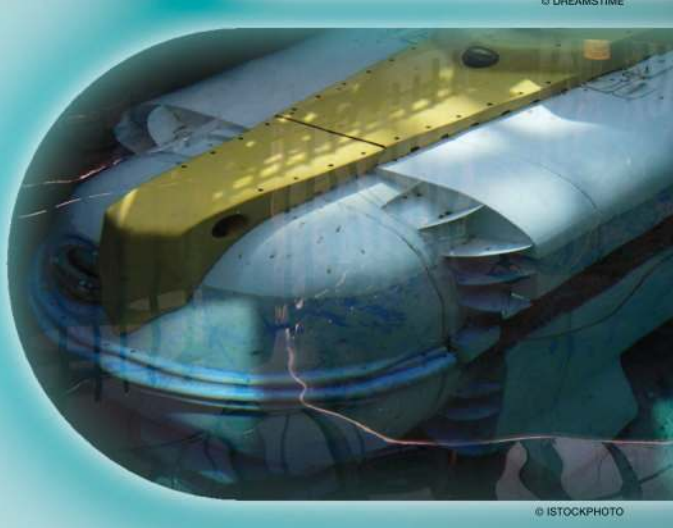
Courteous Cars

Decentralized Multiagent Traffic Coordination

BY HADAS KRESS-GAZIT, DAVID C. CONNER, HOWIE CHOSET, ALFRED A. RIZZI, AND GEORGE J. PAPPAS







Mobile Multirobot Systems

A major goal in robotics is to develop machines that perform useful tasks with minimal supervision. Instead of requiring each small detail to be specified, we would like to describe the task at a high level and have the system autonomously execute in a manner that satisfies that desired task. While the single-robot case is difficult enough, moving to a multirobot behavior adds another layer of challenges. Having every robot achieve its specific goals while contributing to a global coordinated task requires each robot to react to information about other robots, for example, to avoid collisions. Furthermore, each robot must incorporate new information into its decision framework to react to environmental changes induced by other robots since this knowledge may effect its behavior.

This article uses the approach presented in [1], in which low-level continuous feedback control policies are combined with a formally correct discrete automaton, thus satisfying a specified high-level behavior for any initial state in the domain of the low-level policies. This allows the approach to be applied to systems that react to changing dynamic environments and that may have complex nonlinear constraints, such as nonholonomic constraints, input bounds, and obstacles or body shape. Furthermore, given a collection of local feedback control policies, the approach is fully automatic and correct by construction.

Multirobot high-level behavior is captured naturally in a decentralized manner in this approach. By allowing each robot's automaton to depend on information gathered locally from other robots and the environment, each robot can react during the execution to the other robots' behaviors. The approach [1] also supports creating a single centralized controller for the group of robots. However, such a controller would encode global knowledge of all robots' state and therefore will not scale well. Furthermore, agent synchronization issues might emerge. By choosing the decentralized approach, the controller remains tractable and the agent's behavior only depends on local events. Although the decentralized approach has some limitations too, it seems more suited for multirobot behaviors.

The approach combines the strengths of control theoretic and computer science approaches. Control theoretic approaches offer provable guarantees over local domains; unfortunately, the control design requires a low-level specification of the task. In the presence of obstacles, designing a global control policy becomes unreasonably difficult. In contrast, discrete planning advances from computer science offer the ability to specify more general behaviors and generate verifiable solutions at the discrete level but lack the continuous guarantees and robustness offered by feedback.

By using a collection of local feedback control policies that offer continuous guarantees and composing them in a formal manner using discrete automata, the approach automatically creates a hybrid feedback control policy that satisfies a given high-level specification without ever planning a specific configuration space path. To be more specific, given the robot's workspace, its limitations, its sensors (i.e., the local information it can get from the environment and the other robots), and the high-level specifications it should satisfy, the

Digital Object Identifier 10.1109/M-RA.2007.914921

1070-9932/08/$25.00©2008 IEEE

approach first populates the configuration space with local continuous feedback control policies. These policies drive the robot in paths that are guaranteed to stay in the appropriate lane while avoiding collisions with static obstacles. Furthermore, these policies induce a discrete graph, i.e., if policy Φ_A drives the robot to the domain of policy Φ_B, there is a discrete transition from policy Φ_A to Φ_B. Using this discrete graph, the approach automatically synthesizes a discrete automaton that satisfies the high-level specifications.

These high-level specifications are given in a subset of linear temporal logic (LTL). Loosely speaking, temporal logic extends propositional logic (AND, OR, NOT) by adding temporal connectives (ALWAYS, EVENTUALLY, ...), thus enabling one to reason about propositions that can change truth value with time. The specifications that are considered in this article usually depend on the local input from the environment and from the other robots that are part of the environment from one robot's perspective. Finally, the system continuously executes the automaton based on the state of the environment and the vehicle by activating the continuous policies. Given proper sensor function, this execution guarantees that the robot will satisfy its intended behavior using a decentralized approach.

As a demonstration of the general approach, this article presents a familiar example: conventional Ackermann-steered vehicles operating in an urban environment. Figure 1 shows the environment and a simulation snapshot with eight currently active vehicles. The vehicles in this simulation execute one of two automata. The first automaton satisfies the high-level specification "drive around until you find a free parking space and then park." The second automaton satisfies the specification "Leave the block, obeying traffic rules, through Exit_i," where i is given as input. This article discusses the design and deployment of the local feedback policies, the automatic generation of automata that satisfy high-level specifications, and the continuous execution.

The approach to composing low-level policies is based on our earlier work using sequential composition [2], [3]. Sequential composition depends on well-defined policy domains and well-defined goal sets to enable tests that the goal set of one policy is contained in the domain of another. For idealized (point) systems, several techniques are available for generating suitable policies [4]–[8]. Our recent work extends these ideas to a more complex system model with Ackermann steering, input bounds, and the shape of the vehicle [1].

