The 2010 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 18-22, 2010, Taipei, Taiwan

# LTLMoP: Experimenting with Language, Temporal Logic and Robot Control*

Cameron Finucane, Gangyuan Jing and Hadas Kress-Gazit
Cornell University
Ithaca, NY 14853, USA
{cpf37,gj56,hadaskg}@cornell.edu

***Abstract*—The Linear Temporal Logic MissiOn Planning (LTLMoP) toolkit is a software package designed to assist in the rapid development, implementation, and testing of high-level robot controllers. In this toolkit, structured English and Linear Temporal Logic are used to write high-level reactive task specifications, which are then automatically transformed into correct robot controllers that can be used to drive either a simulated or a real robot. LTLMoP's modular design makes it ideal for research in areas such as controller synthesis, semantic parsing, motion planning, and human-robot interaction.**

***Index Terms*—Motion planning, mission planning, language, temporal logics, sensor-based planning, controller synthesis, hybrid control.**

## I. Introduction

The goal of controlling physical robots at an abstract, high level is one of the fundamental challenges of robotics. Nobody would dispute that it would be useful if we could tell a robot to "go find my glasses and bring them to me"; however, it is still unknown how to perform the sensing and control that will allow the robot to satisfy this task *correctly* and *safely* in a dynamic and uncertain environment.

Traditionally, different research communities have focused on different aspects of this problem. The artificial intelligence community typically addressed discrete planning problems, where the task is complex but the underlying dynamics of the problem (i.e. robot motion) are abstracted away [1]; the control and robotics communities, on the other hand, have typically looked at the problem of performing simple tasks (e.g. navigation between two points) in more physically realistic scenarios, such as ones with dynamic constraints and obstacles.

Lately, there has been a growing interest[1] in looking at theory and algorithms that bridge this gap between high-level behaviors and low-level control. Such techniques include the composition of learned motion primitives [2] and the use of hierarchical task networks [3]. Another approach for addressing this gap is the use of temporal logic [4] as the specification language, and the application of formal methods such as model-checking [5] and synthesis [6] or optimization techniques, in conjunction with hybrid system theory, to automatically generate controllers. This approach ( [7]–[11]) has several strengths, in that it generates *provably correct* robot controllers and it allows for a rich set of specifications that are able to capture notions such as system constraints, infinite behaviors, and sequencing.

This paper introduces the Linear Temporal Logic MissiOn Planning (LTLMoP) toolkit, which was written based on the work in [7], [12]. This toolkit is a modular, open-source software package for designing, implementing, and testing reactive hybrid controllers synthesized automatically from structured English behavioral specifications.

LTLMoP provides a complete development environment, encapsulating each step of the controller generation and implementation process—from parsing of specifications to continuous robot motion control—and thereby helps to bridge the gap mentioned above. At the same time, LTLMoP is designed to be modular, so that research can be performed on any single component (e.g. semantic parsing or controller synthesis) in isolation, while still benefiting from the integrated system. Furthermore, by treating the robot under control as an abstract interface, LTLMoP allows seamless transition from computer simulation to physical experiment, with the same task specification.

The software is written in Python and Java, and is thus cross-platform. With user-friendly GUIs available for most tasks, the toolkit facilitates experimentation even by those unfamiliar with the technical implementation details.

This paper is structured as follows: Section II briefly reviews some of the fundamental theory behind LTLMoP's functionality, Section III outlines the structure of LTLMoP's code, and Section IV presents an example task, its corresponding structured English description, and the execution of the resulting synthesized controller by a robot in our lab.

## II. Theoretical Background

As mentioned in Section I, LTLMoP was written based on the work described in [7], [12]. Conceptually, the process of transforming a high-level task given in structured English to correct control input for a robot is composed of three stages: parsing the structured English sentences into a formal logic formula defined over an abstraction of the problem, transforming the logic formula into an automaton, and executing the automaton as a hybrid controller where a transition between states corresponds to a basic continuous controller. In the following we give a brief overview of these three components.

*This work is supported by ARO MURI (SUBTLE) W911NF-07-1-0216.

[1]As evident from many recent workshops such as those at RSS '09, ICAPS '09, AAAI '10 and others.

