Automatic Deployment of Autonomous Cars in a Robotic Urban-Like Environment (RULE)

M. Lahijanian, M. Kloetzer, S. Itani, C. Belta, and S. B. Andersson

***Abstract*— We present a computational framework and experimental setup for deployment of autonomous cars in a miniature Robotic Urban-Like Environment (RULE). The specifications are given in rich, human-like language as temporal logic statements about roads, intersections, and parking spaces. We use transition systems to model the motion and sensing capabilities of the robots and the topology of the environment and use tools resembling model checking to generate robot control strategies and to verify the correctness of the solution. The experimental setup is based on Khepera III robots, which move autonomously on streets while observing traffic rules.**

I. Introduction

In formal analysis, finite models of computer programs and digital circuits are checked against rich specifications given as temporal logic statements about the satisfaction of properties of interest. Examples include safety, *i.e.,* something bad *never* happens and liveness, *i.e.,* something good *eventually* happens. Such specifications translate naturally to formulas of temporal logics, such as Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) [1]. Due to their resemblance to natural language, expressivity, and existence of off-the-shelf algorithms for model checking, temporal logics are used to specify properties of other types of dynamic systems [2], [3], [4], including robotic systems [5], [6], [7]. In these works, in order to use a temporal logic as a specification language and a model checking algorithm for analysis, the main challenge is to construct a property-preserving finite abstraction of an infinite model [8].

A particularly important application in mobile robotics is deployment of autonomous cars in urban environments. In such a scenario, an autonomous car equipped with sensing, communication, and computation capabilities is required to accomplish a rich temporal logic motion specification, *e.g.,* "keep surveying roads R_1 and R_2 until an empty parking space is found, and then park", while at the same time obeying traffic laws, staying in lane, and avoiding collisions with other moving cars or static obstacles for all times. The importance of this problem led to the 2007 DARPA Urban Challenge [9], where autonomous cars were provided with a rough map of a city and required to autonomously drive from an initial to a final point while staying on the road and obeying traffic laws.

This work was supported by NSF under grants IIS-0822845 and IIS CAREER-0447721 at Boston University.

M. Lahijanian, M. Kloetzer, C. Belta, and S. B. Andersson are with the College of Engineering at Boston University, {morteza,kmarius,cbelta,sanderss}@bu.edu.

S. Itani is with the School of Engineering at Massachusetts Institute of Technology, itani@mit.edu.

M. Lahijanian is the corresponding author.

In this paper, we present a computational framework and an experimental setup for deployment of autonomous cars in a miniature Robotic Urban-Like Environment (RULE) (see Fig. 1 and the project web site http://iasi.bu.edu/rule/). In our setup, Khepera III car-like robots can be automatically deployed according to arbitrarily rich temporal logic specifications about visiting points of interest in the environment such as streets, intersections, and parking spaces, while at the same time being guaranteed to stay in their lanes, obey traffic rules, and avoid collisions with other cars and obstacles. An example of such a specification is "Visit Road R_1 and then Road R_9 infinitely often. If Road R_8 is ever visited, then Road R_3 should never be reached." In our framework, such specifications given in human-like language translate to formulas of LTL. A modelling framework based on transition systems and synchronous products [10] allows for automatic generation of motion plans and robot control laws by using tools resembling LTL model checking [1].

This paper is closely related to [7], where a computational framework for automatic deployment of car-like robots from temporal logic specifications was developed. While this paper and [7] approach the same problem, the solution presented here allows for more expressivity in the specification. Specifically, any LTL formula is allowed as a specification in our framework, while [7] are limited to formulas in the GR(1) fragment of LTL [11]. On the other hand, our framework cannot capture environmental events as in [7]. The use of synchronous products in our work allows for an extension to the multi-robot case, which is left for future work. All these advantages come at the expense of an increased amount of computation. In this paper we focus on a purely discrete scenario, where the motion of the robot is described from the very beginning as a finite state transition graph. However, the implementation to simple continuous dynamics is immediate by using our previous results on discrete abstractions [4], [12].

II. Preliminaries

Definition 1: A transition system with observations is a tuple $T = (Q, Q_0, \Sigma, \rightarrow, \Pi, \models)$, where Q is a finite set of states, $Q_0 \subseteq Q$ is the set of initial states, Σ is a finite set of actions (inputs, events), $\rightarrow \subseteq Q \times \Sigma \times Q$ is a transition relation, Π is a set of atomic propositions (properties, observations), and $\models \subseteq Q \times \Pi$ is a satisfaction relation.

