8-23-2008

Recycling controllers

Hadas Kress-Gazit
University of Pennsylvania, hadaskg@grasp.upenn.edu

Nora Ayanian
University of Pennsylvania, nayanian@grasp.upenn.edu

George J. Pappas
University of Pennsylvania, pappasg@seas.upenn.edu

Vijay Kumar
University of Pennsylvania, kumar@grasp.upenn.edu

Recycling controllers

Abstract
The problem of designing control schemes for teams of robots to satisfy complex high-level tasks is a challenging problem which becomes more difficult when adding constraints on relative locations of robots. This paper presents a method for automatically creating hybrid controllers that ensure a team of heterogeneous robots satisfy some user specified high-level task while guaranteeing collision avoidance and predicting and reducing deadlock. The generated hybrid controller composes atomic controllers based on information the robots gather during runtime; thus these atomic controllers can be reused in different scenarios for multiple tasks. As a demonstration of this general approach we examine a task in which a group of robots sort different items to be recycled.

Keywords
atomic controller, collision avoidance, control scheme design, deadlock reduction, discrete automaton, heterogeneous robot team, recycling controller, automata theory, collision avoidance, control system synthesis, mobile robots, multi-robot systems

Comments

Kress-Gazit, H.; Ayanian, N.; Pappas, G.J.; Kumar, V., "Recycling controllers," Automation Science and Engineering, 2008. CASE 2008. IEEE International Conference on , vol., no., pp.772-777, 23-26 Aug. 2008
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4626521&isnumber=4626395

4th IEEE Conference on Automation Science and Engineering
Key Bridge Marriott, Washington DC, USA
August 23-26, 2008

Recycling Controllers

Hadas Kress-Gazit, Nora Ayanian, George J. Pappas, Vijay Kumar*
GRASP Laboratory, University of Pennsylvania
Philadelphia, PA 19104, USA
{hadaskg,nayanian,pappasg,kumar}@grasp.upenn.edu

Abstract— **The problem of designing control schemes for teams of robots to satisfy complex high-level tasks is a challenging problem which becomes more difficult when adding constraints on relative locations of robots. This paper presents a method for automatically creating hybrid controllers that ensure a team of heterogeneous robots satisfy some user specified high-level task while guaranteeing collision avoidance and predicting and reducing deadlock. The generated hybrid controller composes atomic controllers based on information the robots gather during runtime; thus these atomic controllers can be reused in different scenarios for multiple tasks. As a demonstration of this general approach we examine a task in which a group of robots sort different items to be recycled.**

I. Introduction

Robot swarms and self-driving cars are no longer a distant dream. As systems' abilities and complexity increase, it becomes infeasible for a system designer to take care of every detail. Currently, behaviors of such systems are most often hard coded and changing them is time consuming, error prone, and requires expert knowledge. A significant challenge for the automation community is creating methods that facilitate "programming" of complex systems by allowing behavior specification at a high-level and *automatically* generating or adjusting the system such that it satisfies the new behavior while providing guarantees of correctness.

One can distinguish between reactive and non-reactive high-level system behaviors. Non-reactive or open-loop behaviors are predefined and do not change no matter what occurs at a high level in the environment. They include robot formations driving to goal positions and complex aerial maneuvering of UAVs [2], [17]. These systems must react to low-level disturbances but high-level behavior remains the same. Reactive behaviors, in contrast, may cause the system to behave differently depending on sensory information gathered. Such behaviors are useful, for example, in robotic search and rescue missions, where hazards may alter the robots' behavior, or a computer game in which the computer must adjust its player's behavior based on actions of other players. Creating a system controller for non-reactive behavior [2] that is correct and robust at the low-level is nontrivial [4]; simultaneously addressing reactive behaviors adds additional complexity to the decision making process.

When dealing with physical systems such as teams of robots working together, high-level planning and behavioral issues, reactive or not, are coupled with the challenges of low-level continuous control. The latter must guarantee *safety*: for example, drive the robots while guaranteeing they do not collide with obstacles, humans, or each other. Furthermore, the low-level control must provide a solution for global system issues which cannot be solved in a *natural* way at the high-level, such as liveness requirements (for multiple agents, ensuring deadlocks are avoided when possible[1]).

In this paper we propose a method for controlling a team of robots, addressing both high-level planning and low-level control challenges. The team accomplishes a user defined *reactive* high-level task (such as sorting), if feasible, while providing *global guarantees for collision and deadlock avoidance*. After specifying the high-level task, workspace, number of robots, and robot proximity constraints, a hybrid controller is automatically generated such that the team is guaranteed to accomplish the task, under some assumptions.

The novelty of this work is in combining provably-correct, high-level planning techniques for multi-robot tasks with reactive behaviors that satisfy user-specified constraints such as proximity between robots and safety. This results in a hybrid system that allows automatic generation of provably correct robot control from a high-level description. Moreover, as illustrated in Section V the method is flexible, allowing several behaviors to be created easily and quickly.