### A. Abstraction and Parsing: Language to logic

The underlying formalism used to capture high-level tasks is Linear Temporal Logic (LTL [4]). Loosely speaking, LTL formulas are defined over sets of Boolean propositions, Boolean operators ($\wedge$ "AND", $\neg$ "NOT", $\vee$ "OR", $\Rightarrow$ "IMPLIES", etc.) and temporal operators ($\bigcirc$ "NEXT", $\Box$ "ALWAYS", $\Diamond$ "EVENTUALLY", $\mathcal{U}$ "UNTIL"). The inclusion of temporal operators means that LTL formulas can describe changes in the truth values of the propositions over time; for example, one can make statements such as "$p$ is always true" or "if $p$ is true then eventually $q$ will become true." The truth of an LTL formula is evaluated over a finite state machine that represents the system; the formula is true if it is true for every possible execution of the system.

In order to capture a continuous behavior (the motion and action of a robot in response to its environment) using a discrete formalism (LTL), the continuous behavior is abstracted using a finite set of propositions. These propositions correspond to sensor information (e.g. "object detected" could be the output of a perception algorithm), actions (e.g. "turn on video camera" or "transmit location") and locations. Note that, for sensor inputs, we assume the information is correct and that any noise or non-determinacy is filtered out by the low-level sensor processing routines. For locations, we decompose the workspace of the robot into convex polygons and assign a proposition to each; the proposition $porch$ would be true whenever the robot is on the porch, and false otherwise (see Section III-A2).

For computational reasons, we restrict ourselves to a subset of LTL as in [6].[2] LTLMoP includes a parser that automatically translates English sentences belonging to a defined grammar into LTL formulas in the subset we are considering. This grammar allows the user to define behaviors for the robot and to specify any assumptions regarding the behavior of the environment (e.g. "a person will never be seen in Region 1"). Furthermore, this grammar naturally captures *reactive* tasks in which the robot's behavior depends on the sensor information it receives while it is executing (e.g. "if you see a red light, stop"). Any sentence that cannot be translated into a corresponding LTL formula will cause an error.

In addition to the written specification, knowledge about the workspace topology (that is, information about which regions are adjacent, and consequently where the robot can move to in one "step") is automatically encoded into the logic formula based on the decomposition, thus constraining the possible motion of the robot based on its physical limitations in the environment.

LTLMoP implements the grammar described in [12], which includes conditionals, goals and safety sentences. In the near future we will enrich the grammar with non-projective locative prepositions such as "between" and "within" as described in [13].

[2]In our experience, this subset of LTL is still sufficient for describing a wide variety of tasks. For details, see the discussion in [7].

### B. Synthesis: Logic to automaton

Once a task is captured using an LTL formula, it is synthesized into an automaton based on the algorithm in [6]. This formal synthesis approach naturally generates correct-by-construction automata: if the task can be accomplished in *all* environments satisfying the given assumptions, then an automaton will be created such that every possible execution satisfies the required task. If, on the other hand, the task is unrealizable—that is, the environment can prevent the robot from achieving its task—then the algorithm will not create an automaton (worst-case approach).

It is important to note that, in this fragment, our specifications take the form of an implication:

$$[\textit{environment assumptions}] \Rightarrow [\textit{robot behavior specification}]$$

This means that the robot is guaranteed to behave according to specification under the synthesized controller only as long as the assumptions made about the environment hold true.

As for the synthesis process itself, the satisfying automaton is generated by treating the problem as a two-player game between system (robot) and environment: for every possible move the environment might take, a countering move for the robot is selected such that it continues to satisfy its task specification. For details, see [6].

### C. Hybrid Control: Automaton to control commands

The previous sections described how a continuous task is abstracted into a discrete representation (LTL) and then solved (synthesized into an automaton). What remains is to discuss the creation of a continuous controller from the discrete solution. To do this, the discrete automaton is executed as a hybrid controller: a transition between states (which depends on the sensor information the robot perceives during execution) corresponds to the activation of one or more atomic continuous controllers. For example, if the proposition $region_1$ is true in one state and in the next state it is false but the proposition $region_2$ is true, in order to satisfy the transition the robot must call a low-level motion controller that drives it from Region 1 to Region 2 without going through any other region.

Clearly, there is a tight connection between the environmental decomposition and the set of atomic controllers that can guarantee certain motion for different kinematics and dynamics. For a wheeled robot moving around a polygonal environment, several methods exist to create such controllers for any convex partition [14]–[18].

For a complete discussion of the hybrid controller, the reader is referred to [7]. Section III-B discusses how atomic controllers are incorporated within LTLMoP.

