Logic-based and Robust Decision Making for Robots in Real World

(Demonstration)

Megumi Fujita
Graduate School of Humanities and Sciences, Nara Women's University, Nara, JAPAN
saboten@ics.nara-wu.ac.jp

Yuki Goto
Research Institute for Mathematical Sciences, Kyoto University, Kyoto, JAPAN

Naoyuki Nide
Faculty, Division of Natural Sciences, Nara Women's University, Nara, JAPAN

Ken Satoh
Principles of Informatics Research Division, National Institute of Informatics, Tokyo, JAPAN

Hiroshi Hosobe
Faculty of Computer and Information Sciences, Hosei University, Tokyo, JAPAN

ABSTRACT

We propose a logic-based mechanism for robot action decisions that is robust over the environmental noise of the real world and has a formal way to reason the possibility of achieving the robot's goal. Our experimental demos show that a robot can eventually reach its destination even if its actions are not that accurate.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Robotics

General Terms

Design

Keywords

Robots, Real world, Logic based, Action decision

Figure 1: Q.bo Lite Evo

1. INTRODUCTION

Recently, there has been growing interest in a combination of robotics and logic programming. In a logical way, the behavior of agents (or robots) can be expressed in a natural manner. In addition, logic-based decisions for actions can be used to check or certify that the robot can achieve its goal in a formal method. However, to reach the goal in the real world, the decision making of a robot has to be robust over variance of its action due to the noise of sensors and actuators.

In this paper, we propose a logic-based mechanism for the robot action decision process, which is being developed together with a logical framework to certify the robot's reachability to the goal formally (see Sec.3). Our method is much like teleo-reactive logic programs (TLPs) [2], but by using atomic actions designed reactively, the robot can eventually reach its goal in the real world, which involves dynamic changes. This will be a meaningful contribution toward realizing the general ability of robots to reach goals in the real world.

[†]This work was supported by Nara Women's University Intramural Grant for Young Women Researchers.

Appears in: *Alessio Lomuscio, Paul Scerri, Ana Bazzan, and Michael Huhns (eds.), Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014), May 5-9, 2014, Paris, France.*

2. APPLICATION DOMAIN AND PROBLEM SCENARIO

2.1 Application domain

We aim to develop a robot that can achieve its goal by thinking and acting for itself. Toward that aim, we chose the housekeeping problem as the application domain. Housekeeping requires appropriate movement under dynamic environment changes. For example, the position of furniture in the house changes. Therefore, we started with the problem of moving around the corridor and finding the target object under a dynamic environment. Therefore in this paper the robot's goal is to reach an objective point.

2.2 Problem scenario

We placed a robot and the target in a corridor. The target was put ahead of it, and it was informed of the rough direction of the target. There were no obstacles on the way to the target at the beginning. When it found the target, we set down an obstacle on the way to the target. The robot's goal is to reach the target while avoiding the obstacle.

2.3 Description of robot

We used the '`Q.bo` Lite Evo' robot (Fig. 1). A Spanish company, TheCorpora S.L., sells this robot, and distributes `Q.bo`'s particular Linux distribution based on Ubuntu.

`Q.bo` moves with two side wheels at the rear, and one caster wheel at the front. It has many applications, *e.g.* face recognition, speech recognition, and object recognition. We can control `Q.bo` with a robotic software platform called ROS (Robot Operating System)[4].

2.4 Technology used

For `Q.bo`'s motion planning, we wrote the action decision making program and computer programs that run `Q.bo`'s atomic actions. Atomic actions were written in Python, and the action decision making program was implemented in SWI-Prolog[5].

In our system, thinking and action were generally run by turn. First `Q.bo` runs the action decision making program to get a new atomic action. Second, `Q.bo` runs this atomic action while sensing in the real world, and passes the requisite information on to the action decision making program after the atomic action terminates. `Q.bo` continues to perform these two processes.

2.4.1 Atomic actions

We controlled `Q.bo` with ROS, which uses ROS Topics for sharing information. In this experiment, we used ROS Topics with a focus on motion planning. The requisite information (*e.g.* start a motor to move, recognize objects, measure distance, direction, and velocity, *etc.*) were shared using ROS Topics.

Atomic actions were made using these topics, which were updated in real time. Thus, `Q.bo` could move in response to changes in the environment in a way that is similar to reflex actions.