The method we propose builds on the work in [10], where a high-level reactive task intended for a single robot was captured using a linear temporal logic formula (LTL) [6]. This formula was then synthesized into a hybrid controller that guaranteed the robot behaved as desired under certain assumptions. That work was later extended to handle multi-robot scenarios in a decentralized way [8]. While this method scales well with the number of robots, it cannot provide collision avoidance in a *natural* way. It is achieved by enforcing contrived constraints such as two robots cannot be in the same room concurrently (mutual exclusion), or adding sensor inputs that alert robots when too close [8]; in both cases, deadlock may occur. Additionally, this method does not naturally allow for pairwise constraints on robots, such as staying within a specified distance.

Here, following [9], [10], we decompose the workspace into convex regions. Then, based on the decomposition and the robots' sensors and actions, we write the high-level specification as a set of Structured English sentences. These sentences are automatically translated into an LTL formula and synthesized into an automaton such that every run of the automaton achieves the desired task. The hybrid controller used to control the team of robots continuously executes the discrete automaton by composing continuous *atomic controllers*, based on sensory information gathered online.

There are many choices of atomic controllers to use. One

*This work was supported by ARO MURI SUBTLE W911NF-07-1-0216, grants DAAD19-02-01-0383, W911NF-05-1-0219, and W911NF-04-1-0148, NSF CNS-0410514, IIP-0742304, and IIS-0427313, and ONR N00014-07-1-0829. N. Ayanian gratefully acknowledges support from NSF. The authors thank Yaniv Sa'ar for his code and assistance.

[1]Based on the environment and proximity constraints, deadlocks may not always be avoided, see Section IV for further discussion

978-1-4244-2023-0/08/$25.00 ©2008 IEEE.

can use single robot controllers [3], [5], [12], which plan on a low dimensional space, but will not guarantee liveness with the collision avoidance schemes described above. To reduce liveness issues, we use controllers which guide multiple robots to a goal set while avoiding collisions with obstacles and other robots. Controllers such as the ones described in [13], [16] are not suitable, however, since they require hand-tuning. We base the low-level continuous controllers on the work described in [1] which, though computationally expensive, is generated automatically. Once these controllers are created (as a preprocessing step) they can be reused to accommodate many different tasks in the same workspace.

The outline of the paper is as follows. Section II defines the problem. Section III describes the process that takes a high-level task description in Structured English and automatically generates a hybrid controller that accomplishes the task, and introduces our example task. Section IV discusses how the atomic multi-robot controllers are generated and Section V presents simulations and shows how different behaviors can be addressed. We conclude in Section VI.

II. Problem

Consider a team of n kinematic robots $V_A = \{a_i | i = 1, \ldots, n\}$ moving in a polygonal workspace $W \subset \mathbb{R}^{d_i}$. Each robot a_i has the configuration $x_i \in \mathbb{R}^{d_i}$ with dynamics:

$$\dot{x}_i = u_i,\ x_i \in X_i \subset \mathbb{R}^{d_i},\ u_i \in U_i,\ i = 1, \ldots, n. \quad (1)$$

The robots must maintain static proximity constraints to ensure collision avoidance (minimum pairwise distance).

Each robot has a set of sensors $Sen = \{s_{i_j} | i = 1, \ldots, n; j = 1, \ldots, m_i\}$ that capture high level information about the world (e.g. whether a person is seen or a fire is detected). The robots may also have a set of actions $Act = \{act_{i_k} | i = 1, \ldots, n; k = 1, \ldots, l_i\}$ such as picking up objects, transmitting messages, or sounding alarms. In this paper we assume such actions do not have explicit time constraints (minimal or maximal duration).

In addition, we consider a high level task φ given as a set of Structured English sentences that the team must achieve. This task describes the desired behavior of the robots and assumptions on the sensor information.

Problem 2.1: Consider a team of robots moving on $\mathbb{R}^d$, $d = \sum_{i=1}^{n} d_i$ with dynamics (1), sensors Sen and actions Act and a high level specification φ. For any possible initial state $\{x_0, Sen_0, Act_0\}$ such that $\{x_0, Sen_0, Act_0\} \models \varphi$ find a control law $u = [u_1, u_2, \ldots, u_n]$ and an action activation policy $\pi : t \rightarrow 2^{Act}$ that for each time t specifies which actions should be active, such that

1) $\dot{x}_i = u_i$;
2) an action act_{i_k} is activated at time t if and only if $act_{i_k} \in \pi(t)$;
3) if $s_{i_j}(t) \models \varphi$, $\forall t \geq 0,\ i_j$ (the sensors satisfy the assumptions on their behavior) then $\{x_i(t), act_{i_k}(t)\} \models \varphi$, $\forall t \geq 0,\ i, i_k$ (the robots satisfy the task);

if such a system exists.

