Valet Parking Without a Valet

David C. Conner†, Hadas Kress-Gazit‡, Howie Choset†, Alfred A. Rizzi†, and George J. Pappas‡
†Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213
‡GRASP Laboratory, University of Pennsylvania, Philadelphia, PA 19104
Email: {dcconner,choset,arizzi}@ri.cmu.edu {hadaskg,pappasg}@grasp.upenn.edu

***Abstract*— What would it be like if we could give our robot high level commands and it would automatically execute them in a verifiably correct fashion in dynamically changing environments? This work demonstrates a method for generating continuous feedback control inputs that satisfy high-level specifications. Using a collection of continuous local feedback control policies in concert with a synthesized discrete automaton, this paper demonstrates the approach on an Ackermann-steered vehicle that satisfies the command "drive around until you find an empty parking space, then park." The system reacts to changing environmental conditions using only local information, while guaranteeing the correct high level behavior. The local policies consider the vehicle body shape as well as bounds on drive and steering velocities. The discrete automaton that invokes the local policies guarantees executions that satisfy the high-level specification based only on information about the current availability of the nearest parking space. This paper also demonstrates coordination of two vehicles using the approach.**

I. Introduction

A major goal in robotics is to develop machines that perform useful tasks with minimal supervision. Instead of specifying each small detail, we would like to describe the high level task, and have the system autonomously execute in a manner that satisfies that desired task. Unfortunately, constraints inherent in mobile robots, including nonlinear nonholonomic constraints, input bounds, obstacles/body shape, and changing environments, interact to make this a challenging problem.

This paper presents an approach that addresses this challenge by combining low-level continuous feedback control policies with a formally correct discrete automaton. The composition and execution strategy is guaranteed to satisfy the high-level behavior for any initial state in the domain of the low-level policies. The approach can handle robots with complex dynamics as well as a variety of nonholonomic constraints. It allows the robot to react to local information during the execution, and supports behaviors that depend on that information. Furthermore, given a collection of local feedback control policies, the approach is fully automatic and correct by construction.

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.

This work is partially supported by National Science Foundation EHS 0311123 and Army Research Office MURI DAAD 19-02-01-0383.

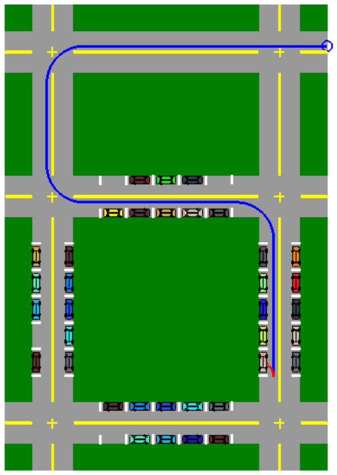


Fig. 1. The environment has 40 parking spaces arranged around the middle city block. The high-level specification encodes "drive around until you find a free parking space, and then park." This example shows the path taken from the time the vehicle enters the area until it parks in the first open parking space it encounters. There are four remaining open spaces.

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. The system continuously executes the automaton based on the state of the environment and the vehicle by activating the continuous policies. This execution guarantees the robot will satisfy its intended behavior.

As a demonstration of the general approach, this paper presents a familiar example: a conventional Ackermann steered vehicle operating in an urban environment. Figure 1 shows the environment, and the results of one simulation run. This run is a continuous execution of an automaton that satisfies the high-level specification "drive around the environment until you find a free parking space, and then park." The paper discusses the design and deployment of the local feedback policies (Section II), the automatic generation of automata that satisfy high level specifications (Section III), and the continuous execution (Section IV). While there has been work done in the past concerning a self parking car [1], and nowdays such a car is even commerically available, this paper focuses on the more general problem of specifying the high level behaviors, and can capture richer and more involved tasks.

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], [5], [6], [7], [8]. This paper

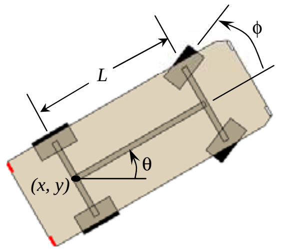


Fig. 2. Car-like system with Ackermann Steering. The inputs are forward velocity and steering angle velocity.

extends the prior work in sequential composition to a more complex system model by considering Ackermann steering, input bounds, and the shape of the vehicle.

Building upon the sequential composition [2] idea, 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], [11], [12] powerful model checking tools were used to find the sequence in which the controllers must be activated in order for the system to satisfy a high level temporal behavior. While 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. The planning community is dealing with "temporally extended goals" as well [13].