Building on the sequential composition idea [2], a recent work has shown how to compose local controllers in ways that satisfy temporal specifications given in temporal logic [9] rather than final goals. In [10]–[12], powerful model checking tools were used to find the sequence in which the controllers must be activated for the system to satisfy a high-level temporal behavior. Although these approaches can capture many interesting behaviors, their fundamental disadvantage is that they are open-loop solutions. They find sequences of policies to be invoked rather than an automaton and therefore cannot satisfy reactive behaviors that depend on the local state of the environment, as determined at run time, or handle uncertain initial conditions.

This work builds on the approach taken in [13], which is based on an automaton synthesis algorithm introduced in [14]. By creating automata rather than specifying sequences of policies, the robot can satisfy behaviors that depend on local information gathered during run time.

Local Continuous Feedback Control Policies

Local continuous feedback control policies form the foundation of the control framework; the policies are designed to provide guaranteed performance over a limited domain. Using continuous feedback provides robustness to noise, modeling uncertainty, and disturbances. This section presents the system model used in the control design, the formulation of the local policies, and the method of deployment.

System Modeling

Although this approach can be applied to different robot models, this article focuses on the control of a rear-wheel drive car-like vehicle with Ackermann steering. The vehicle, which is shown schematically in Figure 2, is sized based on a standard minivan.

The vehicle pose, g, is represented as $g = \{x, y, \theta\}$, where (x, y) is the location of the midpoint of the rear axle with respect to a global coordinate frame and θ is the orientation of the body with respect to the global x-axis. The angle of the steering wheel is $\phi \in \mathbf{I} = (-\phi_{\max}, \phi_{\max})$, a bounded interval.



Figure 1. *The environment has 40 parking spaces arranged around the middle city block. For any vehicle, the high-level specification encodes either "drive around until you find a free parking space and then park" or "leave your parking space and exit the block."*

The nonholonomic constraints inherent in the rolling contacts uniquely specify the equations of motion via a nonlinear relationship between the input velocities and the body pose velocity. Let the system inputs be $u = \{v, \omega\} \in \mathcal{U}$, where $\mathcal{U}$ is a bounded subset of $\mathbb{R}^2$, v is the forward velocity, and ω is the rate of steering. The complete equations of motion are

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ \frac{1}{L}\tan\phi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} A(g,\phi) \\ 0 \quad 1 \end{bmatrix} u. \qquad (1)$$

More compactly, the body pose velocity is $\dot{g} = A(g, \phi)u$, where $A(g, \phi)$ encodes the nonholonomic constraints.

In addition to nonholonomic constraints, the system evolution is subject to configuration constraints. The body pose is restricted by the obstacles in the environment. The pose is further constrained by local conventions of the road, such as driving in the right lane. There is an absolute mechanical limitation of $\pm\phi_{\max}$. For safety and performance reasons, we allow further steering angle constraints at higher speeds. The system inputs are constrained based on speed limits in the environment and system capabilities.

Local Policy Development

The hybrid control framework uses local feedback control policies to guarantee behavior over a local domain. These local policies are then composed in a manner that allows reasoning on a discrete graph to determine the appropriate policy ordering that induces the desired global behavior. For the policies to be composable in the hybrid control framework, the individual policies must satisfy several requirements: 1) domains lie completely in the free configuration space of the system, 2) under influence of a given policy, the system trajectory must not depart the domain except via a specified goal set, 3) the system must reach the designated goal set in finite time, and 4) the policies must have efficient tests for domain inclusion given a known configuration [3], i.e., it is easy to check whether the vehicle is in the domain of a certain policy. This article focuses on one design approach that satisfies these properties.

The navigation tasks are defined by vehicle poses that must be reached or avoided; therefore, this article defines cells in the vehicle pose space. Each cell has a designated region of pose space that serves as the goal set. Over each cell, we define a scalar field that specifies the desired steering angle, ϕ_{des}, such that steering as specified induces motion that leads to the goal set. Taking the steering angle derivative with respect to body pose gives a reference steering vector field over the cell. This leads to a relatively simple constrained optimization problem over the bounded input space. The resulting policies are able to satisfy the four requirements given earlier.

The approach to defining the cell boundary and desired steering angle is based on a variable structure control approach [15]. The cells are parameterized by a local path segment in the workspace plane [Figure 3(a)]. The workspace path is lifted to a curve in body pose space by considering the path tangent vector orientation as the desired heading. One end of the path serves as the center of the goal set. This work uses line segments and circular arcs for the path segments. Other path shapes are possible at a cost of more complex derivative calculations [16].

To perform the control calculations, the body pose is transformed to a local coordinate frame assigned to the closest point on the path to current pose. The policy defines a boundary in the local frames along the path. Figure 3(b) shows the cell boundary defined by the local frame boundaries along the path; the interior of this tube defines the cell. The size of the tube can be specified subject to constraints induced by the path radius of curvature and the vehicle steering bounds. The cell can be tested for collision with an obstacle using the technique outlined in [3].

We define a surface in the local frame to serve as a sliding surface for purposes of defining a desired steering angle [15]. To generate a continuous steering command, the sliding surface is defined as a continuous function with a continuous bounded derivative; a blending zone is defined around the sliding surface. Outside the blending zone, the desired steering is set to a steering limit, ϕ_{lim}, where $|\phi_{\text{lim}}| \leq \phi_{\max}$. The sign of ϕ_{lim} depends on the current direction of travel (forward or reverse) and whether the current body pose in local

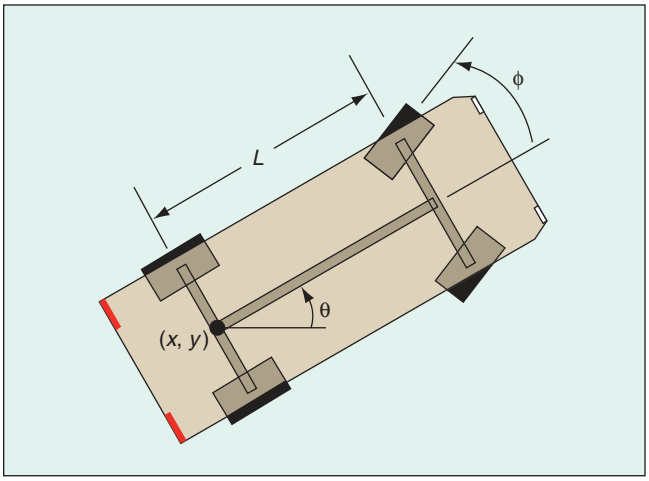


Figure 2. *Car-like system with Ackermann steering. The inputs are forward velocity and steering angle velocity.*

