EFFICIENT PLANNING OF AUTONOMOUS ROBOTS USING HIERARCHICAL DECOMPOSITION

Matthias Rungger, Olaf Stursberg
Institute of Automatic Control Engineering, Technische Universität München, 80333 Munich, Germany
{*matthias.rungger, stursberg*}*@tum.de*

Bernd Spanfelner, Christian Leuxner, Wassiou Sitou
Software and System Engineering, Technische Universität München, 80333 Munich, Germany
{*leuxner, sitou, spanfeln*}*@in.tum.de*

Keywords: Hierarchical Planning, Autonomous Robots, Context Adaptation, Hybrid Models, Predictive Control.

Abstract: This paper considers the behavior planning of robots deployed to act autonomously in highly dynamic environments. For such environments and complex tasks, model-based planning requires relatively complex world models to capture all relevant dependencies. The efficient generation of decisions, such that realtime requirements are met, has to be based on suitable means to handle complexity. This paper proposes a hierarchical architecture to vertically decompose the decision space. The layers of the architecture comprise methods for adaptation, action planning, and control, where each method operates on appropriately detailed models of the robot and its environment. The approach is illustrated for the example of robotic motion planning.

1 INTRODUCTION

The spectrum of applications for future robots will continuously grow, and an increasing percentage will be employed in dynamic environments. Examples are service robots which assist elderly or disabled persons and rescue robots which operate in hostile areas that are devastated by earthquakes or similar incidents. A characteristic of such applications is that an autonomous planning of actions must consider changing environment conditions and include reliable and immediate decisions on which available resources should be employed to fulfill a momentary task. In particular, if the environment includes dynamic objects with non-deterministic or partly unpredictable behavior, the representation of the constraints for a planning problem becomes complex and has a significant impact on the realtime-computability of action plans. Another difficulty encountered in this case is that environment models which are purely identified based on data series measured over a short period of time for a specific situation are merely suitable to reflect the behavior of the environment sufficiently well.

In order to cope with the issues of model complexity and quality of prediction within action planning, this paper proposes a planning architecture which combines multi-layer decision making with the use of a knowledge base for storing learned goal-attaining action strategies. Of course, the idea of hierarchical planning architectures is not new in general, and corresponding approaches can be found, e.g., in (Nau et al., 1998), (Galindo et al., 2007), (Barto and Mahadevan, 2003). However, the novel contribution of this paper is that a three-tier scheme is suggested which separates the planning task into steps of configuring the system structure, of planning a sequence of actions, and refining these actions into locally optimal control trajectories. The configuration step allocates the set of resources (sensors, actuators, or algorithms for planning and control) that seem most appropriate to accomplish a given task. By employing reinforcement learning, the action planning on a medium layer produces a sequence of discrete actions that qualitatively accomplish the task. The third layer refines an action plan by generating control trajectories which correspond to the action plan and establish a quantitative setting for the actuators over time. The overall concept setting is explained in the following section.

2 HIERARCHICAL PLANNING APPROACH

The proposed hierarchy vertically decomposes a complex planning task for reduction of complexity wherever possible, and consists of the layers shown in Fig.1: The adaptation-layer perceives and evaluates the current situation based on measured data and decides on the configuration, i.e. the resources selected to solve a current task. The planning layer considers the allocated resources and existing constraints for calculating an action plan that accomplishes the task. In the opposite direction, information about the existence of a feasible plan for the chosen configuration is transmitted from the planning to the adaptation layer. The control layer refines the action sequences received from the planning layer by computing a control trajectory for each action. The control trajectory is then passed to the corresponding actuators of the system. If no feasible trajectory is found for an action (or a part of the action sequence), this information is passed by to the planning layer and triggers replanning. Thus, the three layers continuously interact during the online execution to compute a task accomplishing strategy. To enable this computation, the system must have the capability to predict the behavior of itself and its environment up to a point of time in which the task is accomplished or cannot be accomplished anymore. These predictions are obtained from evaluating the models shown in Fig. 1 for a choice of configurations, action sequences, and control strategies.

The hierarchy implements a nested feedback loop also in the following sense: as soon as a current situation changes (e.g. because either a task changes or a relevant change of the environment is detected) the previously generated configuration, action plan, or control trajectory is re-evaluated and possibly modified. The reactivity to the behavior of the environment requires continuous update of the models based on measurement signals from the sensors of the autonomous system. On each layer, appropriate identification techniques are included to reparametrize the models to the momentarily perceived situation.