This work builds on the approach taken in [14] which is based on an automaton synthesis algorithm introduced in [15]. There, an automaton was created which enabled the robot to satisfy reactive tasks. This paper lifts several restrictions imposed in [14]. Here the robot model is no longer an idealized, fully actuated point robot and there is no need to partition the workspace into polygonal cells. Furthermore, the automaton execution is different, allowing for "interrupt" type inputs that can induce behavior changes at any time. This extension of [14] allows one to specify "safety critical" tasks such as emergency stop. This paper also allows for uncertainty in the initial position of the vehicle, represented as a set of initial conditions rather than a single one.

II. 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.

A. System Modeling

This paper focuses on the control of a rear-wheel drive car-like vehicle with Ackermann steering, shown schematically in Figure 2. The two rear wheels provide the motive force via traction with the ground; the two front wheels provide steering.

The vehicle pose, g, is represented as $g = \{x, y, \theta\}$; (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 $\phi \in (-\phi_{\max}, \phi_{\max})$, a bounded interval.

The nonholonomic constraints inherent in the rolling contacts uniquely specify the equations of motion via a non-linear 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

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ \frac{1}{L}\tan\phi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{1}$$

The system evolution is also subject to configuration constraints. The body pose is constrained by the interaction of body shape with obstacles in the environment. The pose is further constrained by local conventions of the road, such as driving in the right lane. 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.

B. Local Policy Development

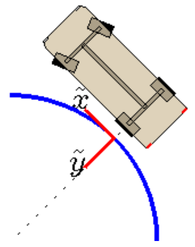
The hybrid control framework uses local feedback control policies to guarantee behavior over a local domain. These local policies are then composed in a manner that allows reasoning on a discrete graph to determine the appropriate policy ordering that induces the desired global behavior. In order for the policies to be composable in the hybrid control framework, the individual policies must satisfy several requirements: *i)* domains lie completely in the free configuration space of the system, *ii)* under influence of a given policy the system trajectory must not depart the domain except via a specified goal set, *iii)* the system must reach the designated goal set in finite time, and *iv)* the policies must have efficient tests for domain inclusion given a known configuration [3]. This paper 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 paper 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.

The approach to defining the cell boundary and desired steering angle is based on a variable structure control approach [16]. The cells are parameterized by a path segment in the workspace plane, as shown in Figure 3-a. The path is lifted to a curve in body pose space by considering the path tangent vector orientation as the desired orientation. One end of the curve serves as the goal set center.

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 [16]. To generate a continuous steering command, the



(a)

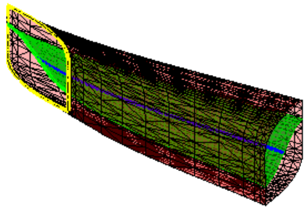

(b)

Fig. 3. Control policy based on [16]: a) workspace path with local frame defined, b) the cell boundary forms a "tube" around the curve in pose space. The sliding surface is shown in the cell interior.

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 $|\phi_{\text{lim}}| \leq \phi_{\text{max}}$. The sign of ϕ_{lim} depends on the current direction of travel (forward/reverse) and whether the current body pose in local coordinates is above or below the sliding surface. 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 space. Given this constraint, a simple constrained optimization is used to find a valid input. Each policy is verified to insure that a valid input exists over its entire domain.

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 $|\phi_{\text{des}} - \phi| < \phi_{\text{margin}}$; otherwise, 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; that is, 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.

C. Local Policy Deployment

To set up the basic scenario, we define the urban parking environment shown in Figure 1. The regularity of the environment allows an automated approach to policy deployment.

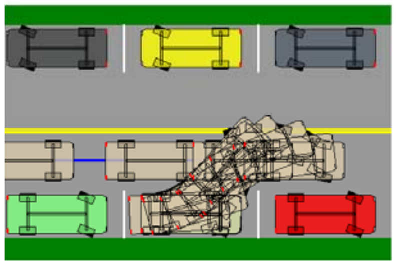


Fig. 4. Parking behavior induced by the composition of local policies. The feedback control policies guarantee the safety of the maneuver.

First, we specify a *cache* of local policies using the generic policy described above. The cache uses a total of 16 policies: a 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 deployed, 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 straight traffic lanes, the grid points lie in the middle of the traffic lanes aligned with the front of the parking space markers; orientation is defined by the traffic flow.

