COMMUNICATIONS IN INFORMATION AND SYSTEMS
Vol. 8, No. 2, pp. 171-184, 2008

006

MOTION DESCRIPTION LANGUAGE-BASED TOPOLOGICAL MAPS FOR ROBOT NAVIGATION*

P. MARTIN† AND M. EGERSTEDT†

Abstract. Robot navigation over large areas inevitably has to rely on maps of the environment. The standard manner in which such maps are defined is through geometry, e.g. through traversability grid maps or through a division of the environment into free-space and obstacle-space. In this paper, we combine certain aspects of the geometric maps, through the notion of distinctive places, with a topological description of how these places are related. What is novel is the idea that the adjacency relation is defined by the existence of a control law that drives the robot between topologically connected places. Moreover, these maps can be automatically constructed based on the premise that the nodes correspond to places associated with a hightened control activity.

1. Introduction. The production of maps for navigation purposes only becomes meaningful if the maps are somehow coupled to their expected use. For example, a pedestrian map may contain many small roads, while an automotive map covering the same area only contains roads big enough for cars to drive on. Similarly, airplanes employ a completely different set of maps than land-based vehicles. What this somewhat informal discussion immediately tells us is that when maps are to be produced for robotics applications, they only become useful when they respect the constraints (may they be dynamic or geometric) imposed by the actual vehicle. This issue is highlighted clearly when constructing configuration spaces for mobile robots that take the spatial dimensions of the robots into account [12, 13].

In this paper we make this observation concrete by explictly taking the capabilities of the robot into account through a collection of predetermined control modes (or behaviors). This navigation system structure, i.e. decomposing a task into building blocks, has proven to be useful in that it allows the designer to produce controllers that are dedicated to performing specialized tasks, such as avoiding obstacles, clearing steep gradients, approaching landmarks, and so on. (See for example [2, 7].) Rather than constructing a single controller, the high-level mission is executed by a supervisor that determines what mode of operation to use in a particular situation. The selection of a set of modes has implications for what the robot can actually do, which we intend to make explicit in this paper.

As the mission scenarios for mobile robots become more elaborate, the trade-off between complexity and expressiveness of the map becomes an issue that must be taken into account [13, 18]. In other words, a highly detailed map may not be particularly useful in that planning over this map might take too long as compared to

*Dedicated to Roger Brockett on the occasion of his 70th birthday.

†School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA. E-mail: {pmartin,magnus}@ece.gatech.edu

the benefits associated with actually incorporating high-fidelity features into the map. As such, in this paper we will produce maps that contain only the information actually needed to navigate the environment, and the resulting maps will be topological in nature rather than geometric. This topological approach to planning can be traced back to the *semantic hierarchy* introduced in [11]. The semantic hierarchy generates so-called *distinctive places* by examining the local maximum of a "distinctiveness measure," based on sensor input.

Alternatively, our approach uses the robot's controller activity, itself, to generate a control-driven topological discretization of the environment. We utilize the framework of Motion Description Languages (MDL), as pioneered by Roger Brockett in 1988 [5] for generating distinctive places and their topological connections. The novelty of this approach lies in the practical application of Brockett's work to the field of mobile robot path planning.

The outline of this paper is as follows: In Section 2, hybrid control programs for robot control are discussed within the contexts of Hybrid Automata (e.g. [10, 15]) and MDL. In Section 3 we discuss how to automatically generate topological maps from executions of the hybrid control programs. The main idea is that individual executions (or runs) will correspond to strings of control laws as well as interrupts, i.e. conditions for the termination of the individual control laws. By replacing the interrupts with a description of the corresponding distinctive place that triggered the interrupt, strings of distinctive places and control laws are obtained. Finally, by identifying the same distinctive place in multiple locations in the string, a finite state machine is obtained, with places as the nodes, and control laws as edges. This finite state machine is in fact the desired topological map. Following this, in Section 4, some experimental results are given as this work is motivated by a scenario in which robots are to navigate unknown, unstructured, outdoor environments repeatedly, and learn the terrain as well as how the robot should interact with the environment, across multiple runs [19].