III. Task controller

This section presents the method used to transform a multi-robot high-level task, captured by Structured English instructions and a discrete abstraction of the workspace, into a hybrid controller guaranteed to drive the robots according to the desired behavior. We demonstrate this method with a recycling example.

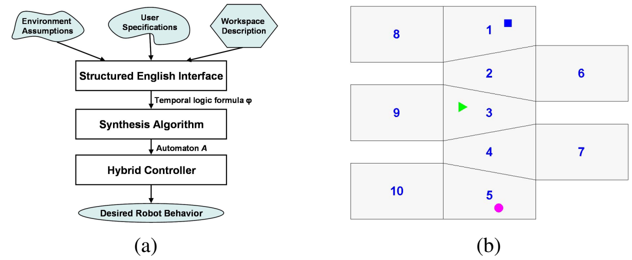


Fig. 1: Overview of method (a) and workspace (b).

Figure 1a shows the three main steps of our approach. First, the user specification and assumptions regarding the environment (the behavior of the sensor inputs, Sen) are captured using Structured English sentences. These are then translated automatically into linear temporal logic (LTL) formulas [6] and combined with a discrete abstraction of the workspace to create the formula φ which belongs to a specific fragment of LTL [10], [14]. Next, an automaton A is automatically synthesized such that every execution of A satisfies φ. Finally, a hybrid controller based on the the automaton A is created.

We illustrate these steps with the following scenario. Three robots, denoted a_1, a_2 and a_3, are moving in workspace $W \subset \mathbb{R}^2$ with ten rooms, shown in Fig 1b. Initially a_1 (blue square) is in Room 1, a_2 (green triangle) is in Room 3 and a_3 (magenta circle) is in Room 5. The high-level task requires robots to pick up different items from predesignated locations and deposit them, according to composition, in the correct location while avoiding collisions and deadlocks.

Example 1: Here robots pick up items from Rooms 6 and 7. These items can be metal, glass, or paper. When an item is ready for pickup, a robot must deposit it appropriately: metal in Room 8, glass in Room 9, paper in Room 10. If no item is present, the robots must wait in Rooms 1, 3, and 5. We additionally impose that there be at most one robot in Rooms 6 and 7 at a time, for the recyclers' peace of mind.

The first step, translation, builds upon [9]. There, the user must first define two sets of binary propositions. One set, Sen in Problem 2.1, represents information robots gather through sensors and communication. The other represents the state of the robots, controlled by the system, including locations and possible actions Act. All these propositions are then used to write the task using Structured English sentences that are automatically translated to an LTL formula.

A task description can be divided into three components, *initial conditions*, *goals*, and *transitions*. The initial conditions capture the state of the environment and system the moment the system is turned on. Goals include assumptions about the environment, for example "Eventually you will sense a flower", and desired behavior for the system , for example "eventually go to Room 9 and eventually go to Room 6". Transitions contain assumed constraints on the changes in sensor information from one time step to the next, for example "If $Mika$ is true it stays true in the next time step" which means that once a robot senses Mika she does not disappear. It also constrains possible moves the system can make, for example "Never go to Room 6" or "If you are beeping, do not beep in the next state".

Tasks we are interested in involve continuous robot motion. To capture the motion of the robots using the discrete

LTL formalism, we partition the workspace into regions and create propositions that relate the location of the robots to these regions. For example, a proposition 2_3_8 is true if a_1 is in Region 2, a_2 is in Region 3 and a_3 is in Region 8 and false otherwise Then, based on adjacency of the regions and allowable robot combinations we restrict the changes in these propositions, constraining robot motion to a feasible behavior. Given a decomposition, adding these restrictions to the transitions component of the LTL formula is automatic.

In Example 1 the sensor propositions, Sen, are: **pu6, pu7** there is an available item in Rooms 6 and 7, respectively; **m1,g1,p1,m2,g2,p2,m3,g3,p3** composition of the item a_i just picked up (metal, glass, paper, respectively). The system propositions relate to different robot actions Act: **a1PU, a2PU, a3PU** a_i should pick up an item; **a1Carry, a2Carry, a3Carry** a_i is carrying an item; **a1D, a2D, a3D** a_i should deposit the item it has been carrying; as well as **robot motion** (locations). The latter correspond to all room combinations: for example, 1_3_5 is true when a_1 is in Room 1, a_2 is in Room 3 and a_3 is in Room 5. Our workspace contains ten rooms and three robots; therefore, there are 1000 possible combinations in general.