A transition system T is called *deterministic* if, for all $\sigma \in \Sigma$ available at an arbitrary state $q \in Q$, there exists a unique $q' \in Q$ such that $q \xrightarrow{\sigma} q'$. For a transition system

T, we denote by $R(T)$ the transition system obtained by keeping only the states from T that are reachable from the set of initial states Q_0 (and the corresponding transitions).

Definition 2: Let $T_i = (Q_i, Q_{0i}, \Sigma, \rightarrow_i, \Pi_i, \models_i)$, $i = 1, \ldots, k$ be a set of transition systems with the same set of inputs. The synchronous product of T_i, $i = 1, \ldots, k$, denoted by $T_1 \parallel \ldots \parallel T_k$, is the transition system $T = R((Q, Q_0, \Sigma, \rightarrow, \Pi, \models))$ defined by:

- $Q = Q_1 \times \ldots \times Q_k$;
- $Q_0 = Q_{01} \times \ldots \times Q_{0k}$;
- $((q_1, \ldots, q_k), \sigma, (q'_1, \ldots, q'_k)) \in \rightarrow$ if $(q_i, \sigma, q'_i) \in \rightarrow_i$;
- $\Pi = \cup_{i=1,\ldots,k} \Pi_i$;
- $(q_1, \ldots, q_k) \models \pi$ if there exists $i = 1, \ldots, k$ such that $q_i \models_i \pi$.

A more general definition of a synchronous product, which allows for different sets of inputs for the transition systems T_i, can be found in [13].

In this paper, we consider motion specifications given in rich, human-like language that translate immediately to Linear Temporal Logic (LTL) formulas. Informally, LTL formulas are recursively defined over a set of atomic propositions Π, by using the standard Boolean and temporal operators, which include $\bigcirc$ ("next"), $\mathcal{U}$ ("until"), $\Box$ ("always"), $\Diamond$ ("eventually"). LTL formulas are interpreted over infinite words in the power set 2^{Π} of Π, as are those generated by the transition system T from Definition 1.

Given a finite transition system T and an LTL formula ϕ over Π, checking whether the words of T starting from each state in Q satisfy ϕ is called LTL model checking, or simply model checking in this paper. An off-the-shelf model checker such as NuSMV [14] takes a transition system T and a formula ϕ as input and returns the states of T at which the formula is satisfied (*i.e.,* the states for which the language originating there satisfies the formula). For the non-satisfying states, a model checker returns a non-satisfying run as a certifying counter-example.

Alternatively, a dual control problem can be formulated for a transition system T: given an LTL formula ϕ over its set of propositions, find a set of initial states and a control strategy such that all the words produced by T satisfy the formula. We approached this problem in our previous work, and proposed solutions for both deterministic and nondeterministic systems [4], [15]. For deterministic systems, the solution is presented in the form of a "shortest" satisfying run with a prefix-suffix structure, which determines a unique control strategy.

III. Problem Formulation and Approach

We define an *Urban Environment* as a collection of *Roads*, *Intersections*, *Traffic Lights*, and *Parking Spaces* with the following restrictions: (1) a road connects two (not necessarily different) intersections; (2) the parking spaces can only be located on the side of (each bound of) a road; (3) there is a traffic light for each bound of a road arriving at an intersection; (4) at each intersection, the traffic lights are synchronized in the usual way (traffic lights corresponding to opposing roads are in phase and in opposite phase with traffic lights on orthogonal roads). An example is given in Fig. 2, which is a schematic representation of the miniature city platform shown in Fig. 1.

The "cars" in our city are Khepera III miniature robots (Sec. VI-A). We assume that each car can (i) drive forward on a road while staying in the right lane, (ii) "sense" intersections and parking spaces, (iii) turn left, right, or move straight through an intersection, (iv) detect the right lane of a road at the end of an intersection, and (v) distinguish the color of a traffic light (Sec. VI-A). We want to be able to automatically generate robot control strategies from rich specifications given as temporal logic statements about visiting roads, intersections, and parking spots. Specifically, we consider the following problem:

Problem 1: Given an urban environment with a set of roads $\mathcal{R}$, set of intersections $\mathcal{I}$, set of parking spaces $\mathcal{P}$, and a motion task in terms of an arbitrary LTL formula over the set $\mathcal{R} \cup \mathcal{I} \cup \mathcal{P}$, find a robot control strategy such that the produced motion of the robot satisfies the specification.