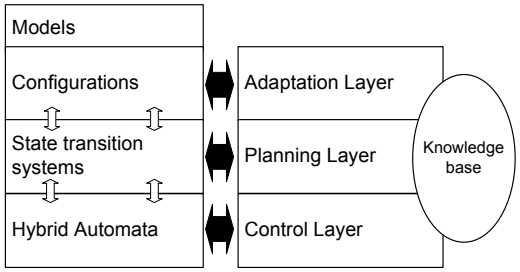

Figure 1: The hierarchical architecture.

In summary, the main objective of the three-tier architecture is (a) to decompose the decision making into three qualitatively distinct categories, (b) to select components, algorithms, and behavior in a top-down manner to reduce the search effort in the decision space, and to (c) include a knowledge-base by which already 'experienced' (or learned) behavior is used as heuristics for efficiently deciding which strategy is goal-attaining. For this scheme, the claim is not to outperform single state-of-the-art algorithms on a particular layer, but to provide an architecture for proper integration of different algorithms to solve a broad variety of complex planning tasks.

3 ADAPTATION LAYER

The term *adaptation* is here understood as the capability of an autonomous system to react to changing tasks or varying aspects of the context in which the system is embedded. The term 'context' refers to the state of the system environment, as e.g. the proximity of obstacles to a moving robot (the 'system'). The adaptation on the uppermost layer of the hierarchy means here to deduce from the current context and a task to be accomplished a suitable *configuration*, which is a subset of available components, i.e. available hardware devices (e.g. actuators) or software algorithms for action planning and control.

Components. On the adaptation layer, the system and its environment are described in terms of communicating units called *components*. Formally, a system is defined as a pair (C, CH) with C as a set of components and CH as a set of channels. The components communicate through directed channels, which are defined by $ch = C \times C \times Id$ with Id as a set of unique names. A system is completely determined by a network of components connected via channels, and the components send messages along channels and thereby express their behavior. Different ways to describe such behavior exist, like e.g. process algebras or relations on inputs and outputs like FOCUS (Broy and Stoelen, 2001) or COLA (Haberl et al., 2008). The *behavior* of a component is here specified by a sequence of data $msg \in MSG$ which is received and sent over the ports I and O of the component.

Adaptation Mechanism. To formalize the adaptation of a component-based model, the notion of *Mode Switch Diagrams* (MSD) is introduced. An MSD is

a tuple (M, map, δ, m_0) defining a transition system with a set of modes M, a transition function δ, a function $map : m \rightarrow C$ that relates modes to components, and m_0 is the initial mode. The transitions are defined by $\delta : M \times P \rightarrow M$ with predicates P depending on the inputs $i \in I$ of the components assigned to a mode. The function δ encodes the transition of the MSD between two modes m_i and m_{i+1}, and it represents the *adaptation*. An example of an adaptation is shown in Fig. 2. An activated mode m_{i+1} determines the components of the system which are selected until a new context change triggers another transition. The mode constitute the frame for the action planning carried out on the medium layer.

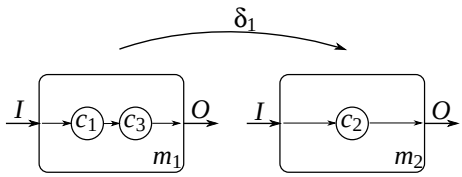

Figure 2: Transition between two modes m_1 and m_2 of an MSD example.

4 PLANNING LAYER

The planning layer comprises algorithms that search for a sequence of actions which transfers the initial state of the system or environment into a desired goal state. The specific algorithm to be used for a given task, the available set of actions, and the relevant model is determined by the mode information received from the adaptation layer. The model used on the planning layer is specified as a state transition system $\Sigma = (S, A, \delta)$ consisting of a set of states $S = \{s_1, s_2, \ldots, s_n\}$, a set of actions $A = \{a_1, a_2, \ldots, a_m\}$, and a transition function $\delta : S \times A \rightarrow 2^S$. The planning task is mapped into a goal state s_G (or a set of goal states $S_G \subseteq S$, respectively) which has to be reached from a current state $s_0 \in S$. A *plan* is a sequence of actions that evokes a state path ending in s_G (or an $s \in S_G$).