## III. Code Structure

The LTLMoP toolkit consists of several interconnected applications and modules, the relationship between which is illustrated in Figure 1. The Specification Editor serves as the main development environment for working on projects, and

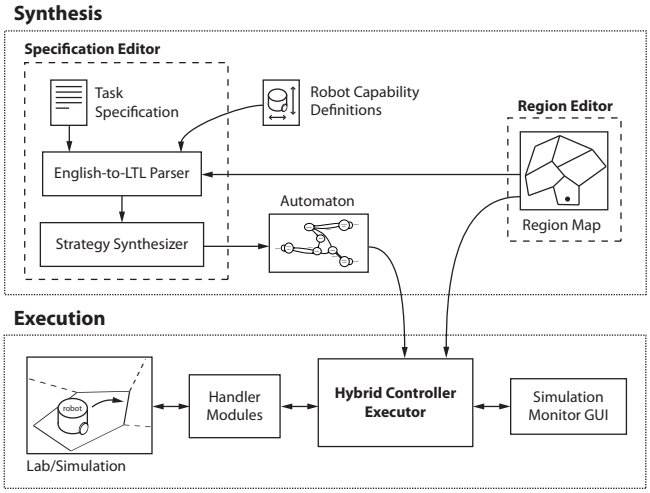


Fig. 1: Overview of LTLMoP components

all other applications and modules are called from within the Specification Editor.[3]

Project data is shared between applications in the form of data and configuration files. All data is stored in a simple plain-text format, so it can easily be modified by hand or exported from another program. The specific file formats are not discussed here, but more information can be found in the API documentation on LTLMoP's website [19].

A brief overview of each component is given in the following subsections, listed in the order in which one would typically encounter them when working on a project.

### A. Application Components

*1) Specification Editor:* The Specification Editor (see screenshot in Figure 2) is a development environment used for editing, compiling, and executing structured English specifications.

In addition to writing the specification in a standard text editor, region propositions (imported from region files created by the Region Editor) can be chosen visually from a map, and sensor and actuator propositions (imported from Robot Description Files) can be selected from a sidebar for convenience. For more complex behavior, custom propositions can also be declared.

Compilation is accomplished by calling the parsing and synthesis modules described below. The outputs of both stages of compilation can be viewed as well, either as the raw LTL specification or as an image of the resulting automaton (using the GraphViz package [20])

To run the compiled controller, an experimental setup must be defined by creating or selecting a configuration from the simulation configuration dialog (see screenshot in Figure 3). This configuration, which specifies the relationship between the abstract map and propositions referred to by the specification and their concrete counterparts in the experiment environment, is then passed on to the controller executor (described below). By having multiple experiment configurations associated with a single specification, the user can easily switch between simulated and real-world experiments, or even between different robots that provide the same sensors and actuators.

*2) Region Editor:* The Region Editor (Figure 4) is a simple graphical editor for creating named polygonal regions on a map (with an optional background image) and for automatically generating a topological connectivity graph based on region adjacency. In order to later find a coordinate frame mapping with the Calibration Tool (described below), the Region Editor provides a way to indicate which map vertices to use for the calibration. Also, because the motion controller included with LTLMoP (see Section III-B5) requires regions to be convex, Region Editor will quietly highlight any concave regions as they are drawn.

*3) Structured English Parser:* This module converts specifications in structured English into an equivalent LTL specification, as described in Section II-A. In addition to performing template-based mapping between syntaxes, the parser automatically replaces region name propositions with a minimal vector of propositions representing the region index, thus reducing the total number of propositions in the LTL formula. The parser module also uses topological connectivity information from the Region Editor to encode movement constraints on the robot.

*4) Controller Synthesizer:* A synthesis algorithm, described briefly in Section II-B and implemented on top of the JTLV framework [21], is used to check for realizability and generate a controller automaton from the LTL specification.

*5) Calibration Tool:* This script helps the user to experimentally determine the coordinate transformation between points on the region map and points in the coordinate system being used in an experiment, by taking measurements at calibration vertices that were pre-defined on the map using the Region Editor.

*6) Controller Executor:* This module executes the hybrid controller for a robot in a simulated or real environment, using the handlers listed in Section III-B.

*7) Simulation Monitor:* This application (see screenshot in Figure 5) provides real-time information about the status of the controller's execution in an experiment. The current position of the robot is plotted on the map, the controller update frequency is shown in the status bar, and debugging messages are displayed in the log window. Buttons are provided to pause and resume controller execution, as well as to export the log messages for later reference.