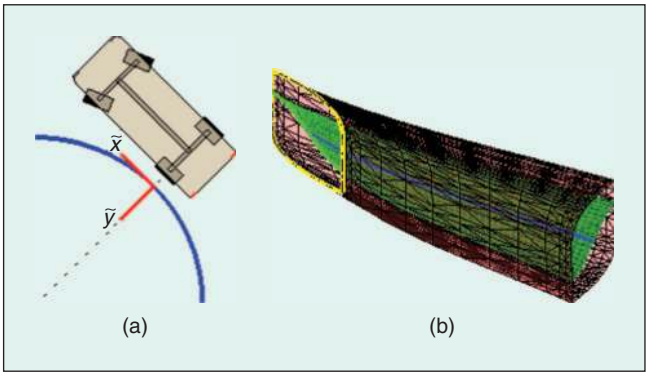


Figure 3. *Control policy based on [15]: (a) workspace path with local frame defined and (b) the cell boundary forms a tube around the path in pose space. The sliding surface is shown in the cell interior.*

coordinates is above or below the sliding surface. (For policies that move the system in reverse, the positive or negative signs are swapped.) Inside the blending zone, let

$$\phi_{\text{des}} = \eta\phi_{\text{lim}} + (1-\eta)\phi_{\text{ref}}, \tag{2}$$

where $\eta \in [0, 1]$ is a continuous blending function based on distance from the sliding surface and ϕ_{ref} is the steering command that would cause the system to follow the sliding surface. Thus, (2) defines a mapping from the body pose space to the desired steering angle for any point in the cell. The sliding surface is designed such that steering according to ϕ_{des} will cause the system to move toward the sliding surface and then along the sliding surface toward the specified curve in the desired direction of travel. At the boundary of the cell, the desired steering must generate a velocity that is inward pointing, which constrains the size and shape of a valid cell.

For a closed-loop policy design, the system must steer fast enough so that the steering angle converges to the desired steering angle faster than the desired steering angle is changing. This induces an additional constraint on the input (velocity and rate of steering) space. Given this constraint, a simple constrained optimization is used to find a valid input. Each policy is verified to ensure that a valid input exists over its entire domain during specification.

The vehicle closed-loop dynamics over the cell induce a family of integral curves that converge to the curve specifying the policy. To guarantee that an integral curve never exits the cell during execution, we impose one additional constraint. Define the steering margin, ϕ_{margin}, as the magnitude of the angle between the desired steering along the cell boundary and the steering angle that would allow the system to depart the cell. During deployment, the policies must be specified with a positive steering margin. To use the control policy, we require that $|\phi_{\text{des}} - \phi| < \phi_{\text{margin}}$. Initially, if $|\phi_{\text{des}} - \phi| \geq \phi_{\text{margin}}$, the system halts and steers toward the desired steering angle until $|\phi_{\text{des}} - \phi| \leq \phi_{\text{margin}}$. Invoking the policies this way guarantees that the system never departs the cell, except via the designated goal set; i.e., the policy is conditionally positive invariant [3]. As the vehicle never stops once the steering policy becomes active, the system reaches the designated goal in finite time.

Local Policy Deployment

To set up the basic scenario, we define the urban parking environment, shown in Figure 1, based on a green practices guideline for narrower streets [18]. The regularity of the environment allows an automated approach to policy deployment.

First, we specify a cache of local policies using the generic policy described earlier. The cache uses a total of 16 policies: one policy for normal traffic flow, four policies associated with left and right turns at the intersections, six policies associated with parking, and five associated with leaving a parking space. Ten of the policies move the vehicle forward, and six move the vehicle in reverse. Each policy in the cache is defined relative to a common reference point. At this point, the specification of the free parameters for each policy in the cache is a trial-and-error process that requires knowledge of the environment, the desired behaviors, and some engineering intuition. During specification of the policies, we verify that the convergence and invariance properties are satisfied and that the policies are free of obstacle collision based on the road layout.

Policies from the cache are then instantiated at grid points defined throughout the roadways. This is done offline based on knowledge of the local roadways. The instantiation process selects a subset of the policies in the cache based on the grid point location. Given the cache and specified grid points, the instantiation process is automated. Normally, the test for obstacle collision would be conducted as the policies are instantiated, but the regularity of the roadway renders this unnecessary. For intersections, the four turning policies are deployed for each travel direction along with the basic traffic flow policy. For the straight traffic lanes, the grid points lie in the middle of the traffic lanes aligned with the front of the parking space markers; the orientation is defined by the traffic flow. The basic traffic flow policy is always deployed at these grid points.

If a potential parking space is adjacent to the grid point, a special parking policy is instantiated. Although considered a single policy by the automaton synthesis, each parking policy is actually composed of several policies from the cache. The parking component policies are only instantiated when the parking behavior is invoked for the first time by the global parking automaton (see "Automation Synthesis" section). Figure 4 shows an example parking maneuver induced by the composition of the local feedback control policies. The same applies for special leaving policies that are a composition of several policies causing the vehicle to leave a parking space. For the region defined in Figure 1, there are initially a total of 306 policies, including 40 parking policies associated with the 40 possible parking spaces. Five policies are instantiated for each parking behavior invoked, and five policies instantiated for leaving a parking space. These are added on an as-needed basis; the appropriate nodes are appended to the automaton.