The planning techniques on the medium layer follow the principle of *reinforcement learning*. In order to formulate the planning problem as optimization, an utility function $r : S \times S \rightarrow \mathbb{R}$ is introduced, which defines the reward of taking a transition of Σ. The goal is encoded implicitly by assigning a high reward to a state sequence via which a goal state is reached, and by producing a negative reward for undesired behavior of Σ. Reinforcement learning represents learning from interaction, i.e. the system learns a strategy (as a state-to-action mapping) based on the reward. At time t, the system observes the state $s \in S$, and for a current policy $\pi = P(a|s)$ an action $a \in A$ is chosen. The system moves from state s to s' according to the transition function, and it receives the reward r. The goal is to learn a policy π which maximizes the cumulative discounted future reward. By introducing the action-value function $Q(s,a)$, a measure is available which estimates the profit of taking an action a in state s. The action-value function under policy π is given by the Bellman equation:

$$Q^{\pi}(s,a) = [R(s'|s,a) + \gamma \sum_{a'} \pi(s',a') Q^{\pi}(s',a')],$$

where s' and a' denote state and action at the next time instant. γ is the discount factor and is given by $0 < \gamma < 1$. $R(s'|s,a)$ denotes the reward for taking action a in state s, resulting in the new state s'. If the reward function as well as the system dynamics are known, the optimal policy

$$Q^* = \max_{\pi} Q^{\pi}(s,a)$$

can be calculated explicitly, resulting in a system of $|S| \cdot |A|$ equations. Since this is often computationally intractable (in addiion to some dynamics may not be known before-hand) approximations to the optimal action-value function are used.

One possible algorithm to estimate $Q^*(s,a)$ is called SARSA (Takadama and Fujita, 2004), which learns the current state-value function $Q(s,a)$ by updating the function with the observed reward, while interacting with the environment:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q^{\pi}(s',a') - Q(s,a)],$$

where α is the learning rate. SARSA is a so called *on-policy* RL algorithm, which updates the action-value function while following a policy π. The policy is ε-*greedy*, where the greedy action $a^* = \arg\max_a Q(s,a)$ is selected most of the time. Once in a while, with probability ε, an action is chosen randomly. SARSA is used here as RL technique in order to reduce the "risk" by avoiding negative rewards while paying with a loss of optimality (Takadama and Fujita, 2004).

5 CONTROL LAYER

The control layer ensures the correct execution of the action sequence derived on the planning layer. It establishes a connection of the discrete actions received from the medium layer to the real world by applying continuous signals to the system's actuators. The combination of discrete actions with continuous dynamics motivates the use of *hybrid automata* (Henzinger, 1996) to model the behavior on the control layer. Hybrid automata (HA) do not only establish

an adequate interface to the higher layers, but also provide the necessary expressivity required to model, e.g. robot-object-interaction. For each component selected by the adaptation layer, one hybrid automaton is introduced, and the various automata can interact via synchronization or shared variables.

Using a variant of HA with inputs according to (Stursberg, 2006), a hybrid automaton modeling the system is given by $HA = (X, U, Z, inv, \Theta, g, f)$, with X as the continuous state space, U the input space, Z the set of discrete locations, *inv* the assignment of invariance sets for the continuous variables of the discrete locations, and Θ the set of discrete transitions. A mapping $g : \Theta \rightarrow 2^z$ associates a guard set with each transition. The discrete-time continuous dynamics f defining the system dynamics is $x(t_{j+k}) = f(z(t_k), x(t_k), u(t_k))$. At any time t_k, the pair of continuous and discrete state forms the current hybrid state $s(t_k) = (z(t_k), x(t_k))$. Hybrid automata for modeling relevant components of the environment introduced without input sets U (since not directly controllable).