For example, for the environment shown schematically in Fig. 2, a motion task can be of the form "Visit R_1 and then eventually R_9 in this order, infinitely often. If R_8 is ever reached, make sure that R_3 is never visited." The translation of this human-like language specification to an LTL formula is given in Eqn. (1). To complete the formulation of Problem 1, we need to define a robot control strategy and the satisfaction of an LTL specification by a motion of a robot in the environment. Both of these definitions will be given after introducing robot and environment models in the form of transition systems in Sec. IV.

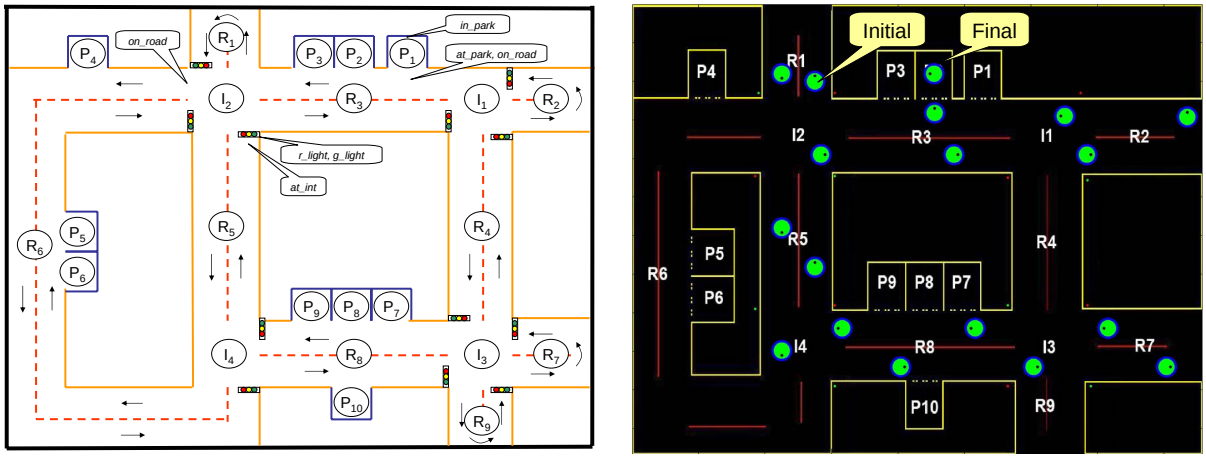
Our approach to Problem 1 is as follows. We first model the motion and sensing capabilities of the robot as a robot transition system T_R. The states of T_R model robot behaviors (*e.g.* driving in a lane, turning right in an intersection), while the inputs labelling transitions capture events generated by the environment (*e.g.,* traffic light turning green, parking spot being detected on the side of the road) or decisions made by the robot. The topology of the environment is captured by an environment transition system T_E. The states of T_E are features of interest in the environment, such as roads, intersections, and parking spaces. The inputs labelling the transitions of T_E are the same as the inputs of T_R. All possible motions of a robot in the environment will be given by the synchronous product $T_R \parallel T_E$. In this transition system , we will find a "short" satisfying run (Sec. II), project it to a run of T_R, and then implement this run as a control strategy for the robot.

IV. Robot and Environment Models

A. Robot Transition System

The robot transition system $T_R = (Q_R, Q_{0R}, \Sigma, \rightarrow_R, \Pi_R, \models_R)$ is shown in Fig. 3. The ovals are the states Q_R, the arrows and their labels are the set of transitions $\rightarrow_R$ and the set of events Σ, the observations Π_R are shown close to the states with which they relate according to the satisfaction relation $\models_R$. The initial states Q_{0R} are marked by incoming arrows with no labels.

Fig. 1. Robotic Urban-Like Environment (RULE). Left: Khepera III car-like robots move autonomously on streets while staying in their lanes, observing traffic rules, and avoiding collisions. Right: A car waiting at a traffic light.