2. Controllers and Interrupts. This paper does not deal explicitly with the issue of low-level control design for mobile robots. Instead, we assume that a set of relevant control laws (or behaviors) have already been designed, dedicated to performing certain tasks such as avoiding obstacles or approaching landmarks. To make this concrete, assume that the robot dynamics can be described by

$$\dot{x} = f(x, u),\ y = h(x),$$

where $x(t) \in X \subseteq \mathbb{R}^n$ is the state of the robot at time t, $u(t) \in U \subseteq \mathbb{R}^m$ is the control signal, and $y(t) \in Y \subseteq \mathbb{R}^p$ is the output value. We moreover assume that the output equation encodes the interactions with the environment. Now, given a collection of control laws $\mathcal{K} = \{\kappa_1, \ldots, \kappa_N\}$, with $\kappa_i : Y \to U,\ i = 1, \ldots, N$, each control law gives

rise to the dynamics

$$\dot{x} = f(x, \kappa_i(y)),\ i = 1, \ldots, N.$$

Similarly, we also assume that we are given a set of so-called interrupt conditions (guard conditions), $\Xi = \{\xi_1, \ldots, \xi_M\}$, with $\xi_i : Y \to \{0, 1\},\ i = 1, \ldots, M$. The interpretation here is that an interrupt condition triggers when it changes values from 0 to 1. Using these two types of building blocks, we can produce a hybrid automaton without resets (e.g. [10]), where the dynamics in each discrete location of the hybrid automaton is defined by one of the control laws in $\mathcal{K}$ and each transition between discrete states is triggered by an interrupt condition in Ξ.

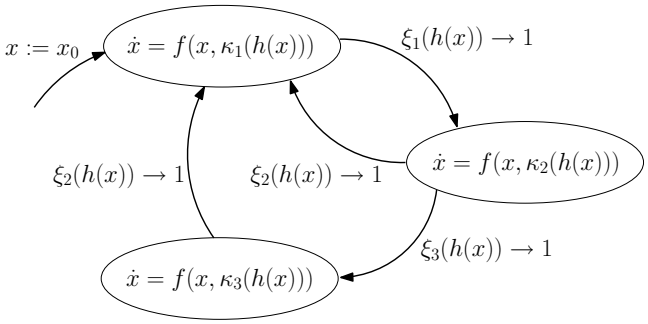

FIG. 1. *A hybrid automaton is shown with three modes, corresponding to each of the three controllers in* $\mathcal{K} = \{\kappa_1, \kappa_2, \kappa_3\}$*, while the transition conditions, denoted* $\xi_i(h(x)) \to 1$*, are driven by the interrupts in* $\Xi = \{\xi_1, \xi_2, \xi_3\}$.

An example of this is shown in Figure 1. The interpretation here is that the system starts at $x(0) = x_0$ and then evolves according to $\dot{x} = f(x, \kappa_1(y))$ until $\xi_1(y)$ triggers, i.e. it changes value from 0 to 1, and the system switches to use control law κ_2. The system thus evolves as $\dot{x} = f(x, \kappa_2(y))$ until either $\xi_2 = 1$ or $\xi_3 = 1$, if $\xi_2 = 1$ the system returns to using control law κ_1, while $\xi_3 = 1$ results in control law κ_3 and so on.

3. Producing Topological Maps. One reason why robot navigation is a challenging problem is its inherent complexity. This complexity stems from at least three different sources, namely the complexity of the robot dynamics, the complexity of the environment in which the robot is deployed, and the complexity of the task itself. As pointed out in [3], in order to manage these complexities, various forms of discretizations are common. These discretizations can either be spatial, e.g. through cell-decompositions of the free space or occupancy grids, or control-driven, as proposed in [3, 1, 4]. Control driven discretizations can arise when, as in the previous section, a finite number of control actions are available to the high-level supervisor. In this paper, we take the point of view that an interesting and potentially useful dis-