Model Predictive Control. In order to generate control trajectories for the HA, the principle of *model predictive control* (MPC) is used (Morari et al., 1989). Considering the control problem not only as a motion planning problem like typically done in the robotic domain, e.g. (LaValle, 2006), but as an MPC problem has the following advantages: (1) an optimal solution for a given cost function and time horizon is computed, (2) model-based predictions for the behavior of the system and the environment lead to more reliable and robust results, and (3) a set of differential and dynamic constraints can be considered relatively easy. The MPC scheme solves, at any discrete point of time t_h, an optimization problem over a finite prediction horizon to obtain a sequence of optimal control inputs to the system. The optimization problem considers the dynamics of the system and the following additional constraints: a sequence of *forbidden regions* $\phi_{F,k} = \{F_k, F_{k+1}, \cdots, F_{k+p}\}$, and a sequence of *goal regions* $\phi_{G,k} = \{G_k, G_{k+1}, \cdots, G_{k+p}\}$ are specified over the prediction horizon $p \cdot (t_k - t_{k-1})$. Here, F_k denotes a state region that the system must not enter, and G_k is the state set into which the system should be driven. The constrained optimization problem can be formulated as:

$$\begin{aligned} \min_{\phi_{u,k}} \quad & J(\phi_{s,k}, \phi_{e,k}, \phi_{u,k}, p, \phi_{G,k}) \\ s.t. \quad & s(t_j) \notin F_j \; \forall j \in \{k, \ldots, k+p\} \\ & u_{min} \leq u(t_j) \leq u_{max} \\ & \phi_{s,k} \in \Phi_s \; and \; \phi_{e,k} \in \Phi_e \end{aligned}$$

where $\phi_{s,k}$ and $\phi_{e,k}$ are the predicted state trajectories of the system and the environment, which must be contained in the sets of feasible runs Φ_s and Φ_e. No state $s(t_j)$ contained in $\phi_{s,k}$ must be in a forbidden region $F(t_j)$. u_{min} and u_{max} are the limitations for the control inputs $u(t_j)$ and thus for the control trajectory $\phi_{u,k}$. A possible solution technique for the above optimization problem is the following sequential one: (1) an optimizer selects a trajectory $\phi_{u,k}$, (2) the models of system and environment are simulated for this choice leading (possibly) to feasible $\phi_{s,k}$ and $\phi_{e,k}$, (3) the cost function J is evaluated for these trajectories, and (4) the results reveals if $\phi_{u,k}$ should be further modified for improvement of the costs or if the optimization has sufficiently converged.

Knowledge Base and Learning. In order to improve the computational efficiency, the MPC scheme is enhanced by a knowledge-base, in which assignments of action sequences to situations is stored. The objective of the learning unit with a knowledge base is to reduce the computational effort by replacing or efficiently initializing the optimization. The situations in the knowledge base are formulated as tuple $\delta = (\phi_{s,k}, \phi_{e,k}, \phi_{G,k}, \phi_{F,k}, \phi_{u,k}, J)$, where $s(t_k)$ and $e(t_k)$ again denote the current state of the system and the environment, and $\phi_{u,k}$, $\phi_{G,k}$, and $\phi_{F,k}$ are the sequences of control inputs, the goal, and forbidden sets, respectively. For a given situation at the current time t_k, by using *similarity* comparison [1], the learning unit infers a proper control strategy $\phi^L_{u,k}$ if it exists. Otherwise, the optimization is carried out and the result is stored in the knowledge-base.

6 APPLICATION TO A KITCHEN SCENARIO

The presented hierarchical architecture is applied to a service robot in a kitchen scenario. The task of the robot is to lay a table (see Fig 3), i.e. the robot is expected to drive back and forth between a table and a kitchenette for positioning plates and cutlery on the table. The scenario obviously formulates a very challenging planning task, as the robot has to decide which object to take, how to move to the desired place at the table, and how to avoid collision with humans moving in the same space – this is the motivation for employing a decomposition-based planning approach. To simplify the upcoming presentation, the

[1] Similarity is here defined by small distances of the quantities specifying a situation in the underlying hybrid state space.

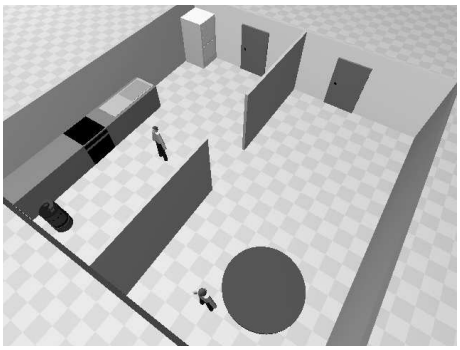

Figure 3: Setting of the kitchen scenario.

further discussion is here restricted here to the part of robot motion planning only.