As part of the instantiation process, we test for goal set inclusion pairwise between policies. The policies in the cache are specially defined so that policies instantiated at neighboring grid points prepare one another appropriately. If the goal set of one policy is contained in the domain of a second, the first is said to prepare the second [2]. This pairwise test defines the

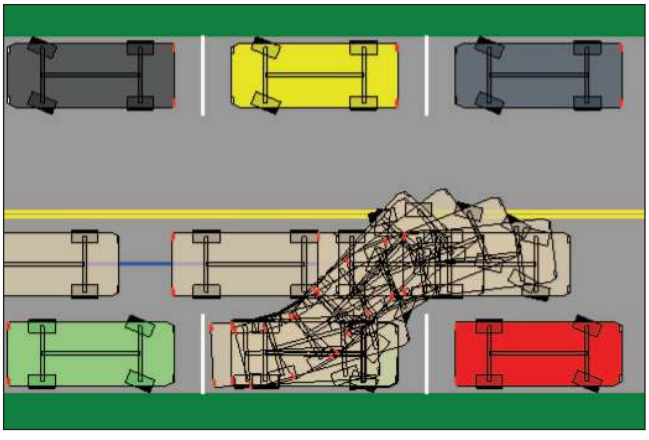


Figure 4. *Parking behavior induced by the composition of local policies. The feedback control policies guarantee the safety of the maneuver.*

prepares graph, which encodes the discrete transition relation between policies. This graph forms the foundation of the automaton synthesis approach described in the next section. The policy specification, instantiation, and prepares testing is done offline prior to the system generating the automaton.

Automaton Synthesis

This section describes the method used to create the automata that governs the local policies' switching strategy. These automata are guaranteed to produce paths that satisfy a given specification in different dynamic environments, if such paths exist.

Synthesis Algorithm

We are given a set of binary inputs (e.g., a binary input that is true when the closest parking spot is empty and false otherwise, a local hazard detected), a set of binary outputs (e.g., whether or not to activate policy Φ_i, signal left (right) turn, parking here, leaving adjacent spot), and a desired relationship between inputs and outputs (e.g., "if you sense an empty parking space, invoke a parking policy"). The realization or synthesis problem consists of constructing a system that controls the outputs such that all of its behaviors satisfy the given relationship or determine that such a system does not exist.

The relationship is given as an LTL with a specific structure [9], and the system is built using the algorithm introduced in [14]. There, the synthesis process is viewed as a game played between the system, i.e., the robot, which controls the outputs, and the environment, which controls the inputs. The two players have initial conditions and a transition relation defining the moves they can make. The winning condition for the game is given as a Generalized Reactivity (1) (a fragment of LTL) formula σ. The way the game is played is that at each step, first the environment makes a transition according to its transition relation, and then the system makes its own transition (constraints on the system transitions include obeying the prepares graph). If the system can satisfy σ no matter what the environment does, we say that the system is winning and we can extract an automaton. However, if the environment can falsify σ, we say that the environment is winning and the desired behavior is unrealizable, which means that there is no automaton that can satisfy the requirements.

The synthesis algorithm [14] takes the initial conditions, transition relations, and winning condition, and then checks whether the specification is realizable. If it is, the algorithm extracts a possible, but not necessarily unique, automaton that implements a strategy that the system should follow to satisfy the desired behavior.

Writing Logic Formulas

Informally, LTL formulas are built using a set of boolean propositions, the regular boolean connectives not ($\neg$), and ($\wedge$), or ($\vee$), implies ($\Rightarrow$), if and only if ($\Leftrightarrow$), and temporal connectives. The temporal connectives include next ($\bigcirc$), always ($\Box$) and eventually ($\Diamond$). These formulas are interpreted over infinite sequences of truth assignments to the propositions. For example, the formula $\bigcirc(p)$ is true if in the next position p is true. The formula $\Box(q)$ is true if q is true in every position in the sequence. The formula $\Box\Diamond(r)$ is true if always eventually r is true, i.e., if r is true infinitely often.

The input to the algorithm is an LTL formula

$$\varphi = (\varphi_e \Rightarrow \varphi_s).$$

φ_e is an assumption about the inputs and thus about the behavior of the environment, and φ_s represents the desired behavior of the system. More specifically,

$$\varphi_e = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e; \quad \varphi_s = \varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s.$$

φ_i^e and φ_i^s describe the initial condition of the environment and the system. φ_t^e represents the assumptions on the environment by constraining the next possible input values based on the current input and output values. φ_t^s constrains the moves the system can make, and φ_g^e and φ_g^s represent the assumed goals of the environment and the desired goals of the system, respectively. For a detailed description of these formulas, see [13].

Translating this formula to a game, the initial condition is $\varphi_i^e \wedge \varphi_i^s$, the transition relations for the players are φ_t^e and φ_t^s, and the winning condition is $\sigma = (\varphi_g^e \Rightarrow \varphi_g^s)$. Note that there are two ways for the system to win. It wins if either φ_g^s is satisfied, i.e., the system reaches its goals, or φ_g^e is falsified. The latter case implies that if the environment does not satisfy its goals (either a faulty environment or the system interfered), then a correct behavior of the system is no longer guaranteed. Furthermore, if during an execution of the automaton, the environment violates its own transition relation, the automaton is no longer valid.

In the following sections, we explain in detail how to encode the specifications. "Adhering to Traffic Laws" section first describes an LTL formula that encodes appropriate behavior in traffic, i.e., the reaction to hazardous conditions and the activation of the turn signals. This LTL formula captures the multirobot aspect of the behavior. "Parking" and "Leaving" sections then add the more specialized behavior for the parking and leaving tasks, respectively.

Adhering to Traffic Laws

Socially acceptable driving behavior includes stopping at stop lights, driving in the designated lane, keeping a safe distance from vehicles ahead, and using the left and right turn signals. To encode such behavior, we define one input, hazard, which becomes true whenever the car must stop. Such an input may be the result of a proximity sensor in the case of keeping a safe distance from another vehicle or of a vision system recognizing a red light at the intersection or another vehicle signaling that it is about to make a turn. The hazard is also used to cause a vehicle intending to park to wait on a vehicle that is ready to leave an occupied parking space. Although in some cases, the more natural reaction to such conditions is to slow down rather than stop, here we take the more conservative approach for simplicity. The local feedback control policies serve as outputs. Additional output propositions are signalL and signalR, which indicate whether the left (right) turn signal should be activated, and the proposition "stop," which indicates whether the vehicle should stop. These outputs are detectable by other robots. The formula encoding this behavior is given in the following list.