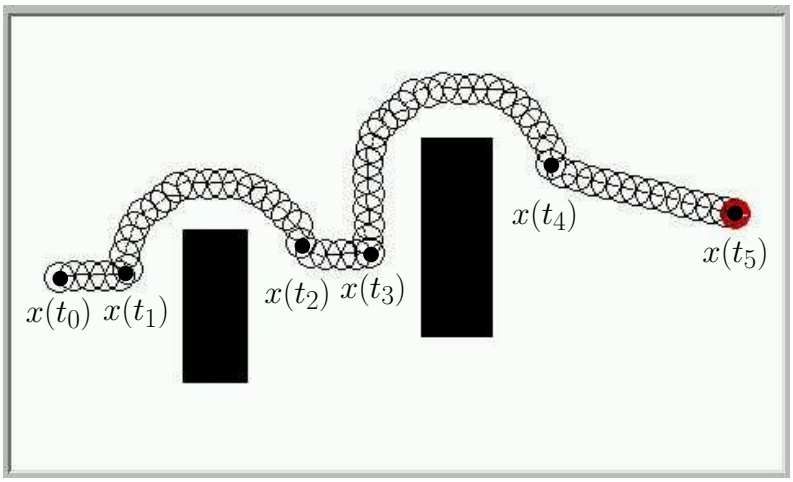

FIG. 2. *A robot executes a hybrid control program that switches between a go-to-goal behavior and an avoid-obstacle behavior.*

cretization lies somewhere between these two views of geometric and control-driven discretizations.

Given that a robot is executing a control strategy corresponding to a hybrid control program specified through the hybrid automaton A, with initial condition $x(t_0) = x_0$, this robot will interact with the environment $\mathcal{E}$ through an execution of A. In fact, the interaction with $\mathcal{E}$ gives rise to particular sequences of control-interrupt pairs, e.g. $(\kappa_1, \xi_1), \ldots, (\kappa_K, \xi_K)$. We will call this string a *run* through the environment, and we denote this by $\mathcal{R}(A, \mathcal{E}, x_0)$.

As an example, consider a robot with two behaviors, κ_1 and κ_2, corresponding to a go-to-goal behavior and a clockwise avoid-obstacle behavior, as shown in Figure 2. The interrupts used in that simulation were ξ_1, ξ_2, and ξ_3, where $\xi_1 \to 1$ when the robot is too close to an obstacle, $\xi_2 \to 1$ when the robot is clear of an obstacle, and $\xi_3 \to 1$ when the robot has arrived at a predetermined goal position. The corresponding hybrid automaton A is shown in Figure 3, where κ_ϵ is the "empty" control law corresponding to doing nothing.

Now, one can note, that the run that was being executed in Figure 2 is in fact the string

$$\mathcal{R}(A, \mathcal{E}, x_0) = (\kappa_1, \xi_1), (\kappa_2, \xi_2), (\kappa_1, \xi_1), (\kappa_2, \xi_2), (\kappa_1, \xi_3).$$

Such strings, obtained through the interaction of a hybrid automaton with the environment, make up words in Motion Description Languages [5, 8, 16]. The main idea behind the work in this paper is that such strings provide the means for coupling the controllers with the environment in a topological manner, as will be shown in the following paragraphs.

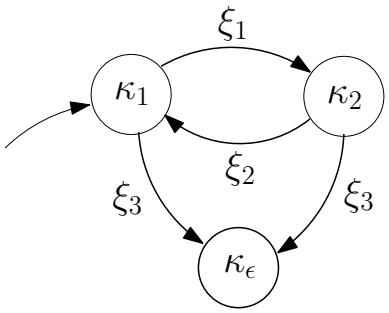

FIG. 3. *A hybrid automaton that generates the motion shown in Figure 2.*

3.1. Distinctive Places. As mentioned before, we utilize the idea of distinctive places proposed in [11]; however, what is novel with our approach, compared to [11] and [1], is that our distinctive places are produced by changes in control modes rather than the positive output of a "distinctiveness" measure. In general, as the robot is deployed in the environment during a run, the hybrid control program starts with a particular control law $\kappa_i \in \mathcal{K}$. This control law will dictate the evolution of the system as $\dot{x} = f(x, \kappa_i(h(x)))$ until an interrupt triggers. Associated with the triggering of this interrupt are the particular environmental conditions behind the triggering, and even though it may trigger explicitly because the robot was within a certain distance of an obstacle, this interrupt does not contain any additional information about which obstacle it was, or how big it was, or where it was located.

In contrast to this, we will associate another interrupt condition $\zeta_i : Y \to \{0, 1\}$ with this situation, defined in such a way that it triggers based on additional information in the environment. This information will be selected in such a way that it that uniquely identifies the distinctive place in the environment where the interrupt triggered. For the experimental purpose of this paper, this information will simply be given by the GPS coordinate of the location where the interrupt triggered. However, one can easily imagine a situation where much more rich environmental descriptions are used, such as visually based submaps, as proposed in [1, 9], or semantic SLAM [17].

