An Architecture for Autonomously Controlling Robot with Embodiment in Real World

Megumi Fujita\textsuperscript{1}, Yuki Goto\textsuperscript{2}, Naoyuki Nide\textsuperscript{3}, Ken Satoh\textsuperscript{4}, and Hiroshi Hosobe\textsuperscript{5}

\textsuperscript{1} Graduate School of Humanities and Sciences, Nara Women’s University, Nara, JAPAN
saboten@ics.nara-wu.ac.jp
\textsuperscript{2} Research Institute for Mathematical Sciences, Kyoto University, Kyoto, JAPAN
\textsuperscript{3} Faculty, Division of Natural Sciences, Nara Women’s University, Nara, JAPAN
\textsuperscript{4} Principles of Informatics Research Division, National Institute of Informatics, Tokyo, JAPAN
\textsuperscript{5} Faculty of Computer and Information Sciences, Hosei University, Tokyo, JAPAN

Abstract. In the real world, robots with embodiment face various issues such as dynamic continuous changes of the environment and input/output disturbances. The key to solving these issues can be found in daily life; people ‘do actions associated with sensing’ and ‘dynamically change their plans when necessary’. We propose the use of a new concept, enabling robots to do these two things, for autonomously controlling mobile robots. We implemented our concept to make two experiments under static/dynamic environments. The results of these experiments show that our idea provides a way to adapt to dynamic changes of the environment in the real world.

Keywords: Autonomous Control, Dynamic environment, Motion planning

1 Introduction

We aim to make robots who decide their actions to achieve their goals in the real world.

It is known that robots in the real world are exposed to various issues that are never present in the virtual or formally modeled worlds [1]. One of them is that, as is widely recognized, the real world is highly dynamic; there are continuous changes of circumstances. Another is that the physical devices can never escape from input/output disturbances, which cause failures in robots’ actions. These issues arise from the fact that robots have embodiment in the real world.

The key idea to solving these issues can be discovered in daily life. When we walk, we are sensing various things, e.g. traffic signals, the roaring of a car engine, the presence of vending machines, while moving ahead. We may stop moving since the traffic signal is red, or reflexively jump back since we sense danger from the roaring. Here, sensing and action work together rather than being separated, and the plan is updated dynamically. We use such actions, e.g. ‘walking forward until finding some obstacle’, to construct plans to achieve our goals. The robots who have embodiment also need to have such ability to act in the real world.

However, traditional planning theory separates sensing and actions. Since the origin of a conceptual model for planning [2] is the model of state-transition systems (also
called discrete-event systems), it equates an atomic action with a step in a state machine. In other words, atomic actions are considered to be carried out in a moment, and no sensing takes place during that action. As discussed above, this feature is not suitable for implementing such behaviors.

We propose a way to implement the key idea described above. To do this, we associate actions with sensing in implementation rather than separating them, and instead let the next action be selected dynamically. In our system, Atomic actions are implemented with sensing and storing external perceptions. This is the main difference between general automated planning and ours. (Fig.1)

The action decision making program, which selects suitable atomic actions, is implemented by Prolog. It takes the robot’s goal as a Prolog goal, and decomposes it to the next action and the remaining subgoal; i.e. each step of the derivation procedure of Prolog and the robot's each action are interconnected. We also describe our experiment on motion planning in the real world.

This paper is organized as follows: Sec.2 explains the difficulty of making motion planning with dynamic planners and describes how our research has overcome it. Sec.3 describes in detail the robot Q.bo which we used in the experiment. The following section Sec.4 shows aspects of our computer programs, and how they work. In Sec.5, we show our experiments in the real world. Related works are shown in Sec.6. Finally, in Sec.7, we provide some discussions on the remaining issues and conclude.

2 Motion planning in real world

Many “dynamic planners” have been published, which respond to dynamic changes in the environment.