(a) (b)

Fig. 2. (a) A schematic representation of the city environment from Fig 1 showing the observations and the location of some environmental events; (b) Snapshots from a movie produced with our simulator showing a robot executing the motion $\mathrm{R}_1\mathrm{I}_2\mathrm{R}_5\mathrm{I}_4\mathrm{R}_8\mathrm{I}_3\mathrm{R}_7\mathrm{I}_3\mathrm{R}_8\mathrm{I}_4\mathrm{R}_5\mathrm{I}_2\mathrm{R}_3\mathrm{I}_1\mathrm{R}_2\mathrm{I}_1\mathrm{R}_3(\mathrm{P}_2)$ satisfying the specification "Visit R_7, and then park at P_2 and stay there; always avoid R_4", or $\Diamond(R_7 \wedge \Diamond\Box P_2) \wedge \Box\neg R_4$.

Each state models a behavior or a collection of related robot behaviors where each is implemented as a low-level feedback controller. There is a transition from a robot state to another once an event (input) is triggered by the environment or the robot itself. For instance, in state **Drive**, the robot moves at constant speed in a lane while looking for intersections and parking spaces (Sec. VI-A). When an intersection or a parking space is found, the event *at_int* or *at_park* is generated, and the robot transits to the next state.

The robot is initiated either in **Parking Wait** or **Drive**. If the robot is driving on a street and reaches an intersection, the event *at_int* is triggered, which forces the robot to transfer to the **Intersection Wait** state. Here the robot checks for the intersection signal light color. If it is red (*r_light*), the robot stays in the same state and continues to check the color of the signal light. Once the light turns green (*g_light*), the robot decides which way to go (*go_left*, *go_right*, or *go_straight*.) For more details on the robot's actions such as parking, see its transition system graph (Fig. 3).

For the case when the robot gets dangerously close to an obstacle or to another robot, each robot state is paired with a **Wait** state, which halts the robot and prevents it from moving. The transition to this state is enabled by the event *hazard*, which is triggered by close proximity to an object. The transition back to the original state is enabled by *no_hazard*, which is generated when the space near the robot is cleared. For simplicity, these extra states are not shown in Fig. 3.

The states of T_R are put in relation with observations from the set $\Pi_R = \{\mathrm{DR}, \mathrm{DI}, \mathrm{PA}, \mathrm{WI}\}$, with the following significance: DR = "Drive on a Road", DI = "Drive through an Intersection", PA = "involved in PArking", and WI = "Wait at an Intersection". These observations are high-level descriptions of the robot behavior. In our current setup, they are not used for motion specification and control. Rather, they are used to check the correctness of the solution. For example, over the set of all possible robot motions in the environment, we want to make sure that the robot is not involved into a parking maneuver (PA) while passing through an intersection (Sec. V).

B. Environment Transition System

For any urban environment satisfying the assumptions enumerated at the beginning of Sec. III, an environment transition system $T_E = (Q_E, Q_{0E}, \Sigma, \rightarrow_E, \Pi_E, \models_E)$ can be constructed by interconnecting the three modules shown in

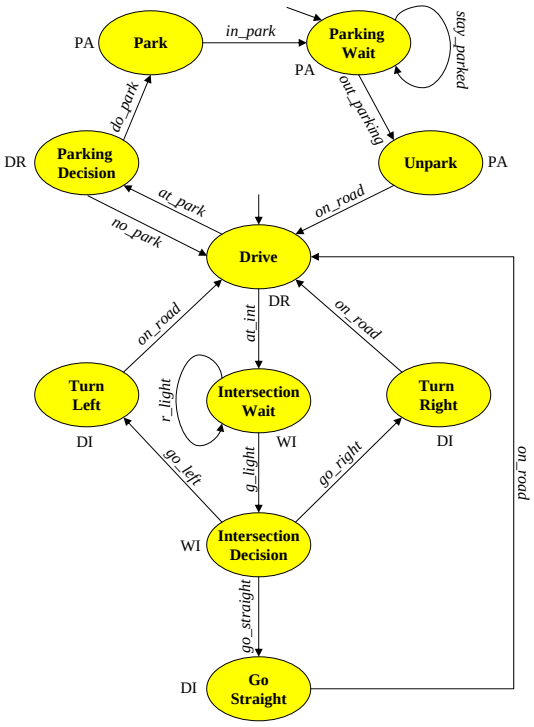


Fig. 3. The robot transition system (T_R). The states label robot behaviors (*e.g.,* **Drive**, **Park**). The transitions are enabled by inputs that can be robot decisions (*e.g., go_left*) or environmental events (*e.g., at_int*). The observations (*e.g.,* DR = "Drive on a road", DI = "Drive through an intersection") are high-level descriptions of the robot behavior.