Since our work uses GPS as its primary localization sensor, we focus on *spatial* data for determining the location of the robot and distinctive places. Hence, we associate a location with each distinctive place, and let $pos(y) \in \mathbb{R}^p$ (where $p = 2$ in the case of a planar world (as in Figure 2) denote the spatial position of the robot when it is in state $x \in X$ with output $y \in Y$, then the new interrupt condition would be

$$\zeta_i(y) = 1 \;\Leftrightarrow\; pos(y) \in \mathcal{B}_\delta(pos(y(t_i))).$$

Here $\mathcal{B}_\delta(z) \subset \mathbb{R}^p = \{z' \in \mathbb{R}^p \mid \|z' - z\| \leq \delta\}$ is the closed ball around z with

radius $\delta > 0$. Moreover, t_i is the time at which interrupt ξ_i triggered during the run. Returning to the example in Figure 2, we have shown the state associated with the robot at the interrupt times $t_1, \ldots, t_5$ as circles, together with the initial state $x_0 = x(t_0)$ at the initial time t_0.

Throughout the remainder of this paper, we will assume that no two distinctive places are allowed to coincide. And, given a distinctive place ζ_i, if another interrupt ξ_j triggers at time t_j during the run in such a way that $\zeta_i(y(t_j)) = 1$ for some $i \neq j$, then the triggering of these two interrupts are assumed to correspond to the *same* distinctive place, which is a mechanism that will be used to produce topological maps with cycles.

3.2. Graph-Based Models. Based on the terminology established so far, given a run

$$\mathcal{R}(A, \mathcal{E}, x_0) = (\kappa_1, \xi_1), \ldots, (\kappa_p, \xi_p)$$

and the corresponding distinctive places $\zeta_i,\ i = 1, \cdots, p$ associated with the locations where the interrupts triggered, the resulting topological map is given by a directed, weighted graph $G = V \times E \times L \times W$. If we let the set of distinctive places be given by Z, we have that $V = Z$ is the set of vertices associated with each distinctive place, and $E \subset V \times V$ is the set of edges associated with the control law that takes the robot between adjacent distinctive places. Furthermore, $L : E \to \mathcal{K}$ associates a label with each edge. This label is defined through the control law used to take the robot between adjacent distinctive places. The final object, W, is the edge-based weight function $W : E \to \mathbb{R}^+$ that associates a weight to each edge. This function could for instance be given by the distance travelled along the edge, or the time that it took for the robot to traverse the edge.

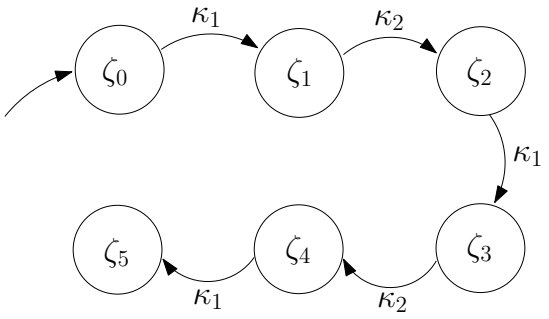

FIG. 4. *A graphical depiction of a run corresponding to the control execution in Figure 2.*

Returning to the example in Figure 2, with the understanding that ζ_i denotes the distinctive place corresponding to the closed, planar ball around the location $x(t_i)$, we get the corresponding graphical model of the run as shown in Figure 4. It should be pointed out that the graph in Figure 4 is a line graph, but with runs that return to

the same distinctive place multiple times, loops will be formed, as shown in the next section. Through this construction, a topological map of the environment is produced that combines the geometric information encoded in the distinctive places, with the control laws needed to take the system between such places. Moreover, multiple runs with varying initial conditions as well as varying hybrid control programs can easily be combined in a straightforward manner. Note that the generation of distinctive places necessarily ties the generated map to the environment. However, this result is natural since, as we described previously, each run through the environment is unique to that environment's features.