### B. Handlers

Handler modules form the glue that connects the discrete control automaton with the continuous world, be it a computer simulation or a physical experiment. The different types of handlers and their functions are described in the following sections, and a corresponding block diagram is given in Figure 6. Note that, within each handler category, different modules can be plugged in as necessary, allowing the same control automaton to easily interface with a wide

[3]However, the applications can also run independently; in particular, the controller executor can run without any GUI, for headless applications.

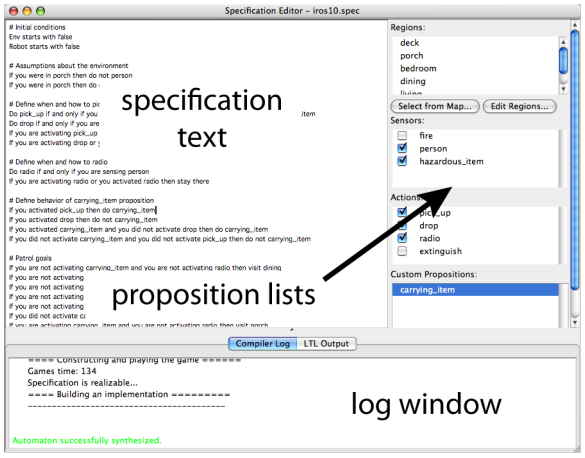


Fig. 2: Specification Editor

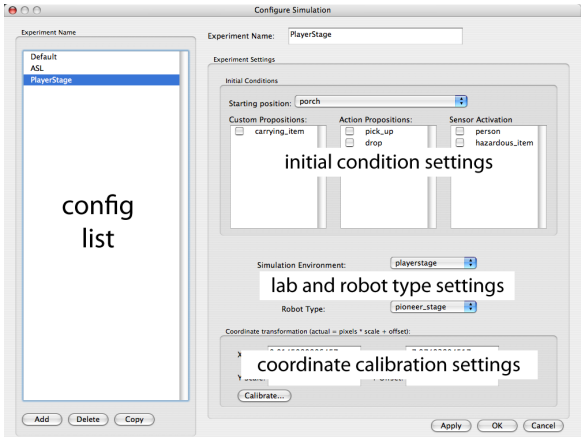


Fig. 3: Simulation configuration window

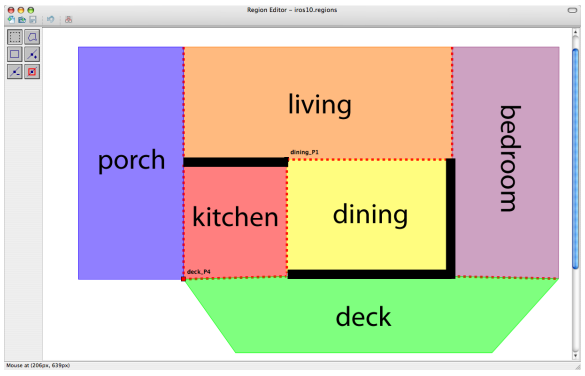


Fig. 4: Region Editor, showing map. (Image modified for legibility; thick black lines are walls.)

