orleasn, LA, April 2004.

[15] S. Pancanti, L. Pallottino, D. Salvadorini, and A. Bicchi, "Motion planning through symbols and lattices," in *Proceedings of the International Conference on Robotics and Automation*, New Orleasn, LA, April 2004, pp. 3914–3919.

[16] M. de Berg, O. Schwarzkopf, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 2nd ed. Springer-Verlag, 2000.

[17] C. Belta, V. Isler, and G. J. Pappas, "Discrete abstractions for robot motion planning and control," *IEEE Transactions on Robotics*, Accepted for publication.

[18] D. C. Conner, A. Rizzi, and H. Choset, "Construction and automated deployment of local potential functions for global robot control and navigation," Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-03-22, November 2003.

[19] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, "Discrete abstractions of hybrid systems," *Proceedings of the IEEE*, vol. 88, no. 2, pp. 971–984, 2000.

[20] G. Holzmann, *The Spin Model Checker, Primer and Reference Manual*. Reading, Massachusetts: Addison-Wesley, 2004.

[21] F. Bacchus and F. Kabanza, "Using temporal logics to express search control knowledge for planning," *Artif. Intell.*, vol. 116, 2000.