3.3. Planning over Graphs. Consider a mobile robot that repeatedly enters an environment from the same initial location, using different control programs at each run. Assume now that these runs generate the following strings of distinctive places/control modes corresponding to the different hybrid control programs:

$$\zeta_0, \kappa_1, \zeta_1, \kappa_2, \zeta_2, \kappa_3, \zeta_1, \kappa_4, \zeta_3$$
$$\zeta_0, \kappa_2, \zeta_2, \kappa_1, \zeta_4, \kappa_2, \zeta_5$$
$$\zeta_0, \kappa_3, \zeta_3, \kappa_1, \zeta_0.$$

The corresponding topological map is shown in Figure 5. (Note that in this case, the robot was initialized at the same distinctive place, ζ_0; however, in general the robot is not required to have the same initial position.)

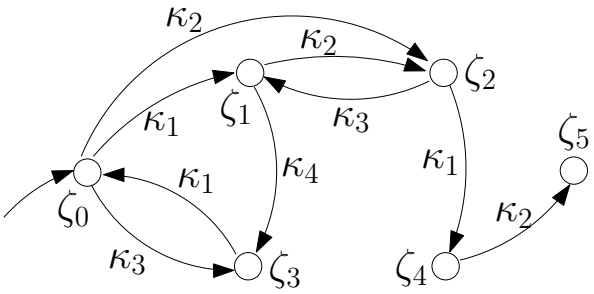

FIG. 5. *A topological map generated by repeated executions of a hybrid control program.*

Now, assume that the task under consideration in the fourth run is to get as quickly as possible to the distinctive place ζ_5 from the initial distinctive place ζ_0. Since our approach uses a graph based structure, this path planning problem can be solved using standard search algorithms such as Djikstra's algorithm or $A^\star$ [6, 13]. The primary difference in interpretation of the search algorithm output is that, in our case, the output is actually a string of desired motion programs. If we assume in this example that all edges take the same amount of time to traverse, the optimal control string is thus given by the MDL string $(\kappa_2, \zeta_2), (\kappa_1, \zeta_4), (\kappa_2, \zeta_5)$, which takes the robot to ζ_5 in three steps.

FIG. 6. *The LAGR robot in a test environment.*

4. Experimental Results. The automatic generation of topological maps was implemented and tested on the robot platform (the LAGR Robot) shown in Figure 6, equipped with four color cameras, a front bump switch, a Garmin GPS receiver, and an internal inertial measurement unit. The cameras are paired together so that each pair can provide stereo depth maps with a range of approximately 6 meters. The robot's turning axis is centered on the front axle, with the back two unpowered wheels turning on casters.

The implementation is based on a message framework that allows multiple processes to run concurrently and pass data without worrying about data corruption or racing conditions. In our architecture we have developed two processes for the management and generation of MDL graphs. The *Behavior Monitor* process inspects the incoming sensor data and determines whether a new behavior should be chosen in order to overcome new obstacles, pursue goals or complete other tasks. The second process necessary for constructing these graphs is the *MDL Graph Monitor*. This process listens for behavior state information from the Behavior Monitor and modifies the graph accordingly.

Figure 7 shows the interaction of our two implemented processes. While a robot is performing a run in the test environment, sensor data is continually sent to the Behavior Monitor. The process will send this data via an *update* message to the MDL Graph Monitor. The MDL Graph Monitor will check to see if the current graph has a node near the incoming GPS coordinates. If that is the case then a *looped node* message is sent back to the Behavior Monitor so that it can change its behavior to take advantage of the information.

Once the robot wakes up, the planning phase is initiated and the MDL Graph

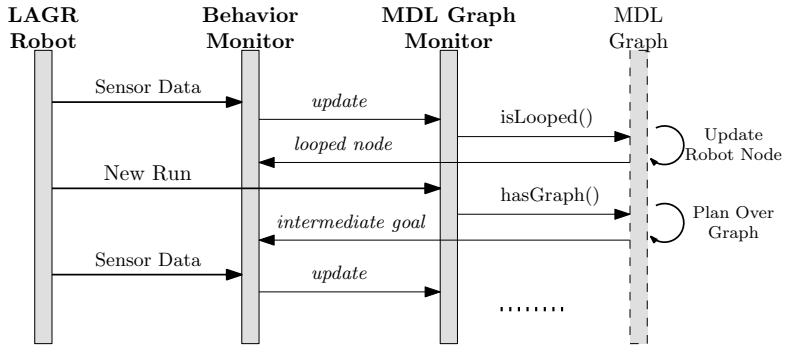

FIG. 7. *The message sequence diagram for the MDL Graph Building system is shown. The LAGR Robot, Behavior Monitor and MDL Graph Monitor are processes; The MDL Graph is an object owned by the MDL Graph Monitor.*