Once the propositions are defined, the task must be specified using Structured English. In the following, the sentences refer to a_1 but the full specification contains the same sentences for a_2 and a_3 as well. $S1 - S5$ capture assumptions about sensor behavior:

- $S1$ "environment starts with false": At system start up, there is no known item to pick up.
- $S2$ "if you did not activate $a1Carry$ then always not $m1$ and not $g1$ and not $p1$": If a_1 is not carrying an item, it has no knowledge about material.
- $S3$ "if you activated $a1Carry$ and you sensed $m1$ then always $m1$" (same for g and p): Once the material type of a carried item is determined, it does not change.
- $S4$ "if you activated $a1Carry$ then always $m1$ or $g1$ or $p1$": The sensors tell the robot what type of material.
- $S5$ "if you sensed $pu6$ and you did not activate $a1PU$ and 6_X_X and you did not activate $a2PU$ and X_6_X and you did not activate $a3PU$ and X_X_6 then always $pu6$"(where 6_X_X corresponds to all room combinations in which a_1 is in Room 6. The same assumption is also written for $pu7$): If an item appears in Room 6 (7) and no robot picked up an item in Room 6 (7), then the item is still there. Without this assumption the environment can prevent satisfying the task (as explained in $S13$).

Desired system behavior is captured in $S6-S14$ together with the LTL formula relating to motion that allows the system state to change at most one robot's region at a time, thus 1_2_5 can change to 1_3_5 but not to 1_3_4.

- $S6$ "system starts in 1_3_5 with false": Initially robots are not carrying, picking up or depositing anything.
- $S7$ "activate $a1PU$ if and only if you did not activate $a1Carry$ and (you are in 6_X_X and you are sensing $pu6$ or you are in 7_X_X and you are sensing $pu7$)" - If the robot is not carrying an item and it is in a room with an available item, it should pick it up.
- $S8$ "activate $a1D$ if and only if you activated $a1Carry$ and you are in 8_X_X and you are sensing $m1$ or you activated $a1Carry$ and you are in 9_X_X and you are sensing $g1$ or you activated $a1Carry$ and you are in 10_X_X and you are sensing $p1$": The robot should drop the item it is carrying if it is in the correct room.
- $S9$ "if you activated $a1PU$ or you activated $a1Carry$ and did not activate $a1D$ then do $a1Carry$": With $S10$ and $S11$ defines that $a1Carry$ should be true between pick up and drop.
- $S10$ "if you did not activate $a1PU$ and did not activate $a1Carry$ then do not $a1Carry$"
- $S11$ "if you activated $a1D$ and activated $a1Carry$ then do not $a1Carry$"
- $S12$ "if you are not activating $pu6$ and you are not activating $pu7$ and you are not activating $a1Carry$ and you are not activating $a2Carry$ and you are not activating $a3Carry$ then go to WaitRegions" (WaitRegions is all possible permutations of Rooms 1,3 and 5): If there are no items in the workspace, the system must drive the robots to the waiting rooms.
- $S13$ "if you are not activating $a1Carry$ then if you are sensing $pu6$ then go to 6_X_X and if you are sensing $pu7$ then go to 7_X_X": When conjuncted with corresponding sentences for a_2 and a_3, if there is an itemless robot and an item, the robot must go pick it up. To satisfy this goal, we must assume $S5$ otherwise the environment can prevent reaching this goal by switching $pu6$ and $pu7$ on and off so that the robots cycle between these rooms without picking up an item.
- $S14$ "if you are activating $a1Carry$ and you are sensing $m1$ then go to 8_X_X": When conjuncted with the corresponding sentences for a_2, a_3, 9_X_X, 10_X_X, g, and p, if a robot is carrying an item, it will go to the correct drop off location.

The requirement that there be at most one robot at a time in Rooms 6 and 7 can be easily captured in the Structured English description (for example "Always not 6_6_X"). However, to reduce the size of the problem, we refer to such combinations as illegal and they are omitted from the discrete graph that represents the locations of the robots and thus never reached. This constraint reduces the number of combinations from the general 1000 to 944.

The next two steps (automaton synthesis and hybrid controller creation) follow the work in [10]. The synthesis algorithm [14] generates an automaton A that implements the desired behavior, *if this behavior can be achieved*. The states of this automaton contain the truth values of the system propositions while the truth values of the sensor propositions guard its transitions. Every execution of the automaton, based on sensor information, is guaranteed to satisfy the desired system behavior as long as the environment satisfies the assumptions encoded in φ. If the environment does not satisfy these assumptions, the automaton is no longer valid and cannot be executed.

The synthesized automaton for Example 1 contains 12,585 states. While such a task can potentially be encoded by hand in a much smaller automaton, this was created automatically and is guaranteed to be correct. A portion of the automaton is shown in Fig. 2. Circles represent the automaton states; robot propositions written inside each circle are those that are true in that state. Edges are labeled with all sensor propositions that are true when that transition is enabled. Starting from the top most state, in which the robots are in Rooms 6,3 and