Adaptation. On the uppermost layer, the set of components C comprises the relevant physical components of the setting and the choice of algorithms being available on the planning and control layers. Thus, the kitchen K, the table T, the kitchenette KN, the refrigerator R, the robot dynamics $B1$, the quantized robot dynamics $B2$, the persons P_1 and P_2, the reinforcement learning algorithm alg_1, the MPC algorithm alg_2, the learning algorithm alg_3, and the knowledge base kb are elements of the set of components for the motion scenario: $C = \{K, T, KN, R, B1, B2, P_1, P_2, alg_1, alg_2, alg_3, kb\}$. Modes are defined for the component structure like described above, i.e. they represent a particular subset of C according to $map : m \rightarrow C$. Note that modes may only affect parts of the system and that nesting of modes is possible (leading to alternative choices).

For the different steps in the motion planning problem, the adaptation chooses appropriate algorithms by switching to a corresponding mode. For the motion of the robot from one point to another within the kitchen, only one mode m_1 has to be be defined. It encodes the path planning by reinforcement learning for the planning layer and the calculation of the corresponding input trajectories to the dynamic system (including learning about situations) for the control layer. For this example, adaptation (in the sense of a switch between modes) occur only if the goal changes due to the current one being achieved or becoming unreachable.

Planning. The states $s \in S$ of the state transition system Σ for this particular example represent different positions of the robot within the kitchen. The actions $a \in A$ on the planning layer denote constant inputs u_i, $i \in \{1, \ldots, n\}$ (applied for a fixed time T) for the HA modeling the robot motion on the control layer. A discrete state of Σ represents the motion for time T, and it correspond to the trajectory obtained from integrating the resulting dynamics $f_z(x, u_i)$ specified on the control layer. Five constant inputs $u_1 = (1,0)^T$, $u_{2,3} = (0, \pm\pi/4)^T$, $u_{4,5} = (1, \pm\pi/4)^T$ encode pure translation, pure rotation, and curve transition respectively. The state set S of Σ represents a discretization of the floor space of the kitchen (modeled by two continuous variables x_1, x_2 on the lower layer). The discretization follows implicitly from the solution of the continuous dynamics on the control layer.

Choosing a reward of $r = -1$ for every taken action, the SARSA algorithm results in a sequence with a minimum number of actions. If the robot hits a static obstacle a reward of $r = -10$ is used to update the action-value function $Q(s,a)$. The outcome of the algorithm is depicted by the bold dashed line in Fig. 5, resulting from the concatenation of the action primitives. Σ consists in about 1000 states in this particular setup and the RL algorithm converges after approx. 80 runs (see Fig. 4), leading to a trajectory consisting about 20 actions. For a faster convergence, the action-value function $Q(s,a)$ is suitably initialized by a solution that leads to the goal position for the case that no dynamic obstacles are present.

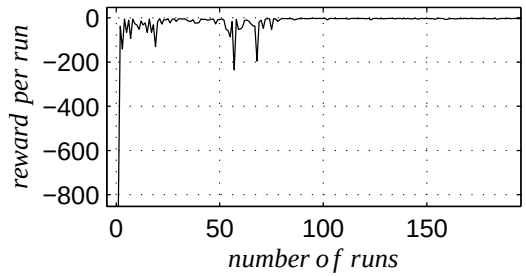

Figure 4: Reward improvement over repetitions of the motion.

Control. The HA used to model the robot that is regarded in the simulation scenario is defined with two locations z_1 and z_2. Location z_1 defines the kitchen domain, and location z_2 represents that the robot enters a circular region around one of the two moving persons being present in the kitchen; in z_2 the speed of the robot is reduced. Thus, the dynamics in the two locations are $f_1 = x(t_k) + (sin(\varphi)u_1(t_k),\ cos(\varphi)u_1(t_k),\ u_2(t_k))^T \cdot \Delta t$ and $f_2 = x(t_k) + 0.5(sin(\varphi)u_1(t_k),\ cos(\varphi)u_1(t_k),\ u_2(t_k))^T \cdot \Delta t$ with a time-step Δt, a position vector x and a heading angle φ. The objective function for the MPC algorithm specifies the deviation from the trajectory y obtained from the planning layer: $J = \sum_{p=1}^{H} \|x(t_{k+p}) - y(t_{k+p})\|_2^2$ with the prediction horizon H. For the computation, a step size of $\Delta t = t_{k+1} - t_k = 0.1$ seconds and horizon of $H = 10$ is chosen. The forbidden regions are defined

as circles around the persons and the goal set follows from the trajectory y.