Monitor checks if a previous run in the environment was recorded with an MDL graph. If a graph is found a shortest path algorithm generates a list of goals and behaviors which are sent to the Behavior Monitor with an *intermediate goal* message. From this point the system returns to its initial operation, refining the MDL Graph while following intermediate goals.

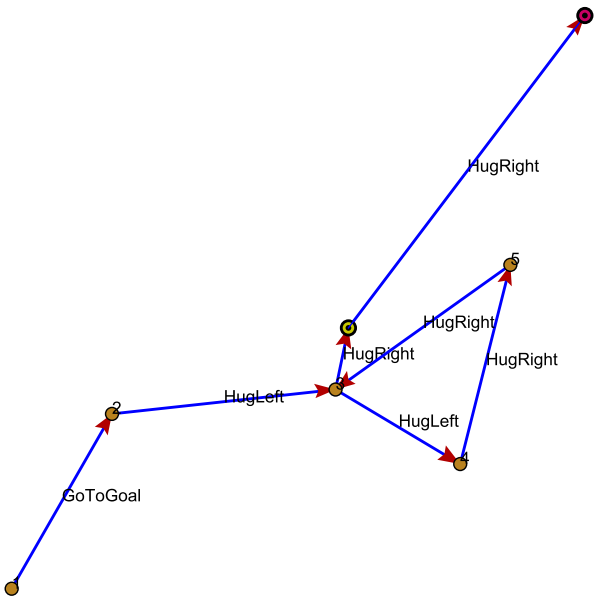

FIG. 8. *Depicted is the outcome of a run in a simple simulated environment that causes the robot to generate a* looped *MDL graph. The goal is seen in the top right corner. The robot is represented by the last node in the graph connected to the goal*

Figure 8 shows a simulated run using the so-called `HugBug` algorithm, which is a variant of Lumelsky's bug algorithm [14], where the robot (or "bug") moves in

a straight line toward a goal, until it encounters an obstacle. When an obstacle is detected, the robot avoids it, hugging the perimeter of the obstacle, until it can return to the previous straight line toward the goal. Our approach is similar, with the modification that we allow the robot to immediately head toward the goal again, once it is clear of the obstacle. Furthermore, we introduce a timeout factor to prevent exploration too far in any given direction, as described in [13].

In Figure 8, the robot starts in the bottom left and runs the `GoToGoal` behavior until an obstacle is detected, causing the `HugLeft` behavior to be selected. At a junction shortly after the first switch, the robot chooses to follow another obstacle southeast until encountering another obstacle. The robot maneuvers around this obstacle and finally reaches a previously visited node. At this point, the MDL Graph Monitor informs the Behavior Monitor that it has visited this node before. The robot then uses `HugRight` to move through the junction opening that it missed before and resumes navigating towards the goal. Note, that the robot associates its *forward* edge to the goal with its *current* control mode. After this run, the planner applies Djikstra's algorithm to generate the string of places and behaviors that would avoid the loop. When a new run is started, the robot reaches the goal more quickly since the cycle in the MDL graph was cut out by choosing the shortest path.

We have moreover performed experiments using this approach in test environments that contain a navigable terrain, with walls of obstacles, built using hay bales. We set up one particular environment such that an approximately 30 meter wall blocked the direct path to the robot's goal. The ends of the wall had perpendicular hay bales, making a U-shape. This U-shaped obstacle was chosen to test the robot's ability to navigate out of a "*cul-de-sac*". At the start of each run, the robot was placed about 75 meters from the goal, with the wall of obstacles in between.

As the robot negotiates the test environment, we are able to monitor its progress on the dashboard, shown in Figure 9. The dashboard shows a view of the cost map (top four images), vision sensor input (side images), and the output MDL graph (center image). The cost map is simply a grid based representation of the environment with a score given to each cell in the grid, representing the cost for the robot to traverse that cell. Higher costs are represented with darker colors, with lower cost (better terrain) represented with lighter colors.