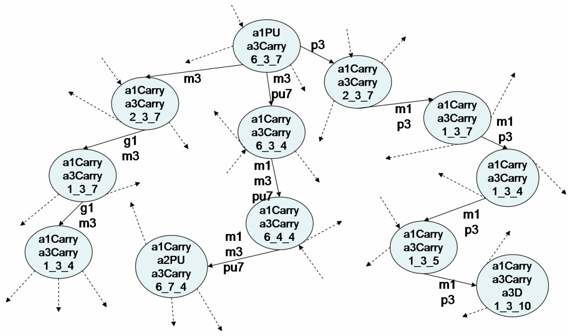
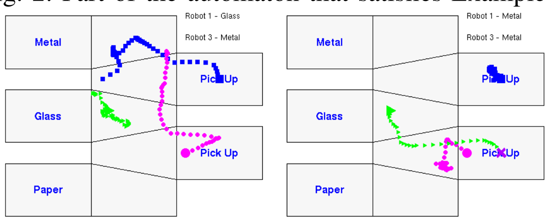


Fig. 2: Part of the automaton that satisfies Example 1



(a) Left branch (b) Middle branch

Fig. 3: Simulation of the automaton segment of Fig. 2

7, a_1 is picking up an item and a_3 is carrying an item. If there are no more items to pick up (left and right branches, in both $pu6$ and $pu7$ are false) the robots proceed to the drop off location (in the right branch, a_3 drops the paper item in Room 10, as required). If there are more items (middle branch, $pu7$ is true) a_2 proceeds to pick up the item.

The final step is to construct the hybrid controller that continuously executes A, based on the sensor inputs. Recall, from Problem 2.1, that we need to construct a motion control law u as well as an action activation policy π.

Definition 3.1: Atomic controllers or primitives are low-level continuous controllers which drive robots from any initial position in one set of locations x_y_z to another set l_m_n without going through any other combination. Furthermore, they maintain prescribed inter-agent constraints, such as avoiding collisions and maintaining communication.

The continuous motion control u is generated by switching between *multi-robot atomic controllers*, discussed in Section IV, according to the sensor inputs and the automaton states. As for the actions, for each time t the action policy $\pi(t)$ is the set of all system propositions that are true at the current automaton state.

It is important to note about automata and hybrid controllers created using this method that goals are satisfied cyclically, that is, the first goal written is reached, then the second, and finally after the last goal is achieved the automaton satisfies the first goal again and so on. In Example 1 this results in the robots first picking available items until either all robots are carrying something or there are no more items, and only then the robots deposit the items.

Figure 3 depicts part of a simulation run that corresponds to Example 1 and illustrates both the continuous execution of the automaton segment shown in Fig. 2 and the fact that using the same automaton, the behavior of the system varies significantly based on what is happening in the environment.

IV. ATOMIC CONTROLLERS

This section addresses the synthesis of atomic, multi-robot controllers that drive robots from one location to another

(a) Collision only (b) Collision with communication

Fig. 4: Proximity Constraints

while guaranteeing *safety* (collision avoidance) and other specified inter-robot constraints. The controller is based on the centralized version of [1] and the work in [7].

Consider the team of n robots V_A with dynamics (1) in some location $L = \{r_1\_r_2\_\dots\_r_n\}$, where r_i denotes the room where robot a_i is located in the workspace. One robot, the *active* robot, must transition to a new room without collisions or without any other robots transitioning to a new room (the reason for this will become clear in Section IV-A).

Definition 4.1: The *configuration space* $\mathcal{C}_i$ of a robot a_i is the set of all transformations of a_i. The *free space* $\mathcal{C}_i^{free}$ of a_i is the set of all transformations of a_i which do not intersect with obstacles in the configuration space. $\mathcal{C}_i^{free}$ is decomposed into p_i rooms with matching facets.

Definition 4.2: The *team configuration space* is the Cartesian product of the configuration spaces of each robot,

$$\begin{array}{c} \mathcal{C}_{all} = \mathcal{C}_1^{free} \times \mathcal{C}_2^{free} \times \cdots \times \mathcal{C}_n^{free} \\ x = [x_1, \dots, x_n] \in \mathcal{C}_{all}. \end{array} \tag{2}$$

Thus the configuration of all n robots is described by a single point in $\mathcal{C}_{all}$. $\mathcal{C}_{all}$ has dimension d and contains $\prod_{i=1}^{n} p_i$ polytopes. We henceforth restrict the discussion to two dimensional workspaces which are identical for each robot; however, it is easily extended to higher dimensional systems and different workspaces for different robots. Thus all robots share the same configuration space, $\mathcal{C}_i = \mathcal{C}_j$, $\mathcal{C}_i^{free} = \mathcal{C}_j^{free}$, $p_i = p_j$, $d_i = 2, \forall i, j \in \{1, \cdots, n\}$.

We specify *proximity constraints* to ensure robot safety. We require each pair (a_i, a_j) to maintain nonzero minimum distance $|x_i - x_j|_\infty \geq \delta^{i,j}_{min}$. We write this constraint

$$\lambda(x_i, x_j) \geq 0 \quad \forall i \neq j. \tag{3}$$

This constraint corresponds to an infinite square annulus in the relative space of two agents (Fig. 4a). In our example, we assume the communication range is at least as large as the workspace. If the robots' communication range is smaller than the workspace, then pairs of agents (a_i, a_j) must maintain maximum distance $|x_i - x_j|_\infty \leq \delta^{i,j}_{max}$ (Fig. 4b).