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, each parking policy is actually composed of several policies from the cache. The parking component policies are only instantiated if the parking behavior is invoked by the global parking automaton (Section III); at this point execution control switches to a local parking controller encoded as a partial order of the parking policies. Figure 4 shows an example parking maneuver induced by the composition of the local feedback control policies. For the region defined in Figure 1, there are a total of 306 instantiated policies, including 40 parking policies associated with the 40 possible parking spaces.

As part of the instantiation process, we test for goal set inclusion pairwise between policies. 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 *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 policies in the cache are specially defined so that policies instantiated at neighboring grid points prepare one another appropriately. The policy specification, instantiation, and prepares testing is done off-line, prior to automaton synthesis.

III. Automaton Synthesis

This section describes the method used to create the automaton that governs the local policies' switching strategy. This automaton is guaranteed to produce paths, if they exist, that satisfy a given specification.

A. The Synthesis Algorithm

We are given a set of binary inputs (e.g. whether the closest parking spot is empty), a set of outputs (e.g. whether or not to activate policy Φ_i), and a desired relationship between the two (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.

When the relationship is given in Linear Temporal Logic (LTL) [9], it is proven that the complexity of the synthesis problem is doubly exponential in the size of the formula [17]. However, by restricting ourselves to a subset of LTL, as described in Section III-B, we can use the efficient algorithm recently introduced in [15]. This algorithm is polynomial in the number of possible states. We present an informal overview of the algorithm, and refer the reader to [15] for a full description.

The synthesis process is viewed as a game played between the system, 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 a formula σ encoded with a fragment of LTL. 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.

The synthesis algorithm [15] takes the initial conditions, transition relations, and winning condition, 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 in order to satisfy the desired behavior.

B. Writing Logic formulas

In this work we use Linear Temporal Logic (LTL) formulas. We refer the reader to [9] for a formal description of this logic. Informally, these logic formulas are built using a set of boolean propositions, the regular boolean connectives 'not'($\neg$), 'and'($\wedge$), 'or' ($\vee$) 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, that is, 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,

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

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 environement 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 the reader is referred to [14].

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 later 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. The implication of this is discussed in Section IV.

C. Parking formula

In our basic scenario, a vehicle is searching for an empty parking space, and parks once it finds one; therefore we define one input, 'park', which becomes true when an empty parking space is found. The policy, Φ_i, to be activated is an output[1].

1) Assumptions on the environment: Initially there is no parking near the vehicle therefore $\varphi_i^e = \neg park$.

We can only determine whether there is a free parking space if we are in a policy next to it. This means that 'park' cannot become true if the vehicle is not next to a parking space or in one. Also, for implementation reasons, we assume that the input 'park' remains true after parking.

$$\varphi_t^e = \left\{ \begin{array}{c} \Box(\ [\ \ (\neg(\vee_{i \in ParkPolicy} \Phi_i)) \wedge \\ (\neg(\vee_{j \in PreparesParkPolicy} \Phi_j))] \\ \Rightarrow \ \neg \bigcirc park \qquad\qquad) \\ \bigwedge \Box(\ (park \wedge (\vee_{i \in ParkPolicy} \Phi_i)) \ \Rightarrow \ \bigcirc park \ ) \end{array} \right.$$

We have no assumptions on the infinite behavior of the environment (we do not assume there is an empty parking spot), therefore $\varphi_g^e = \Box\Diamond(TRUE)$.

2) Constraints on the behavior of the vehicle (system): Initially the vehicle must be in the domain of an initial policy, $\varphi_i^s = \vee_{i \in InitialPolicy} \Phi_i$.

The allowable transitions are encoded as

$$\varphi_t^s = \left\{ \begin{array}{l} \bigwedge_i \ \Box(\Phi_i \ \Rightarrow \ (\bigcirc\Phi_i \vee_{j \in SuccessorsOfPolicy_i} \bigcirc\Phi_j)) \\ \bigwedge_{i \in ParkPolicy} \ \Box(\neg \bigcirc park \ \Rightarrow \ \neg \bigcirc \Phi_i) \\ \bigwedge \ \Box(\bigcirc park \ \Rightarrow \ (\vee_{i \in ParkPolicy} \bigcirc \Phi_i) \ ) \end{array} \right.$$

The first line encodes the transitions of the prepares graph from Section II-C. The vehicle cannot park if there is no parking space available, as indicated by the 'park' input on the second line. The third line states that if there is an empty parking space, it must park; removing this line may allow the vehicle to pass an open spot before parking. Additional outputs can be added to the transition relation. For example, policies that

[1]For ease of reading, we define a different output for each policy. In the actual implementation we encode the policy numbers as binary vectors.

cause left or right turns can trigger appropriate signals. The initial signal status should also be set in φ_i^s.