1) *Assumptions on the environment:* Initially, there is no need to stop; therefore, $\varphi_i^e = \neg$hazard. We do not impose any further restrictions on the behavior of the hazard input; thus, it can become true or false at any time. To keep the structure of the formula, we encode both φ_t^e and φ_g^e as the trivial formula true

$$\varphi_t^e \wedge \varphi_g^e = \Box\,(\text{TRUE}) \wedge \Box \Diamond (\text{TRUE}),$$

which means that these formulas are always satisfied.

2) *Constraints on the behavior of the vehicle (system):* Initially, the vehicle must be in the domain of an initial policy, no turn signal is on (we assume the vehicle starts by driving straight

$$\varphi_i^s = \vee_{i \in \text{InitialPolicy}} \Phi_i \wedge \neg\,\text{SignalL} \wedge \neg\,\text{SignalR} \wedge \neg\,\text{stop},$$

which will be changed in the "Leaving" section), and the vehicle is not required to stop. The vehicle can only transition from one policy to the next based on the prepares graph from the "Local Policy Deployment" section (first line of φ_t^s below). It must turn the left turn signal only if it is turning left and the same for the right turn signal (second and third line). It must stop if and only if the hazard signal is true (last line).

$$\varphi_t^s = \begin{cases} \wedge_i \Box\,(\Phi_i \Rightarrow (\bigcirc\,\Phi_i \vee_{j \in \text{SuccessorsOfPolicy}_i} \bigcirc\,\Phi_j)) \\ \wedge \Box\,((\vee_{j \in \text{LeftTurnPolicies}} \bigcirc\,\Phi_j) \Leftrightarrow \bigcirc\,\text{signalL}) \\ \wedge \Box\,((\vee_{j \in \text{RightTurnPolicies}} \bigcirc\,\Phi_j) \Leftrightarrow \bigcirc\,\text{signalR}) \\ \wedge \Box\,(\bigcirc\,\text{hazard} \Leftrightarrow \bigcirc\,\text{stop}). \end{cases}$$

Finally, since we are only concerned with obeying traffic laws and we do not require the vehicle to go anywhere, we simply write $\varphi_g^s = \Box \Diamond (\text{TRUE})$.

Parking

In this scenario, a vehicle is searching for an empty parking space and parks once it finds one. Starting from the formula in the "Adhering to Traffic Laws" section, we define another input, park, which becomes true when an empty parking space is found.

1) *Assumptions on the environment:* We add these subformulas to φ_e of the "Adhering to Traffic Laws" section: Initially there is no parking near the vehicle; therefore, we add $\neg$ park to φ_i^e. We can only determine whether there is a free parking space if we are in a policy next to it, i.e., park cannot become true if the vehicle is not next to a parking space or in one (first subformula). Also, for implementation reasons, we assume that the input park remains true after parking (second subformula). These subformulas are added to φ_t^e

$$\begin{cases} \Box\,([(\neg(\vee_{i \in \text{ParkPolicy}}\,\Phi_i)) \wedge (\neg(\vee_{j \in \text{PreparesParkPolicy}}\,\Phi_j))] \\ \quad \Rightarrow \neg \bigcirc\,\text{park}) \\ \wedge \\ \Box\,((\text{park} \wedge (\vee_{i \in \text{ParkPolicy}} \Phi_i)) \Rightarrow \bigcirc\,\text{park}). \end{cases}$$

We have no assumptions on the infinite behavior of the environment (we do not assume that there is an empty parking spot); therefore, the goal component remains set to true.

2) *Constraints on the behavior of the vehicle (system):* Here, we add the parking requirement to φ_t^s, which state that the vehicle cannot park if there is no parking space available, indicated by the park input (first line). If there is an empty parking space, it must park (second line).

$$\begin{cases} \wedge_{i \in \text{ParkPolicy}} \Box\,(\neg \bigcirc\,\text{park} \Rightarrow \neg \bigcirc\,\Phi_i) \\ \wedge \Box\,(\bigcirc\,\text{park} \Rightarrow (\vee_{i \in \text{ParkPolicy}} \bigcirc\,\Phi_i)). \end{cases}$$

Finally, we replace φ_g^s by adding a list of policies the vehicle must visit infinitely often if it has not parked yet. These policies define the area in which the vehicle will look for an available parking space.

$$\varphi_g^s = \wedge_{i \in \text{VisitPolicy}} \Box \Diamond (\Phi_i \vee \text{park} \vee \text{stop}).$$

Note that the goal condition is true if either the vehicle visits these policies infinitely often (when there is no parking space available) or it has parked or it has stopped (because of an accident ahead of it or a broken stop light).

Leaving

In this scenario, a vehicle is leaving its parking space and exiting the block via some specified exit. As before, starting from the formula in the "Adhering to Traffic Laws" section, we define as additional inputs Exit Φ_i for i $\in$ ExitPolicies. These are inputs that are constant and define which exit the vehicle should use (the proposition that is true), thus two vehicle leaving may use the same generated automaton with different inputs.

1) *Assumptions on the environment:* We add these subformulas to φ_e. Initially only one Exit Φ_i is true. This is added to φ_i^e

$$\vee_{i \in \text{ExitPolicies}}\ (\text{Exit}\,\Phi_i \wedge_{j \in \text{ExitPolicies}, j \neq i} \neg\,\text{Exit}\,\Phi_j).$$

We require the input to be constant, which means that they cannot change. Therefore, we add to φ_t^e

$$\vee_{i \in \text{ExitPolicies}}\ (\text{Exit}\,\Phi_i \Leftrightarrow \bigcirc \text{Exit}\,\Phi_i).$$

We have no assumptions on the infinite behavior of the environment; therefore, the goal component remains set to true.