Definition 4.3: The *task configuration space* $\mathcal{C}_T$ is the set

$$\mathcal{C}_T = \mathcal{C}_{all} \cap \mathcal{L}, \tag{4}$$

$$\mathcal{L} \equiv \{x | x \in \mathcal{C}_{all}, \lambda(x_i, x_j) \geq 0 \ \forall (a_i, a_j), i \neq j\}. \tag{5}$$

$\mathcal{C}_T$ is a space composed of polytopes in which the agents cannot collide.

A. Feedback Controllers on $\mathcal{C}_T$

We now consider a subproblem of Problem 2.1.

Problem 4.4: For any initial configuration $x_0 \in L_0 \subset \mathcal{C}_T$, consider the system (1) on $\mathbb{R}^d$, where $d = \sum_{i=1}^{n} d_i$, with goal configuration $x^g \in L_g \subset \mathcal{C}_T$. Find a piecewise affine input function $u : [0, T_0] \to U$ for any $x_0 \in L_0$ such that

1) $\forall t \in [0, T_0]$, $x \in L_0 \cup L_g$, $x(T_0)$ arbitrarily close to x^g,

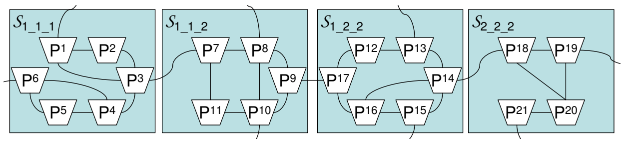


Fig. 5: A partial view of a polytope graph for three robots.

2) $\dot{x}_i = u_i$,
3) $x(t) \in \mathcal{L}$, $\forall t \in [0, T_0]$.

There are two stages in solving Problem 4.4. First, we pursue a hierarchical discrete representation of the team configuration space and find paths in this discrete representation. Second, we translate these paths into feedback controllers.

In the first stage, we associate each polytope with a room combination, $\mathcal{S}_{r_m\text{-}\cdots\text{-}r_p} = \{P^i | x \in P^i \Rightarrow x_1 \in r_m, \cdots, x_n \in r_p\}$, where P^i is the i-th polytope in $\mathcal{C}_T$. Then we define an adjacency graph on the set of all polytopes.

Definition 4.5: The *polytope graph* $G_P = (V_P, E_P)$ on the polytopes in $\mathcal{C}_T$ is the pair of sets $V_P = \{c_1, \cdots, c_n\}$, where c_i is the centroid of the i-th polytope P^i, and E_P, the set of all pairs of polytopes which share a (matching) facet.

A sample polytope graph is shown in Fig. 5. G_P is used in the creation of the automaton; thus, the automaton cannot give instructions that violate the collision constraints (and communication constraints, if included).

For all transitions from one room combination to another, we determine a discrete path from each polytope in the original room combination to the next room combination. For example, referring to Fig. 5, the paths from $\mathcal{S}_{2\_2\_2}$ to $\mathcal{S}_{1\_2\_2}$ may be $P^{19} \rightarrow P^{18} \rightarrow P^{14}$ and $P^{21} \rightarrow P^{20} \rightarrow P^{18} \rightarrow P^{14}$. If the active robot must stay in a room (to pick up/deposit) we determine a discrete path from each polytope in the current room combination to a goal position. Inactive robots stay in their current rooms and go to a goal position, (the goal position is described for every robot). We are not concerned with whether the inactive robots reach their goal position.

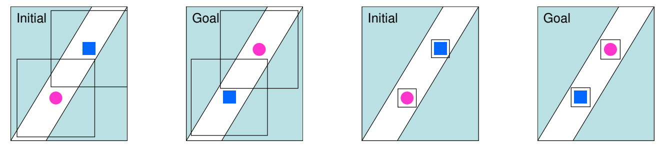
Theorem 4.6 (Necessary condition): Problem 4.4 has a solution only if the polytope graph, G_P is connected.

Proof: see [1].

We use an algorithm such as Dijkstra to choose a path which minimizes the number of polytopes visited, which minimizes the number of transitions between polytopes.

Once the paths are identified, we synthesize feedback controllers to solve Problem 4.4. The synthesis procedure is developed in [7] for determining an affine state feedback law that satisfies a set of inequalities on a polytope. This results in controllers that drive an affine system from any initial condition in a polytope through a desired exit facet. Because the atomic controllers direct states to a facet, not an edge, only one robot will cross a room threshold at any time. Thus, we restrict the automaton to commands which result in room change for only one robot, limiting the path on the polytope graph to the polytopes in the initial and final room combination. Once in the polytope containing the goal configuration we steer states to the goal configuration. This procedure, which is solved on a triangulation of the polytope, is discussed in detail in [1] (for centralized control, ignore constraints on the feedback matrix in [1] Problem 3.4).