Figs. 4 and 5. The connections are made along transitions marked with $\cdots$ by matching the name of the events on the transitions. There are three types of observations in Π_E: road labels $\mathrm{R_n}$ from the set $\mathcal{R}$, intersection labels $\mathrm{I_i}$ from the set $\mathcal{I}$, and parking space labels $\mathrm{P_k}$ from the set $\mathcal{P}$. Note that the motion specification in Problem 1 is given over these sets.

Any road $\mathrm{R_n}$, connecting exactly two intersections $\mathrm{I_i}$ and $\mathrm{I_j}$, can have an arbitrary number of parking spaces. A road $\mathrm{R_n}$ with no parking spaces connecting intersections $\mathrm{I_i}$ and $\mathrm{I_j}$ generates a (part of a) transition system obtained by connecting Module 1 at the right of Module 2, where in Module 2 the incoming transition labelled *no_park* in state **Int Prep** is removed. For a road $\mathrm{R_n}$ spanning between intersections $\mathrm{I_i}$ and $\mathrm{I_j}$ with one parking space $\mathrm{P_k}$, one Module 3 is inserted between a Module 1 and a Module 2, where the incoming transition labelled by *no_park* in **Park Prep** is removed. To insert an additional parking space, another Module 3 is inserted at the left of the previous Module 3, following the connection rule defined above. The last parking space (Module 3) is then connected to Module 2 to complete the construction of the road.

State **Int End** in Module 1 corresponds to a region in the environment at the end of intersection $\mathrm{I_i}$, and is therefore labelled with observation $\mathrm{I_i}$. In this state, the robot can read the event *on_road*, denoting that it is out of the intersection and on road $\mathrm{R_n}$ (Fig. 2 (a)). State **Int Prep** in Module 2 corresponds to the portion of road $\mathrm{R_n}$ where no parking spaces can be found any more, and the robot is looking for the beginning of an intersection, which is signalled by the event *at_int* (*e.g.,* the entrance from $\mathrm{R_5}$ to $\mathrm{I_2}$ in Fig. 2 (a)). The rest of the states in Module 2 are self explanatory.

Note that the intersections at the end of a road can be the same (*i.e.,* i=j in Fig. 4). By connecting road segments on their end transitions such that the labels match, any city satisfying the restrictions stated at the beginning of Sec. III can be constructed. For more details on T_E states and events, see Figs. 4 and 5.

V. Generation of a Robot Control Strategy

The synchronous product $T = T_R \parallel T_E = (Q, Q_0, \Sigma, \rightarrow, \Pi, \models)$ captures all possible motions of a robot in the city. This is achieved through synchronization on common events and through pruning unreachable states in the product (Defn. 2). A state (q_R, q_E) in T gives "complete" information about the robot behavior (q_R) and about its location in the environment (q_E). As already suggested, this information is, in general, too detailed, as some states in T_E were artificially introduced for synchronization purposes only. A coarser description is the corresponding observation $\{\pi_R, \pi_E\}$. According to our definitions from Sec. IV-B and IV-A, π_E specifies on which road, intersection, or parking space the robot is located, while π_R indicates if the robot is driving on a road, passing through an intersection, or maneuvering or waiting in a parking space.

As stated in Problem 1, a task is specified as an LTL formula ϕ over $\Pi_E = \mathcal{R} \cup \mathcal{I} \cup \mathcal{P}$. This is also an LTL formula over $\Pi = \Pi_R \cup \Pi_E$. By using the procedure described at the end of Sec. II, we find a run in T of minimal overall length, where the overall length is defined as the sum of the lengths of the prefix and suffix. The projection of this run of T along T_E gives the motion of the robot in the environment. The projection along T_R (which is deterministic) gives the robot control strategy. Therefore, as an infinite run of T_R, the robot control strategy is a sequence of robot behaviors (states from Q_R), where the transitions between the behaviors are triggered by events read from the environment while the robot moves (*e.g., on_road*, *at_park*) or by robot decisions (*e.g., go_left*).