As shown in the experiment performed in Figures 9 and 10, the robot starts in the `GoToGoal` behavior and heads toward the goal until it reaches the wall of obstacles which force the robot to turn to the left. At this point, the robot enters the `HugRight` behavior and a node is created in the MDL graph. The robot continues to navigate along the wall, until it reaches the edge of hay bales extending out from the wall. At this point of the obstacle wall, the robot experienced "phantom" obstacles generated by faulty vision input. These phantom obstacles caused the LAGR robot to alternate between several `HugRight` and `HugLeft` behaviors. We simplify the cycle

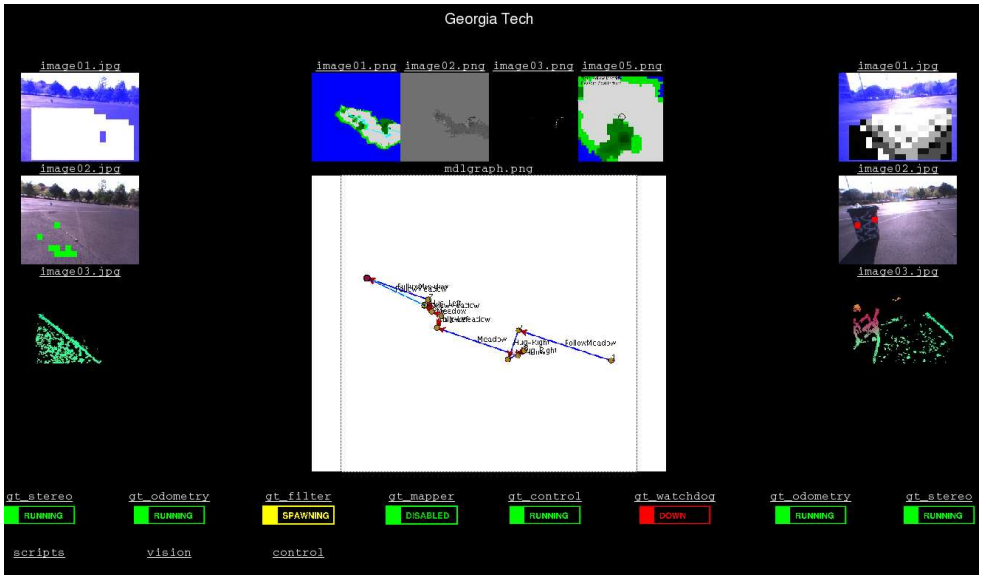

FIG. 9. *The robot's MDL graph associated with a particular experiment is shown.*

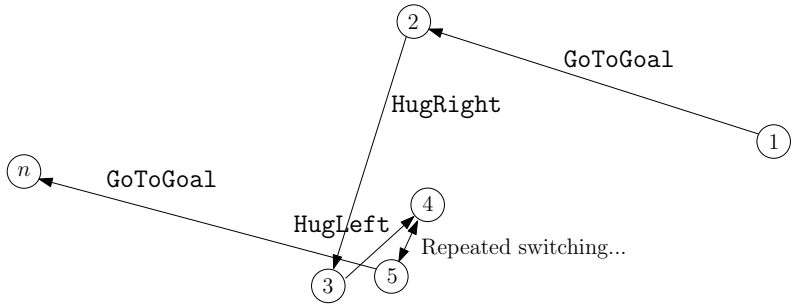

FIG. 10. *This illustration shows a closer view of the initial cycle in Figure 9, where the LAGR robot executed numerous behavior switches in order to overcome an obstacle.*

in Figure 10 by showing a bi-directional edge between nodes 4 and 5. Eventually, the robot managed to get sensor information showing that a `GoToGoal` behavior should be executed.

With each experiment in this environment, the MDL graph is created due to the changes in the robot's behavior state. At each behavior state change, we are able to see the change on the dashboard and monitor the addition of new nodes in the MDL graph, corresponding to the behavior events. Follow up experiments in this same environment use the previously generated MDL graphs for planning. If the robot is not close enough to any distinctive place, it starts a new graph from its starting position, possibly connecting to the current graph if its navigation through the environment leads to a previous distinctive node.