2) *Constraints on the behavior of the vehicle (system):* Initially, the car is leaving a parking space, hence it must turn on the left turn signal. We modify φ_i^s to be

$$\varphi_i^s = \vee_{i \in \text{InitialPolicy}} \Phi_i \wedge \text{SignalL} \wedge \neg\text{SignalR} \wedge \neg\text{stop}.$$

We do not add any further subformulas to φ_t^s of the "Adhering to Traffic Laws" section. As for φ_g^s, we replace it with the requirement that the vehicle must go to the designated exit policy if it has not stopped.

$$\varphi_g^s = \wedge_{i \in \text{ExitPolicies}} \Box \Diamond ((\Phi_i \Leftrightarrow \text{Exit}\Phi_i) \vee \text{stop}).$$

Continuous Execution of Discrete Automata

The synthesis algorithm generates an automaton that governs the execution of the local policies; however, the continuous evolution of the system induced by the local policies governs the state transitions within the automaton. In this section, we discuss the implementation of the policy switching strategy.

Execution

A continuous execution of the synthesized automaton begins in an initial state q_0 that is determined by linearly searching the automaton for a valid state according to the initial body pose of the vehicle. From state q_i at each time step, the values of the binary inputs are evaluated. (We assume the time step is short compared with the time constant of the closed-loop dynamics.) On the basis of these inputs, all possible successor states are determined. If the vehicle is in the domain of policy Φ_l, which is active in a successor state q_j , the transition is made. Otherwise, if the vehicle is still in the domain of Φ_k, which is active in state q_i, the execution remains in this state. The only case in which the vehicle is not in the domain of Φ_k, or in any successor Φ_l, is if the environment behaved badly. It either violated its assumptions, thus rendering the automaton invalid, or it caused the vehicle to violate the prepares graph (e.g., a truck running into the vehicle). In the event that a valid transition does not exist, the automaton executive can raise an error flag, thereby halting the vehicle and requesting a new plan. This continuous execution is equivalent to the discrete execution of the automaton [10], [12].

Guarantees of Correctness

We have several guarantees of correctness for our system, starting from the high-level specifications and going down to the low-level controls. First, given the high-level specification encoded as an LTL formula, the synthesis algorithm reports whether the specification is realizable or not. If an inconsistent specification is given, such as, "always keep moving and if you see a stop light stop," the algorithm will return that there is no such system. Furthermore, if a specification requires an infeasible move in the prepares graph, such as "always avoid the left north or south road and eventually loop around all the parking spaces," the algorithm will report that such a system does not exist.

Second, given a realizable specification, the algorithm is guaranteed to produce an automaton such that all its executions satisfy the desired behavior if the environment behaves as assumed. The construction of the automaton is done using φ_t^e, which encodes admissible environment behaviors; if the environment violates these assumptions, the automaton is no longer correct. The automaton state transitions are guaranteed to obey the prepares graph by the low-level control policy deployment unless subject to a catastrophic disturbance (e.g., an out of control truck). Modulo a disconnect between φ_t^e and the environment, or a catastrophic disturbance to the continuous dynamics, our approach leads to a correct continuous execution of the automaton that satisfies the original high-level desired behavior.

Sensors, or more specifically, the binary inputs used by the automaton, are of great importance in this framework. First, as mentioned earlier, they must satisfy the assumptions made about them in the LTL formula; otherwise, the automaton will not be correct. Second, even if they do satisfy these assumptions, they may still cause correct yet unintended behavior. For example, if the proximity sensor set the hazard input to true whenever another vehicle was in a certain radius, even if that vehicle was behind in a forward driving lane, both vehicles may get deadlocked, i.e., both would stop forever. Although this behavior satisfies the original specification, it does not follow the spirit of finding a parking space. (This is a classical problem in concurrent systems. There, fairness assumptions are imposed on the inputs to ensure that the system will not deadlock.) On the other hand, both cars stopping might be a desired behavior when an accident occurred; therefore, we would not want to forbid it in the specifications. Such unintended behavior would not be present in a centralized approach where the controller has full knowledge and not just local information as is the case here. However, with careful design of the inputs, such behaviors can be avoided.

Results

The approach is verified in a simulation executed using MATLAB. First, the workspace is laid out, and a cache of policies is specified. Second, the policies are automatically instantiated in the configuration space of the vehicle, and the prepares graph is defined. Next, the LTL formulas are written. Each LTL formula is then given to the automatic synthesis algorithm implemented by Piterman et al. [14] on top of the temporal logic verifier system [17]. At this point, the resulting automaton is used to govern the execution of the local policies, based on the local behavior of the environment. The vehicles are able to react in real time to disturbances via the local continuous feedback and environmental changes sensed locally due to the automaton.

In such an execution, we must simulate the sensors that govern the behavior of the park and hazard inputs. The park input is set to true whenever there is a free parking space near by. The hazard input that enables the traffic law abiding behavior and thus the multirobot task should be set to true whenever the car must stop. Here, we simulate a proximity sensor with added logic that sets hazard to true whenever the car is too close to a car ahead of it (keeping safe distance), whenever a car ahead is backing up to park (being polite), whenever the car is leaving a parking space and another car passes by, and whenever another car is leaving a parking space which the car will park in next. We also simulate a vision system that detects whether the stoplight is red.

In the following example, the workspace is the one shown in Figure 1, with the 306 policies instantiated as described in the "Local Policy Deployment" section. In the parking LTL formula, the visit policies correspond to the eight lanes around the parking spaces (four going clockwise and four going counter clockwise), and the initial policies correspond to the ten entry points to the workspace. Likewise, in the leaving automaton, the 40 parking spaces are the possible initial policies, and the ten exit points are the possible goals. Initially, 35 of the 40 parking spaces were randomly specified as occupied.