In the rest of this section, we discuss the issues of *completeness* (*i.e.,* whether a solution can be found when one exists) and *correctness* (*i.e.,* whether the solution corresponds to a safe and deadlock-free motion of the robot in the environment) of our approach to Problem 1. Intuitively, both these issues are addressed through the construction of T_R and T_E and through the use of the synchronous product from Def. 2. Indeed, T_R captures the motion and sensing capabilities of our robot, and only allows for correct transitions between the behaviors. T_E captures the topology of the environment, and transitions are only allowed between adjacent regions and under events that can be generated locally in the corresponding regions. In the synchronous product, the robot and the environment synchronize on such events, and the pairs of the states that are not reachable from the initial states are eliminated.

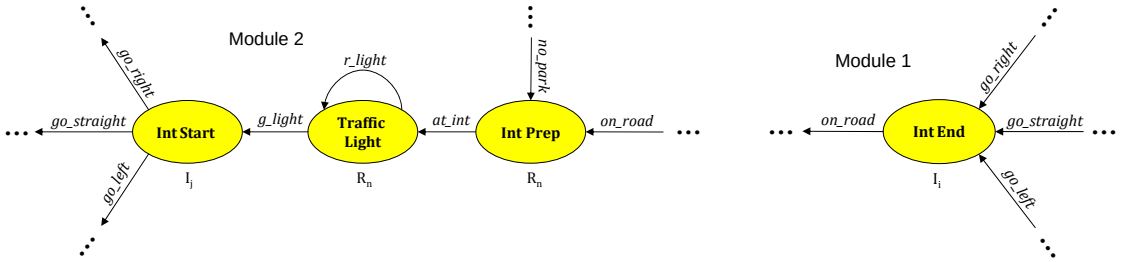


Fig. 4. Modules 1 and 2 necessary for the construction of roads and intersections in the environment transition system T_E

Even though this construction seems to lead to complete and correct solutions, automatic verification can prove useful, especially since the size of T_E can become large for "complicated" urban environments, and errors in coding and representations of T_E and T are possible. This can be achieved through off-the-shelf LTL model checking (Sec. II), where the formulas combine robot and environment observations. First, in accordance with the restrictions formulated at the beginning of Sec. III and the allowed modules shown in Figs. 4 and 5, we need to make sure that an intersection is always followed by a road and a parking space can only occur on a road. These requirements translate to the LTL formula $\bigcirc(\vee \mathrm{R_n})$, which must be true at all states with observations $\mathrm{I_i}$ or $\mathrm{P_k}$ in T_E, for all $\mathrm{I_i} \in \mathcal{I}$ and $\mathrm{P_k} \in \mathcal{P}$. Second, we want to make sure that, in the synchronous product T, robot behaviors are correctly paired with regions in the city. Using the robot observations from Fig. 3, this translates to the set of purely logical formulas $\mathrm{DR} \Rightarrow \bigvee \mathrm{R_n}$, $\mathrm{DI} \Rightarrow \bigvee \mathrm{I_i}$, and $\mathrm{PA} \Rightarrow \bigvee \mathrm{P_k}$. We performed these checks to verify our implementation for the environment shown in Fig. 2. It is important to note that these correctness checks are only for the models and the software implementation described here, and are valid under the assumptions that the robot does not break (its motors and sensors function properly), and that the events are "well behaved" in the sense that the robot reads events such as *on_road* and *at_park* correctly, each traffic light eventually turns green (event *g_light* occurs infinitely often) and the *hazard* event is eventually reset if it ever occurs. Such assumptions also translate to LTL formulas (not shown here due to space constraints), which can be used to augment the correctness formulas enumerated above. More thorough verification can be achieved by augmenting the set of observations of T_R and T_E.

By using the framework described above, several robots can be deployed at the same time in the same environment (Sec. VI-B). The traffic lights solve possible conflicts in intersections, while the **Wait** - *hazard* mechanism takes care of situations in which two robots are too close.

VI. Experimental Setup and Results

A. RULE Setup

Our Robotic Urban-Like Environment (RULE) is a 8' $\times$ 11' surface with two-way streets, traffic lights, and parking spaces (Fig. 1). The cars are Khepera III robots equipped with KorebotLE extensions. A desktop computer is used to control the traffic lights and to process the images captured with four overhead cameras. For more information on RULE setup, see the project web site (`http://iasi.bu.edu/rule/`).