For example, in [3], Ayan et al. use it for Noncombatant Evacuation Operations planning (e.g. selecting the means of transportation and the route to be followed), and in [4], Sánchez-Garzón et al. use it for therapy planning systems in a real clinical environment.
However, traditional dynamic planners in the real world cannot avoid making a significant assumption that the world can be accurately described as a state-transition system based on atomic actions. In other words, it must be assumed that the effects, pre- and post-conditions of atomic actions can be accurately described. However, due to the I/O disturbances of devices described in Sec.1, for example, ‘to go ahead exactly with 1 meter toward north’ is very hard for robots in the real world. If a robot tried to do so and actually proceeded by 0.98 meter, a dynamic planner would not consider that the robot has normally reached the expected state, and it will consider that situation as a failure of the plan.

To overcome this difficulty, we have taken a different approach. We implement atomic actions analogous to our natural behaviors in daily life discussed in Sec.1, as follows.

– Sensing and actions working together
  Instead of the ‘sensing after action’ manner used by traditional dynamic planners, we combine sensing and action, just like we sense surroundings while we walk. In our method, an atomic action is done while always receiving data from the sensor, and it terminates when some terminating condition (e.g. a certain period of time has been passed, an obstacle is found, etc.) is satisfied. At that time, information about the current environment of the robot will be sent to the action decision making program which is used to determine the next action. Any atomic action is guaranteed to terminate eventually (or, to be strict, we design atomic actions to satisfy this property).

– Dynamic action selection depending on current condition
  Using the information sent from the sense-and-action part described above, the action decision making program decides the best suited atomic action to be executed next. It uses rules written in Prolog and always returns some atomic action (see Sec.4.2). It is very much like teleoreactive logic programs [5] (but see Sec.6).

In Sec.5, by our experiments, we show that even if the robot’s actions are not sufficiently accurate, it can continue to act and eventually reach its goal. In the experiments, the robot sometimes lost sight of the target due to the inexactness of the object recognition routine implemented in the SVM (Support Vector Machine) [6]. However, while going toward the approximate direction of the target, it found the target again and went toward it. We also show that, under both the static environment and the dynamic one, the robot reached the target using the same pair of action decision making program and atomic action set.

3 Detailed description of robot

We used the robot ‘Q. bo Lite Evo’. A Spanish company, TheCorpora S.L., sells this robot, and distributes Q. bo’s particular Linux distribution based on Ubuntu. We illustrate Q. bo in Fig.2.

Q. bo moves with 2 side wheels (rear), and 1 caster wheel (front). There are many applications, e.g. face recognition, speech recognition, and object recognition, and so on. We can control Q. bo with a robotic software platform, Robot Operating System (ROS) [7].
4 Motion planning

For motion planning, we write the action decision making program and computer programs that run Q.bo’s atomic actions. Atomic actions are described with Python, and the action decision making program is implemented with SWI-Prolog [8].

In our system, thinking and action are generally run by turn. At first Q.bo runs 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 action decision making program after the atomic action terminates. Q.bo continues these two processes.

4.1 Atomic actions

We control Q.bo with ROS, which uses ROS Topic for sharing information. In this experiment, we use 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.) are shared using ROS Topics.

Atomic actions are made using these topics, which are updated in real time. In this way, Q.bo can move in response to changes in their environment in a way that is similar to reflex actions.

A detailed explanation of atomic actions follows6.

- looking_Qbo
  - Q.bo looks for the target while moving his head. It returns the direction of the target, when he detects it.
  - If he cannot detect the target, it does nothing and terminates this process.

---

6 As mentioned in Sec.2, 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.
- search_Qbo(Direction)
  • Q.bo turns around, searches for a direction, in which there are no obstacles, and Q.bo moves forward in this direction. When Q.bo searches for such a direction, he makes an effort to choose the direction as close to the argument Direction as possible.
- forward_Qbo(Direction)
  • Q.bo moves ahead for a fixed distance until some obstacle is found.
  • If Q.bo finds some obstacle when he is moving forward, stops moving, and terminates this process.

In looking_Qbo, object recognition for finding the target is currently implemented using libSVM [9]. It was trained using about 40 images of the target and 40 images of the different things, and classifies the images from Q.bo’s camera into ones of the target and of the different things. Currently the accuracy of the classification is not so high. We are planning to compare this with one implemented using OpenCV.