The solution of the optimization problem ϕ_u, is stored in the knowledge base together with the observed situation. The similarity measure of situations, needed to extract a previously calculated control input, is defined in terms of the Euclidean distances of the current state of the system and position of the persons.

Fig. 5 shows by the dotted line as outcome of the MPC algorithm the motion of the service robot from the kitchenette (1) to the table (2) and further to the refrigerator (3). The underlying map represents the kitchen in 2D and is used for the localization of the robot. For three time stamps t_1, t_2, and t_3 the positions of the robot and the two persons are marked by a circle, and by a square / triangle respectively. The dash-dotted lines mark the motion of the two persons.

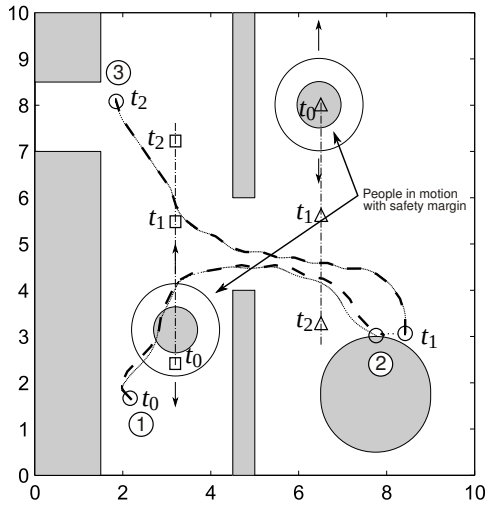

Figure 5: Trajectories of the robot moving between kitchenette (1), table (2), and refrigerator (3).

For each varying goal positions, the knowledge base is filled with information on goal-attaining strategies for future use. Employing the knowledge base reduces the calculation time essentially. The service scenario is simulated in the *Player/Stage/Gazebo* (PSG) environment.

7 CONCLUSIONS AND OUTLOOK

The aim of the presented concept is to provide an architecture by which the combination of control methods applied to hybrid systems, techniques from artificial intelligence, and vertical decomposition of tasks is realized. Simultaneously, the approach intends to cover the whole perception-reasoning-action loop of an autonomous system that has to make decisions for goal-oriented behavior in new situations. While the considered example is quite simple, it illustrates the mechanism of task decomposition and improves efficiency of computing suitable control trajectories by the use of machine learning techniques.

Current work is focussed on extending the planning hierarchy to different learning and control algorithms.

ACKNOWLEDGEMENTS

Partial financial support of this investigation by the DFG within the Cluster of Excellence *Cognition for Technical Systems* is gratefully acknowledged.

REFERENCES

Barto, A. G. and Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1-2):41–77.

Broy, M. and Stoelen, K. (2001). *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Springer.

Galindo, C., Fernandez-Madrigal, J., and Jesus, A. G. (2007). *Multiple abstraction hierarchies for mobile robot operation in large environments*. Studies in Computational Intelligence, Springer.

Haberl, W., Tautschnig, M., and Baumgarten, U. (2008). Running COLA on Embedded Systems. In *Proceedings of The International MultiConference of Engineers and Computer Scientists 2008*.

Henzinger, T. (1996). The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS '96)*, pages 278–292.

LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.

Morari, M., Garcia, C., and Prett, D. M. (1989). Model predictive control: Theory and practice-A survey. *Automatica*, 25(3):335–348.

Nau, D. S., Smith, S. J. J., and Erol, K. (1998). Control strategies in HTN planning: Theory versus practice. In *AAAI/IAAI*, pages 1127–1133.

Stursberg, O. (2006). Supervisory control of hybrid systems based on model abstraction and refinement. *Journal on Nonlinear Analysis*, 65(6):1168–1187.

Takadama, K. and Fujita, H. (2004). Lessons learned from comparison between q-learning and sarsa agents in bargaining game. In *Conference of North American Association for Computational Social and Organizational Science*, pages 159–172.