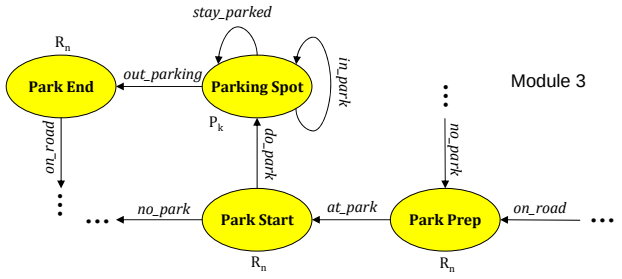


Fig. 5. Module 3 of T_E necessary for insertion of parking spaces on roads

The city is easily reconfigurable. The lanes of the roads are marked with tape (yellow for road sides and red for middle line, the posts of the traffic lights can be screwed in several predefined positions, and the parking spots can be easily relocated by simply moving the corresponding blue boxes (Fig. 1). The environment events *at_int* and *at_park* are implemented by using white tape of different widths on the road. While driving on a road, the robot continuously looks for the beginning of a white region using its bottom infrared sensors. When such a portion is found, the robot uses odometry to measure the width of the region (which is accurate enough for this task), and therefore distinguish between *at_int* and *at_park*. The event *r_light* (red color at a traffic light) is generated by an IR emitter LED, which can be read by the robot's IR proximity sensors when the robot is at the traffic light. The robot uses its bottom IR sensors to stay in a lane and to find a lane (event *on_road*) by following and, respectively, finding a transparent tape glued in the middle of the lane (Fig. 1). Finally, the robot uses its proximity sensors to detect the back wall of a parking space, when the event *in_park* is generated.

In RULE, robot deployment is achieved through a user-friendly graphical interface. The image of the city obtained from four overhead cameras is converted into a schematic representation, such as the one from Fig. 2. Labels are automatically generated for roads, intersections, and parking spaces and presented to the user, who can specify a task as an arbitrary LTL formula over these labels. The desktop computer performs all the computation (generation of the Büchi

automaton, construction of the synchronous product, construction of the product automaton between the synchronous product and the Büchi automaton, and the generation of the robot control strategy), and sends the control strategy to the robots through Wi-Fi. Then, the robots execute the task autonomously by interacting with the environment. The user has the option to simulate the run (Fig. 2 (b)) before trying it on the actual platform.

B. Experimental Results

Consider the RULE setup shown schematically in Fig. 2 and the following two motion specifications:

Spec. 1: "Visit Road R_1 and then Road R_9 infinitely often. If Road R_8 is ever visited, then Road R_3 must never be reached."

Spec. 2: "Visit Road R_7 and then Road R_5 infinitely often."

Specifications 1 and 2 translate immediately to the LTL formulas ϕ_1 and ϕ_2 below, respectively:

$$\phi_1 \quad : \quad \square\diamondsuit(\mathrm{R}_1 \wedge \diamondsuit\mathrm{R}_9) \wedge \square(\mathrm{R}_8 \rightarrow \neg\diamondsuit\mathrm{R}_3) \qquad (1)$$

$$\phi_2 \quad : \quad \square\diamondsuit(\mathrm{R}_7 \wedge \diamondsuit\mathrm{R}_5) \qquad (2)$$

Using the computational framework proposed in this paper, we find robot control strategies for each specification. For Spec. 1 and initial state R_2 in T_E, the resulting robot motion, which is a run in the environment transition system T_E, is the following:

$$\mathrm{R}_2\mathrm{I}_1\mathrm{R}_3(\mathrm{I}_2\mathrm{R}_1\mathrm{I}_2\mathrm{R}_5\mathrm{I}_4\mathrm{R}_8\mathrm{I}_3\mathrm{R}_9\mathrm{I}_3\mathrm{R}_8\mathrm{I}_4\mathrm{R}_6) \qquad (3)$$

Spec. 2 with initial state R_6 for T_E leads to the following motion:

$$\mathrm{R}_6\mathrm{I}_2(\mathrm{R}_3\mathrm{I}_1\mathrm{R}_4\mathrm{I}_3\mathrm{R}_7\mathrm{I}_3\mathrm{R}_8\mathrm{I}_4\mathrm{R}_5\mathrm{I}_2) \qquad (4)$$

In Eqns. (3) and (4), the parts of the runs between parenthesis represent the suffixes, and are therefore repeated infinitely. It should be noted that Spec. 1 does not require R_1 to be visited right after R_9. Similarly, Spec. 2 does not enforce visiting R_5 immediately after R_7. Thus, the produced runs are acceptable even though many roads and intersections are visited between R_1 and R_9 and between R_7 and R_5 in Eqns. (3) and (4) respectively. Movies showing the actual runs can be downloaded from the RULE web site (`http://iasi.bu.edu/rule/`).