Finally for the goal, we add a list of policies the vehicle must visit infinitely often if it has not parked yet, thus $\varphi_g^s = \wedge_{i \in VisitPolicy} \Box\Diamond(\Phi_i \vee park)$. These policies define the area in which the vehicle will look for an available parking space. 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.

IV. Continuous execution of discrete automata

The synthesis algorithm of Section III-A 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.

A. 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[2], the values of the binary inputs are evaluated. Based on 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 it's 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].

B. 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/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.

[2]The policies are designed as continuous control laws; however, the implementation on a computer induces a discrete time step. We assume the time step is short compared to the time constant of the closed-loop dynamics.

V. Results

The approach is verified in a simulation executed using MATLAB$^{\text{TM}}$. 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, based on the desired scenario, an LTL formula is written. The LTL formula is then given to the automatic synthesis algorithm implemented by Piterman, Pnueli and Sa'ar [15] on top of the TLV system [18]. At this point, the resulting automaton is used to govern the execution of the local policies, based on the local behavior of the environment.

In the following examples, the workspace is the one shown in Figure 1, with the 306 policies instantiated as described in Section II-C. In the LTL formulas, the visit policies correspond to the 8 lanes around the parking spaces (4 going clockwise and 4 going counter clockwise), and the initial policies correspond to the 10 entry points to the workspace. Initially, 35 of the 40 parking spaces were randomly specified as occupied.

A. Basic parking scenario

The basic parking scenario corresponds to the LTL formula described in Section III-C. For each run, a new vehicle was introduced at a random entrance, while the parking spaces were filled according to the previous run. As the automaton executes, if a parking policy is a successor to the current state, the empty/occupied status is checked via a local sensor. This work does not address the required sensor, but assumes a binary output. Transition to the parking policy is enabled if the associated space is empty. If the transition is enabled, other transitions are disabled until the vehicle pose enters the domain of the parking policy, at which point the control shifts to the local parking controller.

Six runs were simulated using the global parking automaton; Figure 5 shows the results for two of these runs. In Run #4 the vehicle parks in the first available parking space. In Run #6, there are no parking spaces available; therefore, the vehicle continues to circle past every possible parking space, waiting on another vehicle to leave.

B. Hazard

To provide more expressive behavior, we define an additional input 'hazard' that allows the vehicle to react to external changes in the environment not encoded in the instantiated policies. These hazards could include blocked roads or pedestrians detected using a proximity sensor or vision based system. We allow the input 'hazard' to change at any time and we require the vehicle to stop when 'hazard' is true, thus interrupting the execution of the feedback control policy. Once 'hazard' becomes false again, the vehicle resumes moving under feedback control. We do not show the exact additions to the basic formula due to space constraints.

To demonstrate this capability, we encoded a timed 'stop-light' at the intersections, and rewarded vehicle #6's patience

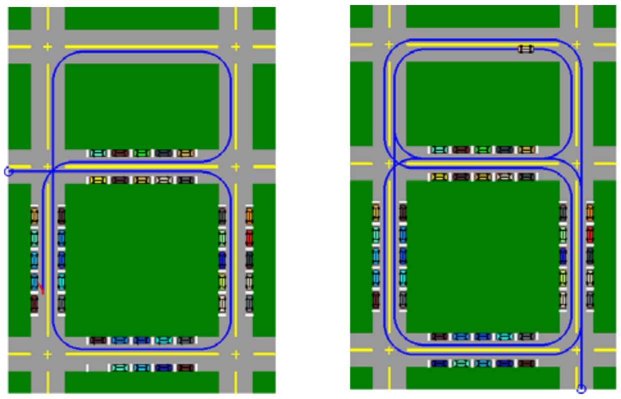


Fig. 5. Two executions of the basic scenario. The initial conditions for each run are circled. The first five executions successfully find a parking space; the last execution continues to loop as no parking spaces are available.