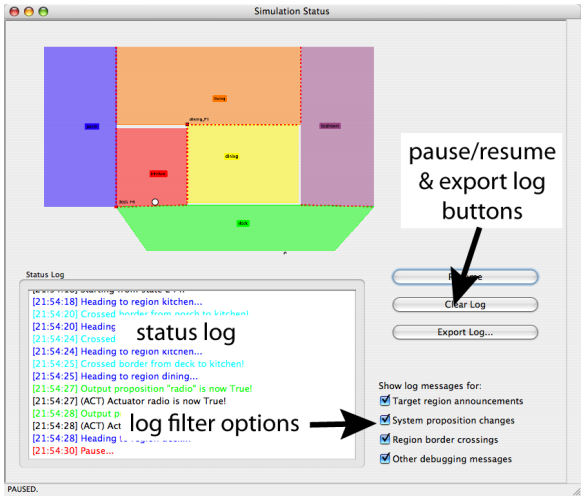


Fig. 5: Simulation Monitor

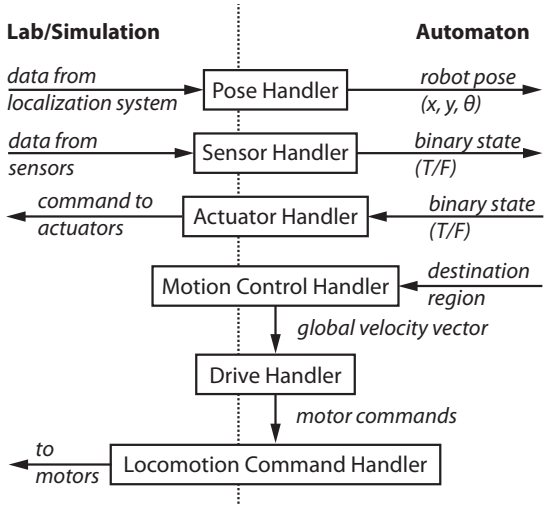


Fig. 6: Block diagram of handler modules

variety of robots and experimental setups. The handlers are determined per-experiment, based on choice of lab and robot configuration files.

*1) Initialization:* This handler does any initialization that needs to occur before other handlers are loaded. This includes tasks such as launching external simulation environments or opening any network connections that will need to be shared between multiple handlers. LTLMoP includes a handler to start a Stage simulation and a handler to connect to a Player server [22], [23].

*2) Pose:* This handler finds out the robot's current pose from a localization system. LTLMoP includes pose handlers for Player/Stage and the Orca robotics framework [24].

*3) Sensor:* Given the name of a sensor proposition, this handler returns a boolean value corresponding to the abstracted state of the corresponding physical sensor. LTLMoP includes a handler for receiving UDP data from Orca, as well as a simple virtual sensor simulator, which provides a GUI

with buttons so the user can adjust sensor values in real-time.

*4) Actuator:* Given the name of an actuator proposition and a state, this handler sets the corresponding physical actuator to be in that state. LTLMoP includes a handler for sending UDP packets to Orca, as well as a pass-through handler which simply prints out changes in actuator state.

*5) Motion Control:* This handler calculates an instantaneous linear velocity vector in the global reference frame that will help the robot get from the current region to a specified next region. LTLMoP includes a potential field-based motion control handler based on the work of Conner et al. [14].

*6) Drive:* Given a desired linear velocity vector $[V_x, V_y]$ (specified in the global reference frame) from the motion controller and knowledge of the current orientation $\theta$ of the robot, this handler constructs a drive command for the robot's motors that will cause it to follow this vector as closely as possible. This is to account for the fact that real robots are generally not holonomic or kinematic. LTLMoP includes a drive handler for differential-drive robots and a pass-through handler for simulated holonomic point robots.

*7) Locomotion Command:* This handler sends a locomotion command from the drive handler to the robot. LTLMoP includes handlers for Player/Stage and Orca.

## IV. Example

A specification for a simple "fire-fighting" scenario was written and tested in both simulation (using Player/Stage [22]) and in a lab environment.

In this example scenario, the robot's task is to enter a house (see map in Figure 4) from its porch and patrol its rooms. If it encounters a person in the house, the robot is to stay in the room with the person and radio for help. On the other hand, if it encounters a potentially hazardous item, the robot is to pick it up and take it to the front porch. Furthermore, the robot is to satisfy these goals infinitely often, so it will continue its patrol until its run is terminated. (For an excerpt of the specification, see Listing 1.)

In simulation, the virtual sensor handler was used (so the user could click on toggle buttons to dynamically adjust the value of the sensor propositions), as well as the virtual actuator handler (which simply prints out the status of an actuator when its state changes). A holonomic point-robot model was used for simplicity, with the heat-based potential-field motion controller [14] for moving between regions. The simulation monitor window for this experiment can be seen in Figure 5.

In the lab, a Pioneer 3-DX robot was used. A Vicon motion capture system was used for localization, and the same potential-field motion controller was used in conjunction with a basic feedback linearization algorithm to generate differential drive commands. For sensor input, a camera and simple blob-detection algorithm were used to detect red ("*person*") and blue ("*hazardous_item*") golf balls. For actuator output, a small red flag attached to a servo gimbal was used as a multi-function actuator: "*radio*" was signaled by waving the flag back and forth in the horizontal plane, and "*pick_up*" was indicated by the flag being raised to vertical, where it remained until the "*drop*" command restored it back to level. All components were linked using the lab's existing infrastructure, which itself uses the Orca robotics framework [24]. An annotated video of this demonstration accompanies this paper, and several frames are shown in Figure 7.