The approach we propose here is computationally expensive. The amount of computation scales exponentially with the length of the specification formula. However, this theoretical upper bound is almost never achieved in practice. For example, in the case studies shown above, the robot and environment transition systems have 10 and 46 states respectively, and the computation of the runs takes less than a second on a regular desktop computer.

VII. Conclusions and Future Work

We presented a computational framework and experimental setup for deployment of autonomous cars in a Robotic Urban-Like Environment (RULE). We use transition systems to model the motion and sensing capabilities of the robots and the topology of the environment, formulas of Linear Temporal Logic (LTL) to allow for rich specifications, and tools resembling model checking to generate robot control strategies and to verify the correctness of the solution. The experimental setup is based on Khepera III robots, which move autonomously on streets while observing traffic rules.

As future work, we plan to accommodate changing environments and unexpected malfunctions in the environment by reformulating the problem as a game, where nondeterminism is treated as an adversary, as suggested in our previous work [15]. We will also investigate automatic generation of distributed robot control and communication strategies from global (robot-abstract) specifications about features of interest in the environment.

VIII. Acknowledgements

We would like to thank Ben Heng, Greg Vulkih, and Boyan Yordanov (Boston University) for their help with the construction of the platform and software implementation, Kim Wheeler (Road Narrows) for assisting with Khepera-related problems, and Thomas Lochmatter (EPFL) for his help with coding the KorebotLE.

References

[1] E. M. Clarke, D. Peled, and O. Grumberg, *Model checking*. MIT Press, 1999.

[2] J. Davoren, V. Coulthard, N. Markey, and T. Moor, "Non-deterministic temporal logics for general flow systems," in *HSCC: 7th International Workshop*, 2004, pp. 280–295.

[3] P. Tabuada and G. Pappas, "Model checking LTL over controllable linear systems is decidable," ser. Lecture Notes in Computer Science, O. Maler and A. Pnueli, Eds. Springer, 2003, vol. 2623.

[4] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Transactions on Automatic Control*, vol. 53, no. 1, pp. 287–297, 2008.

[5] S. G. Loizou and K. J. Kyriakopoulos, "Automatic synthesis of multiagent motion tasks based on LTL specifications," in *43rd IEEE Conference on Decision and Control*, 2004.

[6] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Hybrid controllers for path planning: a temporal logic approach," in *Proceedings of the 2005 IEEE Conference on Decision and Control*, 2005.

[7] H. Kress-Gazit, D. Conner, H. Choset, A. Rizzi, and G. Pappas, "Courteous cars," *IEEE Robotics and Automation Magazine*, vol. 15, pp. 30–38, March 2008.

[8] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, "Discrete abstractions of hybrid systems," *Proceedings of the IEEE*, vol. 88, pp. 971–984, 2000.

[9] (2007) 2007 darpa urban challenge. [Online]. Available: http://www.darpa.mil/GRANDCHALLENGE/

[10] A. Arnold, *Finite transition systems: semantics of communicating systems*. Prentice Hall, 2007.

[11] N. Piterman, A. Pnueli, and Y. Saar, "Synthesis of reactive(1) designs," in *VMCAI*, Charleston, SC, 2006, pp. 364–380.

[12] C. Belta and L. Habets, "Control of a class of nonlinear systems on rectangles," *IEEE Transactions on Automatic Control*, vol. 51, no. 11, pp. 1749 – 1759, 2006.

[13] M. Mukund, "From global specifications to distributed implementations," in *Synthesis and control of discrete event systems*. Kluwer, 2002, pp. 19–34.

[14] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking," in *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, ser. LNCS, vol. 2404. Copenhagen, Denmark: Springer, July 2002.

[15] M.Kloetzer and C. Belta, "Dealing with non-determinism in symbolic control," in *Hybrid Systems: Computation and Control: 11th International Workshop*, ser. Lecture Notes in Computer Science, M. Egerstedt and B. Mishra, Eds. Springer Berlin / Heidelberg, 2008, pp. 287–300.