Although the feedback controller synthesis is for point robots and requires heavy computation, it has many benefits. First, by using feedback linearization, slow-moving nonholonomic robot models can be effectively abstracted to



(a) (b) (c) (d)

Fig. 6: Panels (a) and (b): excessive minimum distance constraints, represented by boxes around the robots cause deadlock. Panels (c) and (d): resolution of the deadlock by reducing the constraints.

point robots. Additionally, a solution is guaranteed for fully actuated systems, if the polytope graph G_P is connected. Only a small number of cases result in a polytope graph that is not connected. This can sometimes be alleviated by reducing the minimum distance between robots as in Fig. 6. Finally, the solution is entirely automatic; once the number of robots, workspace, and proximity constraints are described, no other user input is required to solve the low-level problem.

In summary, the algorithm for controller synthesis or the solution to Problem 4.4 involves the following four steps:

Algorithm 4.7:

1) Construct task configuration space $\mathcal{C}_T$ (Definition 4.3).
2) On each polytope P^i, solve for a linear feedback controller as in [7] which drives any state in P^i to the exit facet it shares with each adjacent polytope P^j.
3) On each polytope P^i, solve for a linear feedback controller as in [1] which drives every state inside P^i to the goal configuration in P^i.
4) Find paths on G_P for each possible transition from one room combination L_0 to another L_g and combine the controllers which correspond to that path.

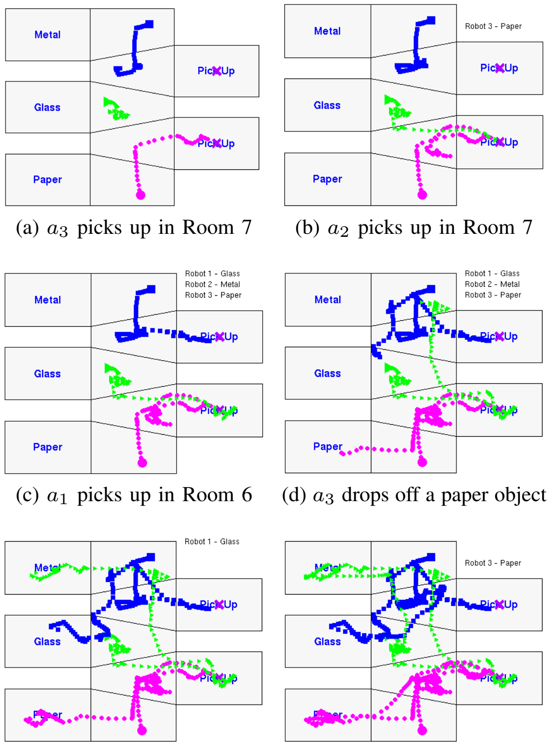
V. Simulations

In this section we show a MATLAB simulation and demonstrate how different tasks can easily be accommodated using the same atomic controllers but a different automaton. The atomic controllers were designed in MATLAB using the Multi-Parametric Toolbox for polytope computations [11]. The automata were synthesized using a prototype of the JTLV system [15].

Figure 7 depicts a sample simulation of Example 1. In this scenario, there is always something to pick up (denoted as a purple X) in both locations. a_1 (blue square), a_2 (green triangle), and a_3 (magenta circle) start in Regions 1, 3, and 5 respectively. First a_3 goes to Room 7 and picks up an object (a), then a_2 picks up in Room 7 (b) then a_1 picks up in Room 6. Note that, as required, there is at most one robot in Rooms 6 and 7 at any given time. Also, as discussed in Section IV for every discrete transition in the automaton, only one robot is changing the region it is in.

Once all robots have identified their carried item (b,c) they drop it off appropriately. a_3 drops off the paper item (d), a_1 (a_2) drops off a glass (metal) item (e). Since there are more items to pick up, the robots move towards the pickup rooms and a_3 picks up more paper in Room 7 (f).

Adding robot motion constraints can be done in two ways. One is to explicitly state such constraints in the user specifications. The other is to remove nodes and transitions from the discrete representation of $\mathcal{C}_T$.



(a) a_3 picks up in Room 7 (b) a_2 picks up in Room 7

(c) a_1 picks up in Room 6 (d) a_3 drops off a paper object

(e) a_1 and a_2 drop off a glass and a metal object respectively (f) a_3 picks up in Room 7 again

Fig. 7: Simulation of Example 1

Example 2: Add a "baby sister" constraint to Example 1 requiring robot a_2 to always follow robot a_1, i.e. they must always be either in the same or adjacent rooms.
Adding this constraint to Example 1 reduces the number of atomic controllers from 944 to 256 and the resulting automaton contains 6,874 states.

Sensor inputs as well as system outputs can be added in a very flexible way as long as the added specification does not create a logical contradiction with the previously specified task or results in an infeasible motion request.