**Listing 1** Abridged structured English specification for firefighter example

```
# Initial conditions
Env starts with false
Robot starts with false

# Assumptions about the environment
If you were in porch then do not person
If you were in porch then do not hazardous_item

# Define when and how to pick up and drop
Do pick_up if and only if you are sensing
    hazardous_item and you are not activating
    carrying_item
Do drop if and only if you are activating
    carrying_item and you were in porch
                ⋮
# Define when and how to radio
Do radio if and only if you are sensing person
If you are activating radio or you activated radio
     then stay there

# Patrol goals
If you are not activating carrying_item and you
    are not activating radio then visit dining
                ⋮
If you are activating carrying_item and you are
    not activating radio then visit porch
```

## V. Conclusions and Future Work

The most recent version of LTLMoP can be downloaded from our SVN repository; instructions for doing so can be found at `http://code.google.com/p/ltlmop/`. The distribution includes a number of example projects and a step-by-step tutorial. Detailed API documentation can also be found at the same website.

The more diverse the array of available sensors, actuators, and motion controllers, the more interesting the specifications that can be written. To this end, we plan to create handlers for using the ROS robotics framework [25] with LTLMoP, and to interface with a legged robot with a manipulator in the near future.

Looking further ahead, we plan to extend LTLMoP's grammar to support a more expressive language, experiment with ways of providing useful feedback to the user in the case of an unrealizable specification, add support for multiple-robot scenarios, and explore methods for generating optimal strategies over probabilistic systems.

*Acknowledgements*

We would like to thank George Pappas for his guidance with earlier work, David Conner for his potential field controller code, Yaniv Sa'ar for his JTLV library and GR(1) synthesis code, Daniel Lee for his blob detector code, and Benjamin Johnson for beta-testing.