4.2 action decision making program

Q.bo runs action decision making program to get a new atomic action that is suitable for the present circumstance. As shown in Sec.4, action decision making program is implemented with Prolog. The derivation of a Prolog enables Q.bo to infer a new atomic action, and the unification in a Prolog enables Q.bo to update requisite information such as sensor information.

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

Abstract descriptions of the action decision making program are as the following pseudo-Prolog code.

Toplevel :- Initialize, Goal.
  Goal :- Percept, Goal’.
  Goal’ :- Condition, !, Atomic_action, Goal.

The following is a detailed explanation of 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 some 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, which is the first to have its condition satisfied with the perceptions, is chosen to execute.

4. run a atomic action
   Q.bo runs following atomic action. At this point, Q.bo waits for a termination of the atomic action.
The action decision making program continues to repeat from 2 to 4. We show almost the whole action decision making program code below.

```prolog
/* Action decision making routine. First rule of search_target/7
is selected at only the first time that initial state
information was obtained and returns no_operation action. */
search_target(D,X,Y,Op,Obj,Input,Output) :-
  retract(first),
  assert(initial_state(D,X,Y)),
  Op = none.
/* The main rule for Qbo’s action decision making. */
search_target(D,X,Y,Op,Obj,Input,Output) :-
  get_directions(D,I),
  around_search(F,FD,Input,Output),
  decide_action(F,FD,D,I,Op,Obj).
/* Get the directions which Qbo towards at the initial state. */
get_directions(Direction,Initial_Direction) :-
  initial_state(Initial_Direction,_,_).
/* Send the command "looking_Qbo" to Qbo for looking around
and search the target. */
around_search(Found,Found_Direction,Input,Output) :-
  write(Output,looking_Qbo),nl(Output),flush_output(Output),
  recognize_target(Found,Found_Direction,Input).
/* If Qbo recognizes the target, Qbo sends back "True" message
and the argument 'Found' is unified to it. Otherwise,
'Found' is unified to "False". */
recognize_target(Found,Found_Direction,Input) :-
  read_line_to_codes(Input, T1, T2),
  name(Found, Found_Direction, T1, T2).
/* If Qbo recognizes the target, Qbo tries to go toward the
target. */
decide_action(F,FD,D,I,Op,Obj) :-
  F = true, !,
  write('target found'), nl,
  go_forward(D,FD,Op,Obj).
/* Otherwise, Qbo goes toward the initial direction. */
decide_action(F,FD,D,I,Op,Obj) :- !,
  write('target not found'), nl,
  go_forward(D,I,Op,Obj).
/* The first rule of go_forward/4 sends the command which
```
 leads Qbo to go forward avoiding the obstacle. */
go_forward(Direction, Initial_Direction, Operator, Obj) :-
  Obj = 1, !,
  Operator = search_Qbo(Initial_Direction).
/* The second one sends the command to go forward by a fixed
   distance until Qbo finds the obstacle. */
go_forward(Direction, Initial_Direction, Operator, Obj) :- !,
  Operator = forward_Qbo(Initial_Direction).
/* Top level routine (which acts as a TCP client); creates
   socket for connecting to the program (a TCP server)
   managing Qbo’s atomic action, and calls the robot’s
   goal (start searching for the target). */
client(H, P):-
tcp_socket(S),
tcp_connect(S, H:P, I, O),
prompt(_, ''),
searching(S, I, O).
/* Robot’s goal; Searching the target. */
searching(S, I, O) :-
  /* Read the sensor information from Qbo */
  read_line_to_codes(I, T1,T2,T3,T4),
  name(D,X,Y,Obj,T1,T2,T3,T4),
  /* Call the action decision making routine. */
  search_target(D,X,Y,Operator,Obj,I,O),
  /* Send the atomic action Operator to Qbo. */
  write(O, Operator), nl(O),
  flush_output(O),
  /* Recursively call this goal. */
  searching(S, I, O).

5  Experiments in real world

We carried out the following two experiments.

- **First experiment**
  We placed Qibo, an obstacle, and the target in the space for our experiments. The target is put ahead of Qibo, but the obstacle obstructs the way to the target. Qibo’s goal is to reach the target while avoiding the obstacle.