In this simulation, eight cars enter the block at different times and from different entry points, looking for a parking

space. During the execution, an additional three cars leave their parking spaces and exit the workspace. Figure 5 shows a general snapshot of the simulation. At this point in time, seven cars are moving in the workspace. Cars that are marked with red ellipses are the cars whose hazard input is true; therefore, they have stopped. All stopped cars in this figure are obeying stoplights.

Figure 6 shows several close-up looks at different traffic behaviors encountered during the simulation. In Figure 6(a), the blue car that is leaving the parking space has stopped, indicated by a red ellipse, to let the brown car drive by. This hazard was invoked based on a proximity sensor. In Figure 6(b), red car is parking while the blue car waits for it to finish before passing. In Figure 6(c), the orange car is stopping to allow the gray car to complete a left turn. The white car on the left is leaving the parking space that later will be occupied by the brown car. Figure 6(d) shows two cars stopping before a stoplight. While the white car stopped based on the stoplight, the black car behind stopped based on the proximity to the car ahead of it. Figure 6(e) and (f) is the two snapshots of two cars parking simultaneously in opposite lanes. The car that started the parking maneuver later (bottom lane) pauses to allow the other car to park safely.

The video of this simulation can be viewed at [19].

Conclusions and Future Work

In this article, we have demonstrated, through the parking and leaving example, how high-level specifications containing multiple temporally dependent goals can be given to a team of realistic robots, which in turn automatically satisfy them. By switching between low-level feedback control policies and moving in a well-behaved environment, the correctness of each robot's behavior is guaranteed by the automaton. The system satisfies the high-level specification without needing to plan the low-level motions in configuration space.

Furthermore, given a collection of local feedback control policies, the approach is fully automatic and correct by construction.

Sensor inputs play a crucial role in this framework, as explained in the "Continuous Execution of Discrete Automata" section. A hazard input becoming true at the wrong time may lead to deadlock. Deciding when and how long to stop is a hard problem even for humans, as sometimes demonstrated at four-way stops, let alone robots. Therefore, in the future, we wish to explore how such inputs should be designed, implemented, and verified.

We plan to extend this work in several other directions. At the low level, we wish to consider more detailed dynamics. At the high level, we intend to address more complex robot coordination and tasks. Our research also focuses on accessible specification languages such as some form of natural language. Furthermore, we plan to run several experiments with real systems that demonstrate the work described in this article.

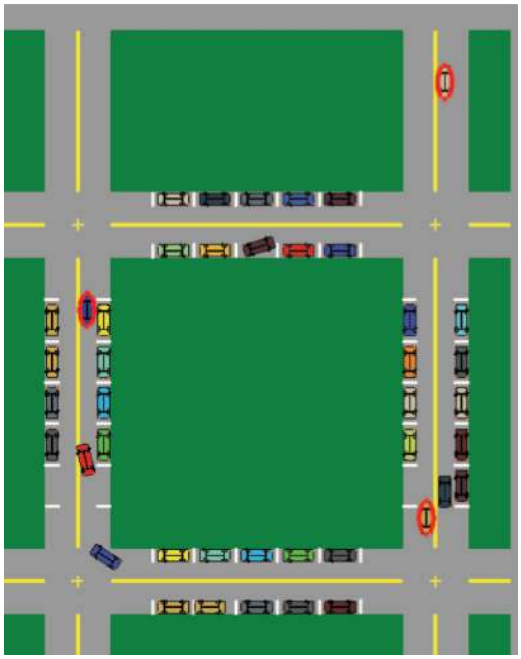


Figure 5. *A snapshot of the simulation. Cars surrounded by red ellipses are cars that are stopping because of the hazard input, in this case based on a stoplight.*

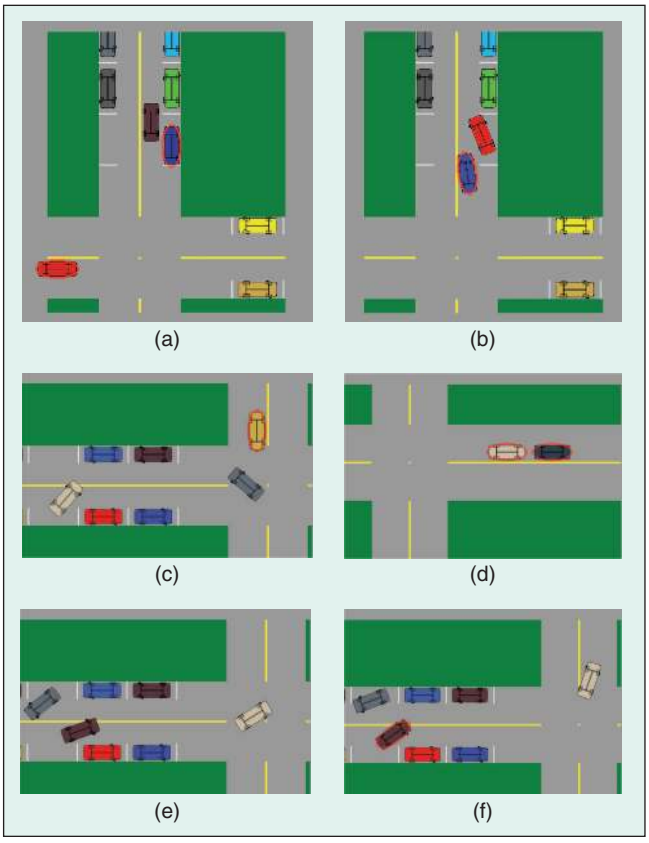


Figure 6. *Close-up looks at different behaviors seen throughout the simulation. (a) Blue car leaving. (b) Red car parking. (c) Yielding to turn in progress. (d) Two cars at stoplight. (e) Two cars parking. (f) Two cars parking.*

Acknowledgment

This work was partially supported by Army Research Office MURI DAAD 19-02-01-0383.

Keywords

Multirobot, hybrid control, motion planning.

References

[1] D. C. Conner, H. Kress-Gazit, H. Choset, A. A. Rizzi, and G. J. Pappas, "Valet parking without a valet," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, San Diego, CA, Oct. 2007, pp. 572–577.