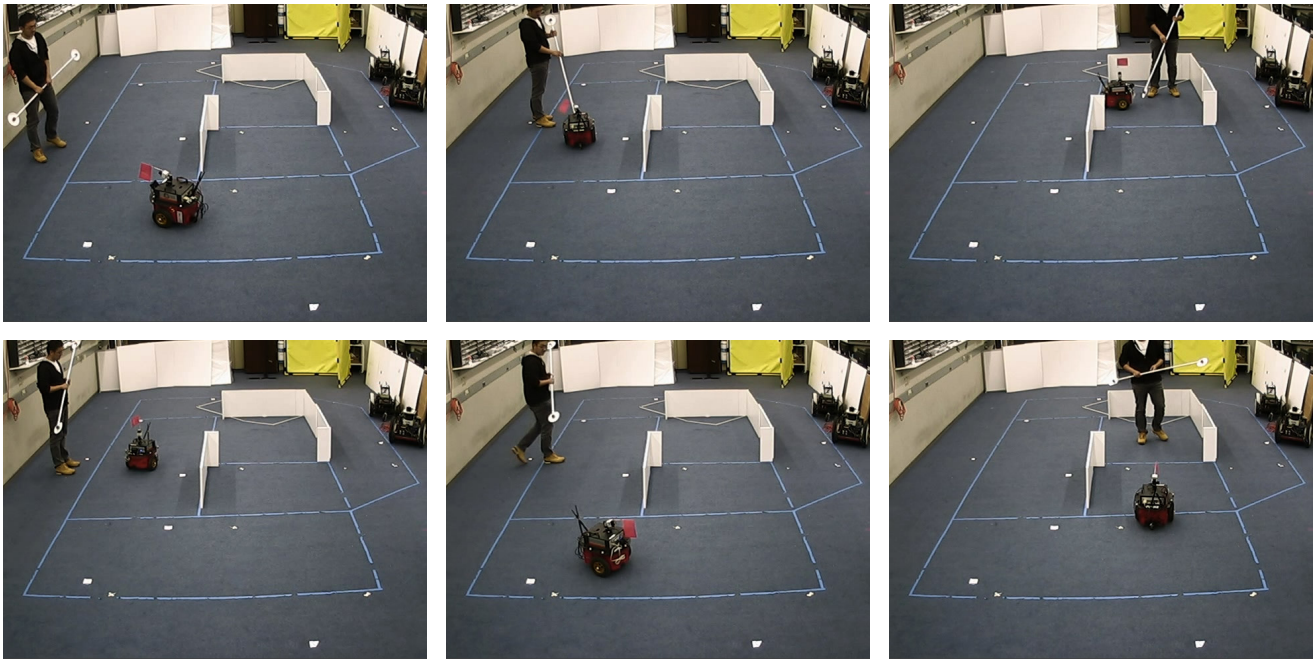

Fig. 7: Experimental run of the "fire-fighting" scenario. Left-to-right, top-to-bottom: A) The robot starts patrolling from the porch. B) It encounters a "person" in the living room and stops, waving its flag. C) Once the "person" is gone, the robot continues to the bedroom, where it now encounters and picks up a "hazardous item." D) The robot carries the item towards the porch. E) The robot drops the item on the porch. F) The robot heads to the kitchen to continue its patrol of the house.

## REFERENCES


[1] S. Russell and P. Norvig. *Artificial Intelligence, A Modern Approach*. Prentice Hall, second edition, 2003.

[2] W. Takano and Y. Nakamura. Integrating whole body motion primitives and natural language for humanoid robots. In *IEEE-RAS International Conference on Humanoid Robots*, 2008.

[3] Jason Wolfe, Bhaskara Marthi, and Stuart Russell. Combined task and motion planning for mobile manipulation. In *International Conference on Automated Planning and Scheduling*, Toronto, Canada, 05/2010 2010.

[4] E. Allen Emerson. Temporal and modal logic. In *Handbook of theoretical computer science (vol. B): formal models and semantics*, pages 995–1072. MIT Press, Cambridge, MA, USA, 1990.

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

[6] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of Reactive(1) Designs. In *VMCAI*, pages 364–380, Charleston, SC, January 2006.

[7] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Temporal logic based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

[8] Georgios E. Fainekos, Antoine Girard, Hadas Kress-Gazit, and George J. Pappas. Temporal logic motion planning for dynamic robots. *Automatica*, 45(2):343 – 352, 2009.

[9] M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from temporal logic specifications. *Automatic Control, IEEE Transactions on*, 53(1):287–297, February 2008.

[10] Michael Melholt Quottrup, Thomas Bak, and Roozbeh Izadi-Zamanabadi. Multi-robot planning: a timed automata approach. In *IEEE International Conference on Robotics and Automation*, pages 4417–4422, New Orleans, LA, 2004. IEEE.

[11] S. Karaman and E. Frazzoli. Complex mission optimization for multiple-uavs using linear temporal logic. In *American Control Conference*, Seattle, Washington, 2008.

[12] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Translating structured english to robot controllers. *Advanced Robotics Special Issue on Selected Papers from IROS 2007*, 22(12):1343–1359, 2008.

[13] Hadas Kress-Gazit and George J. Pappas. Automatic synthesis of robot controllers for tasks with locative prepositions. In *IEEE International Conference on Robotics and Automation*, 2010. To Appear.

[14] David C. Conner, Alfred A. Rizzi, and Howie Choset. Composition of Local Potential Functions for Global Robot Control and Navigation. In *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, pages 3546 – 3551, Las Vegas, NV, October 2003.

[15] Calin Belta and L.C.G.J.M. Habets. Constructing decidable hybrid systems with velocity bounds. In *IEEE Conference on Decision and Control*, Bahamas, 2004.

[16] D. Conner, H. Choset, and A. Rizzi. Integrated planning and control for convex-bodied nonholonomic systems using local feedback control policies. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2006.

[17] Stephen. R. Lindemann, Islam I. Hussein, and Steven M. LaValle. Realtime feedback control for nonholonomic mobile robots with obstacles. In *IEEE Conference on Decision and Control*, San Diego, CA, 2006.

[18] S. R. Lindemann and S. M. LaValle. Smooth feedback for car-like vehicles in polygonal environments. In *IEEE Conference on Robotics and Automation*, 2007.

[19] LTLMoP Project Website and API Documentation, http://code.google.com/p/ltlmop/.

[20] GraphViz Project Website, http://www.graphviz.org/.

[21] JTLV Project Website, http://jtlv.sourceforge.net/.

[22] Brian Gerkey, Richard T. Vaughan, and Andrew Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *11th International Conference on Advanced Robotics ICAR*, pages 317–323, Coimbra, Portugal, June 2003.

[23] The Player Project, http://playerstage.sourceforge.net/.

[24] The Orca Robotics Project, http://orca-robotics.sourceforge.net/.

[25] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully B. Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *International Conference on Robotics and Automation*, Open-Source Software workshop, 2009.