A detailed explanation of atomic actions follows[1].

- `looking_Qbo`
  - `Q.bo` looks for the target while moving its head. This action returns the direction of the target, when `Q.bo` detects it.
  - If `Q.bo` cannot detect the target, this action simply returns the rough direction given in Sec. 2.2.
- `search_Qbo(Direction)`
  - `Q.bo` turns around, searches for a direction with no obstacles, and moves forward in this direction. When `Q.bo` searches for such a direction, it makes an effort to choose the direction that is as close to the argument `Direction` as possible.
- `forward_Qbo(Direction)`
  - `Q.bo` moves ahead for a fixed distance or until an obstacle is found.
  - If `Q.bo` finds an obstacle when it is moving forward, it stops moving, and terminates this process.

In `looking_Qbo`, the current version of object recognition for finding the target was implemented using libSVM[1]. It was trained using about 40 images of the target and 40 images of different things, and classified the images from `Q.bo`'s camera into images of the target and images of different things. Currently the accuracy of the classification is not that high. We are planning to compare this implementation with one implemented using OpenCV.

2.4.2 Action decision making program

`Q.bo` ran action decision making program to get a new atomic action that was suitable for the circumstances at that time. Action decision making program is implemented in Prolog. The derivation of Prolog enables `Q.bo` to infer a new atomic action, and the unification in Prolog enables `Q.bo` to update the requisite information such as sensor information.

By and large, two kinds of rules made up the action decision making program. One group relates to storing sensor information, and the other relates to conditional execution of atomic action.

Abstract descriptions of the action decision making program were as the following pseudo-Prolog code[2].

```
Toplevel :- Initialize, Goal.
Goal :- Percept, Goal'.
Goal' :- Condition, !, Atomic_action, Goal.
```

The following is a detailed explanation of the action decision making program.

1. Initialize: Initialize information from stored sensor information of `Q.bo`. In particular, `Initial_state/3` is updated to record `Q.bo`'s initial direction and location.
2. Perception: `Q.bo` runs a specific atomic action (*e.g.* `looking_Qbo`) to have external perceptions to get some information like its position and direction.
3. Branch condition: The branch that is the first to have its condition satisfied with the perceptions is chosen to execute.
4. Run an atomic action: `Q.bo` runs the following atomic action (*e.g.* `search_Qbo/1`, `forward_Qbo/1`). At this point, `Q.bo` waits for termination of the atomic action.

The action decision making program continues to repeat 2∼4.

3. THE INNOVATION OF THE SYSTEM

Our method of generating and executing actions is logically much similar to that of TLPs. That method robustly directs an agent toward a goal in a manner that continuously takes into account the agent's changing perceptions of a dynamic environment. However, most studies for application using teleo-reactive logic programs do not use actions associated with sensing; they only deal with problems in which the effects of actions can be accurately modeled (e.g. block worlds, driving in simulators). In contrast, in our method, the robot can respond to dynamic continuous changes of the environment, where we cannot model accurately, using reactive actions.

We are currently developing a logical framework to certify the possibility of achieving the robot's goals in a formal way. We are planning to construct it based on our extended BDI-logic with probabilistic semantics named $\mathcal{TOMATO}$[3], and its goal is to show that the possible range of the robot's existence 'converges' to the target point.

4. CONCLUSION

As the demo shows, the robot can reach its destination while its atomic actions are rather rough. Our future plan, in addition to trying larger and more realistic examples, is to move ahead with developing a formal way to verify the robot's actions further.

5. REFERENCES

[1] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[2] P. Langley and D. Choi. Learning recursive control programs from problem solving. *Journal of Machine Learning Research*, 7:493–518, 2006.

[3] N. Nide, S. Takata, and M. Fujita. BDI logic with probabilistic transition and fixed-point operator. In *Proc. of CLIMA '09*, pages 71–86, 2009.

[4] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[5] J. Wielemaker, T. Schrijvers, M. Triska, and T. Lager. SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1–2):67–96, 2012.

[1] Any atomic action terminates eventually, *i.e.* no action has a timeout. In particular, `search_Qbo` currently assumes that it can eventually find a direction without an obstacle.

[2] For simplicity, we did not consider what happens after the robot reached the target. To do so, we can add a clause like "`Goal' :- Termination_Condition, !.`".