5. Conclusion. In this paper, we combine the capabilities of the robot, as defined by a collection of control laws, with geometric information of the environment in which the robot is deployed. The way these seemingly disparate entities are combined is in a topological map, where the nodes correspond to distinctive places found in the environment, i.e. to geometric objects. At the same time, edges between nodes correspond to control laws that take the robot between adjacent nodes. The result is thus a sparse representation of the environment that encodes how the robot should move around in the environment, in contrast to purely geometric descriptions of the environment, which are decoupled from the actual capabilities of the robot.

We moreover present an automatic way of obtaining the topological maps from example runs, when the robot is executing a given, hybrid control program. Experimental results in unstructured, outdoor environments testify to the viability of the proposed approach, and some future directions involving how to plan over the resulting graphs are outlined.

Acknowledgements. This work was supported by DARPA through Grant number FA8650-04-C-7131.

The authors are grateful to Charles Pippin, Tucker Balch, James Reigh, Frank Dellaert, Aaron Bobick, and David Wooden for helpful comments and discussions.

REFERENCES

[1] S. Andersson and D. Hristu-Varsakelis. *Symbolic Feedback Control for Navigation*. IEEE Transactions on Automatic Control, 51:6(2006), pp. 926-937.

[2] R.C. Arkin. *Behavior Based Robotics*. The MIT Press, Cambridge, MA, 1998.

[3] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G.J. Pappas. *Symbolic Planning and Control of Robot Motion: State of the Art and Grand Challenges*. IEEE Robotics and Automation Magazine, 14(2007), pp. 61-70.

[4] A. Bicchi, A. Marigo, and B. Piccoli. *Feedback Encoding for Efficient Symbolic Control of Dynamical Systems*. IEEE Transactions on Automatic Control, 51:6(2006), pp. 1-16.

[5] R. W. Brockett. *On the Computer Control of Movement*. Proceedings of IEEE Conf. Robotics Automation, New York, Apr. 1988, pp. 534540.

[6] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 2001.

[7] M. Egerstedt. *Behavior Based Robotics Using Hybrid Automata*. Lecture Notes in Computer Science: Hybrid Systems: Computation and Control, pp. 103-116, Pittsburgh, PA, Springer-Verlag, March 2000.

[8] M. Egerstedt and R.W. Brockett. *Feedback Can Reduce the Specification Complexity of Motor Programs*. IEEE Transactions on Automatic Control, 48:2(2003), pp. 213–223.

[9] J. Folkesson, P.Jensfelt, and H. I. Christensen. *Graphical SLAM using vision and the measurement subspace*. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, 2005.

[10] T. Henzinger. *The Theory of Hybrid Automata*. Proceedings of the 11th Annual Symposium on Logic in Computer Science (LICS), IEEE Computer Society Press, 1996.

[11] B. Kuipers and Y. Byun. *A Robot Exploration and Mapping Strategy Based on a Semantic Hierarchy of Spatial Representations*. Robotics and Autonomous Systems, 8(1991), pp. 47-63.

[12] J.C. Latombe. *Robot Motion Planning*. Kluwer, Boston, MA, 1991.

[13] S. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, UK, 2006.

[14] V. Lumelsky and A. Stepanov. *Path Planning Strategies for Point Mobile Automaton Moving Amidst Unknown Obstacles of Arbitrary Shape*. Algorithmica, 2(1987), pp. 403-430 .

[15] J. Lygeros, C. Tomlin, and S. Sastry. *Controllers for Reachability Specifications for Hybrid Systems*. Automatica, 35:3(1999).

[16] V. Manikonda, P. S. Krishnaprasad, and J. Hendler. *J.C. Willems, J. Baillieul, editor. Languages, behaviors, hybrid architectures and motion control*. In: Mathematical Control Theory, pages 199-226. Springer, 1998.

[17] A. Ranganathan and F. Dellaert. *Semantic Modeling of Places using Objects*, Robotics: Science and Systems, 2007.

[18] D. Wooden and M. Egerstedt. *Oriented Visibility Graphs: Low-Complexity Planning in Real-Time Environments*. IEEE Conference on Robotics and Automation, Orlando, FL, May 2006.

[19] http://www.darpa.mil/IPTO/programs/lagr/lagr.asp