Example 3: For safety, we forbid a_3 to enter Room 6 if a child is present. To encode this, we add to Example 2 a sensor input $kidIn6$ indicating a child is in Room 6. Then the a_3 equivalent of $S13$ becomes "if you are not activating $a3Carry$ then if you are sensing $pu6$ and not $kidIn6$ then go to X_X_6 and if you are sensing $pu7$ then go to X_X_7" and the constraint "if you are sensing $kidIn6$ then always not X_X_6" is added. The automaton contains 16,724 states.

VI. Discussion and Future Directions

We have presented a method of designing provably-correct control schemes for robot teams that achieve complex high-level tasks which are described in Structured English while providing low-level guarantees of collision and deadlock avoidance. The method involves creating a discrete automaton satisfying the task and a set of low-level controllers which can continuously implement every possible transition in the automaton. We showed the application of this method in a simulation involving three robots in ten rooms.

Given a workspace decomposition and the robots' capabilities, the method is entirely automatic and "recyclable" with minimal additional computation. Furthermore, as demonstrated in Section V, changing the specification and adding more sensor information or different robot actions is quick and easy. These advantages result in an extremely flexible system which allow non-experts to design complex systems that perform a large variety of interesting tasks.

Although this method requires an initial preprocessing stage to create the low-level controllers (which can be computationally expensive) the method requires only upfront user input (the space, number of robots, proximity constraints and the high-level specification) and no hand-tuning. Furthermore, the controllers are reused to accommodate a wide variety of high-level tasks.

The price for having a flexible, reusable system is that it is often sub-optimal. Here we allow at most one robot to change rooms at any given time which results in robots waiting their turn instead of moving concurrently. By incorporating a measure of cost into the automaton creation and examining the possibility of several robots changing rooms during a short interval of time (requiring them to change rooms at the exact same time is not a realistic solution) we can ease this limitation. This is a future research direction. Other directions include exploring the use of other controllers to solve the low-level problem and developing different sets of atomic controllers for other domains such as manipulation, assembly and computer graphics.

References

[1] N. Ayanian and V. Kumar. Decentralized feedback controllers for multi-agent teams in environments with obstacles. In *IEEE International Conference on Robotics and Automation*, Pasadena, CA, 2008.
[2] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas. Symbolic planning and control of robot motion: State of the art and grand challenges. *Robotics and Automation Magazine*, 14(1):61–70, 2007.
[3] C. Belta and L. Habets. Constructing decidable hybrid systems with velocity bounds. In *IEEE Conf. Dec. and Control*, Bahamas, 2004.
[4] J. F. Canny. *The Complexity of Robot Motion Planning*. PhD thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, MA, 1987.
[5] D. C. Conner, A. A. Rizzi, and H. Choset. Composition of local potential functions for global robot control and navigation. In *IEEE/RSJ Int'l. Conf. on Intel. Robots and Sys.*, Las Vegas, Oct. 2003.
[6] E. A. Emerson. Temporal and modal logic. In *Handbook of theoretical computer science (vol. B): formal models and semantics*, pages 995–1072. MIT Press, Cambridge, MA, USA, 1990.
[7] L. Habets and J. van Schuppen. A control problem for affine dynamical systems on a full-dimensional polytope. *Automatica*, 40(1), 2004.
[8] H. Kress-Gazit, D. C. Conner, H. Choset, A. A. Rizzi, and G. J. Pappas. Courteous cars: Decentralized multi-agent traffic coordination. *Robotics and Automation Magazine*, March 2008.
[9] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. From structured english to robot motion. In *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, San Diego, CA, October 2007.
[10] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Where's waldo? sensor-based temporal logic motion planning. In *IEEE International Conference on Robotics and Automation*, pages 3116–3121, 2007.
[11] M. Kvasnica, P. Grieder, and M. Baoti. Multi-parametric toolbox (mpt), http://control.ee.ethz.ch/ mpt/ 2004.
[12] S. Lindemann and S. Lavalle. Computing smooth feedback plans over cylindrical algebraic decompositions. In *Robotics: Science and Systems*, Philadelphia, Pennsylvania, August 2006. MIT Press.
[13] S. G. Loizou and K. J. Kyriakopoulos. Closed loop navigation for multiple holonomic vehicles. In *Intl. Conf. on Robotics and Automation*, Lausanne, Switzerland, October 2002.
[14] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of Reactive(1) Designs. In *VMCAI*, pages 364–380, Charleston, SC, January 2006.
[15] Y. Sa'ar. Java temporal logic verifier, http://jtlv.sourceforge.net 2008.
[16] H. Tanner, S. Loizou, and K. Kyriakopoulos. Nonholonomic navigation and control of cooperating mobile manipulators. *IEEE Trans. Robotics and Automation*, 19(1), February 2003.
[17] M. M. Zavlanos and G. J. Pappas. Dynamic assignment in distributed motion planning with local coordination. *IEEE Transactions on Robotics*, 24(1):232–242, February 2008.