[2] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, "Sequential composition of dynamically dexterous robot behaviors," *Int. J. Robot. Res.*, vol. 18, no. 6, pp. 534–555, 1999.

[3] D. C. Conner, H. Choset, and A. A. Rizzi, "Integrated planning and control for convex-bodied nonholonomic systems using local feedback control policies," in *Proc. Robotics: Science and Systems II*, Philadelphia, PA, 2006, pp. 57–64.

[4] A. A. Rizzi, "Hybrid control as a method for robot motion programming," in *IEEE Int. Conf. Robotics and Automation*, May 1998, vol. 1, pp. 832–837.

[5] D. C. Conner, A. A. Rizzi, and H. Choset, "Composition of local potential functions for global robot control and navigation," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, Las Vegas, NV, Oct. 2003, pp. 3546–3551.

[6] L. Yang and S. M. Lavalle, "The sampling-based neighborhood graph: An approach to computing and executing feedback motion strategies," *IEEE Trans. Robot. Automat.*, vol. 20, pp. 419–432, June 2004.

[7] C. Belta, V. Isler, and G. J. Pappas, "Discrete abstractions for robot planning and control in polygonal environments," *IEEE Trans. Robot.*, vol. 21, pp. 864–874, Oct. 2005.

[8] S. R. Lindemann, I. I. Hussein, and S. M. LaValle, "Realtime feedback control for nonholonomic mobile robots with obstacles," in *IEEE Conf. Decision and Control*, San Diego, CA, 2006, pp. 2406–2411.

[9] E. A. Emerson, "Temporal and modal logic," *Handbook of Theoretical Computer Science (vol. B): Formal Models and Semantics*. Cambridge, MA: MIT Press, 1990, pp. 995–1072.

[10] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for mobile robots," in *IEEE Int. Conf. Robotics and Automation*, 2005, pp. 2020–2025.

[11] G. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Hybrid controllers for path planning: A temporal logic approach," in *IEEE Conf. Decision and Control*, Seville, Spain, 2005, pp. 4885–4890.

[12] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from LTL specifications," in *9th Int. Workshop on Hybrid Systems: Computation and Control*, Santa Barbara, CA, 2006.

[13] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Where's waldo? Sensor-based temporal logic motion planning" in *IEEE Int. Conf. Robotics and Automation*, 2007, pp. 3116–3121.

[14] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of Reactive(1) Designs," in *VMCAI*. Springer-Verlag, New York, Jan. 2006, pp. 364–380.

[15] A. Balluchi, A. Bicchi, A. Balestrino, and G. Casalino, "Path tracking control for Dubin's car," in *IEEE Int. Conf. Robotics and Automation*, Minneapolis, MN, 1996, pp. 3123–3128.

[16] W. L. Nelson, "Continuous curvature paths for autonomous vehicles," in *IEEE Int. Conf. Robotics and Automation*, Scottsdale, AZ, 1989, vol. 3, pp. 1260–1264.

[17] A. Pnueli and E. Shahar. (1996, May 5). The TLV system and its applications. [Online]. Available: http://www.cs.nyu.edu/acsys/tlv/

[18] Green home building guidelines. http://www.nahbrc.org/greenguidelines/userguide_site_innovative.html. Accessed Jan. 2008.

[19] Multiple cars in traffic. http://www.grasp.upenn.edu/~hadaskg/MultipleCarsInTraffic.avi

Hadas Kress-Gazit graduated with B.Sc. degree in electrical engineering from the Technion in 2002. During her undergraduate studies, she worked as a hardware verification engineer for IBM. After graduating and prior to entering graduate school, she worked as an engineer for RAFAEL. In 2005, she received the M.S. degree in electrical engineering from the University of Pennsylvania, where she is currently pursuing a Ph.D. Her research focuses on generating robot controllers that satisfy high-level tasks using tools from the formal methods, hybrid systems, and computational linguistics communities. She is a Student Member of the IEEE.

David C. Conner received the Ph.D. degree in robotics from Carnegie Mellon University in 2007. He received an M.S. degree in robotics in 2004. His research interests include applications of differential geometry, hybrid systems, and control theory for mobile robot navigation and control. He graduated with B.S. and M.S. degrees in mechanical engineering from Virginia Tech (VPI & SU) in 1991 and 2000, respectively. He is currently a research scientist with TORC Technologies in Blacksburg, Virginia. He is a Student Member of the IEEE.

Howie Choset received the Ph.D. degree in mechanical engineering from the California Institute of Technology in 1996. He is currently an associate professor at Robotics Institute at Carnegie Mellon University, where he conducts research in motion planning and design of serpentine mechanisms, coverage path planning for demining and painting, mobile robot sensor-based exploration of unknown spaces, and education with robotics. He is a Member of the IEEE.

Alfred A. Rizzi received the Sc.B. from MIT in 1986 and the M.S. and Ph.D. degrees from Yale University in 1990 and 1994, respectively, all in electrical engineering. Prior to entering graduate school, he served as a design engineer with the Northrop Corporation. He is currently a lead robotics scientist at Boston Dynamics. His research focus includes a combination of topics related to dynamically capable robotic systems and industrial and commercial application of such systems. He is a Member of the IEEE and a member of the editorial board for the *International Journal of Robotics Research*.

George J. Pappas received the Ph.D. degree in electrical engineering and computer sciences from the University of California, Berkeley, in 1998. He is currently a professor in the Department of Electrical and Systems Engineering and the Deputy Dean of the School of Engineering and Applied Science. His research focuses on the areas of hybrid and embedded systems, hierarchical control systems, distributed control systems, nonlinear control systems, and geometric control theory, with applications to robotics, unmanned aerial vehicles, and biomolecular networks. He is a Senior Member of the IEEE.

Address for Correspondence: Hadas Kress-Gazit, GRASP Laboratory, University of Pennsylvania, Philadelphia, PA 19104 USA. E-mail: hadaskg@grasp.upenn.edu.