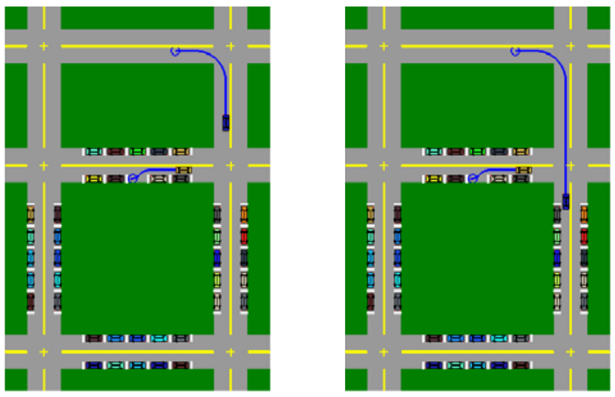


Fig. 6. Two snap shots of the multiple vehicle scenario. The vehicle heading east stops in response to the timed hazard signal to allow the other vehicle to travel through the intersection.

by having one vehicle leave its parking space and exit the area. The leaving behavior is encoded as a new automaton with an exit path as its goal. The 'stop-light' behavior is coded with an external timer that raises a hazard flag with policies that enter an intersection in a particular direction. When the timer expires, the hazard flag is raised for the crossing lanes, and after another brief period the original hazard flags are lowered allowing the vehicles to cross. The cycle repeats periodically.

Figure 6 shows the continuation of Run #6 with the hazard inputs added to the parking automaton, and a 'leaving automaton' with hazards added to control the second vehicle. This gives a rudimentary form of multi-vehicle coordination. In the first snapshot, vehicle #6 is just beginning to approach the intersection, while vehicle #7 stops for the light. The second snapshot shows vehicle #7 dutifully waiting for the signal, while vehicle #6 has passed through the intersection. Although not shown, after the 'stop-light' changes, vehicle #7 exits the area and vehicle #6 continues around under the control of the global parking automaton and parks in the newly open spot.

VI. Conclusions and Future Work

In this paper we have demonstrated, through the parking example, how high level specifications containing multiple temporally dependent goals can be given to a realistic robot, which in turn automatically satisfies them. We synthesized an automaton that forces the vehicle to park, turns the vehicle signal lights on and off appropriately, and obeys hazard conditions. By switching between low level feedback control policies and moving in a "well behaved" environment, the correctness of the 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.

We plan to extend this work in several directions. At the low level, we wish to consider more detailed dynamics. At the high level, we intend to formally address multiple robot coordination with more complex traffic conditions, and formally verify that the system avoids deadlock. Our research also focuses on accessible specification languages such as some form of natural language. We are currently running experiments with a real system to demonstrate the ideas shown in this paper.

References

[1] I. Paromtchik, P. Garnier, and C. Laugier, "Autonomous maneuvers of a nonholonomic vehicle," in *International Symposium on Experimental Robotics*, Barcelona , Spain, 1997.
[2] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, "Sequential composition of dynamically dexterous robot behaviors," *International Journal of Robotics Research*, 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 *Proceedings of Robotics:Science and Systems II*, Philadelphia, PA, 2006.
[4] A. A. Rizzi, "Hybrid control as a method for robot motion programming," in *IEEE International Conference on Robotics and Automation*, vol. 1, May 1998, 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 *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, Las Vegas, NV, October 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 Transactions on Robotics and Automation*, vol. 20, no. 3, 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 Transactions on Robotics*, vol. 21, no. 5, pp. 864–874, October 2005.
[8] S. R. Lindemann, I. I. Hussein, and S. M. LaValle, "Realtime feedback control for nonholonomic mobile robots with obstacles," in *IEEE Conference on Decision and Control*, San Diego, CA, 2006.
[9] E. A. Emerson, "Temporal and modal logic," in *Handbook of theoretical computer science (vol. B): formal models and semantics*. Cambridge, MA, USA: 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 International Conference on 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 Conference on Decision and Control*, Seville, Spain, 2005.
[12] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from LTL specifications," in *9th International Workshop on Hybrid Systems: Computation and Control*, Santa Barbara, California, 2006.
[13] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, , and P. Traverso, "MBP : A model based planner," in *In Proc. IJCAI'01 Workshop on Planning under Uncertainty and Incomplete Information*, 2001.
[14] 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*, 2007, pp. 3116–3121.
[15] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of Reactive(1) Designs," in *VMCAI*, Charleston, SC, January 2006, pp. 364–380.
[16] A. Balluchi, A. Bicchi, A. Balestrino, and G. Casalino, "Path tracking control for Dubin's car," in *IEEE International Conference on Robotics and Automation*, Minneapolis, MN, 1996, pp. 3123–3128.
[17] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *POPL '89: Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM Press, 1989, pp. 179–190.
[18] A. Pnueli and E. Shahar, "The TLV system and its applications," 1996. [Online]. Available: http://www.cs.nyu.edu/acsys/tlv/