- **Second experiment**
  We placed Qibo, and the target in the same space. The target is put ahead of Qibo again, and there are no obstacles on the way to the target at the beginning. When Qibo finds the object, we set the obstacle on the way to the target. Qibo’s goal is to reach the target, avoiding the obstacle. (This is equivalent to the prior goal.)
The first experiment shows how our programs work under the static environment. The second one shows how to get over dynamic changes of the environment in the real world. In these two experiments, we use the same action decision making program and atomic actions because the goal of these is the same, i.e. “reach the target”.

5.1 Experiment results

The first experiment result is shown in Fig.3, and the log messages of this experiment are shown in Fig.4. The result of the second experiment is shown in Fig.5. We now give a detailed description of our experiments of these figures in turn.

First, Q.bo, the target and the obstacle are set on the floor as shown in (1) of Fig.3. He found the target and started to move forward to it (2). However, he could not go...
We also explain log messages Fig.4 in detail. He found the target and the obstacle simultaneously (1) and (2). The central part of Fig.4 shows the situation of (3) and (4). The end of this shows the situation of (5).

In the second experiment, \textit{Q. bo} and the target were set on the floor (1) as shown in Fig.5. He found the target and went towards it (2), and we put a tall obstacle in front of the target to prevent him from seeing the target (3). Though he became unable to find the direction of the target, he knew the direction which he faced at the initial state. Thus, to continue along that direction he avoided this obstacle (5) and he found the target again (6). He went forward to the target (7), and he reached a point in front of it (8).

6 Related works

Our method of generating and executing actions is described in Sec.2, is logically much similar to that of teleoreactive logic programs [5] in that problem solving (action selection) and skill execution are run in turn. However, most studies using teleoreactive 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 creates its behavior by concatenating actions with sensing, such as ‘proceed until some obstacle is found’. Thus, in the real world, we can leave absorption of inaccuracy of actions to low-level actions, and high-level action decision making program can concentrate on essential problem solving (such as route finding)\textsuperscript{8}.

On the other hand, since the basis of making behaviors are common to both, it is promising that some techniques effective in teleoreactive logic programs can also be applied to our method. For example, [5] refers to learning skills by generating logic programs. It is also possible in our method in principle by dynamically asserting new Prolog clauses.

Several other pieces of work on robot control combine high-level goal selection using Prolog and low-level controlled atomic actions.

Pozo et al.’s [10] presented a system which consists of a planner based on the situation calculus written in Prolog and a Visual Basic program to control a mobile robot via the serial port using the plan. However, unlike our system, it is based on common planning theory and does not take environmental changes while performing atomic actions into consideration.

Nalepa et al. [11] proposed the use of a Prolog-based design of implementing embedded control systems. However, it is principally aimed at controlling devices such as mobile phones and elevator systems, and does not take real-world robots into account. In a similar way, Matyasik et al.’s work [12] provides a controller of the Stenzel Ltd.

\textsuperscript{8} One possible policy for classifying low- and high-level actions is to distinguish reflexive actions and deliberative ones. For example, when we go somewhere by bicycle, we keep our posture reflexively, and select a route to the destination by deliberation.
Connect to Qbo’s atomic action managing program. Data from Qbo. (Qbo's current direction, X,Y coordinations, judgment on existing obstacle.) Operator for Qbo’s moving. (Qbo does nothing initially.) Data on whether Qbo finds the target or not.

Data on whether Qbo finds the target or not.

Qbo found obstacle in front of him. Qbo found target.

Qbo avoided obstacle.

Qbo continued avoiding obstacle, but lost sight target.

Qbo found target again.

Qbo went towards the target.

Qbo lost sight of target again. (Probably Qbo’s object recognition function did not detect target.)

Qbo went towards the target and reached a point in front of it.

Qbo found target.
Fig. 5. State of second experiment using Q. bo
HEXOR mobile robot, but it does not deal with inaccuracies of robots’ moves in the real world and lacks a way to hide these inaccuracies from high-level action controls.

Qureshi et al. [13] provided a space robotics system by using a combination of a visual perception system and a high-level reasoning system using GOLOG [14]. It is specialized in space robotics tasks such as rendezvous and docking while our system is intended to provide a generalized architecture that combines high-level goal selections and atomic actions which are robust enough to bear up under inaccuracies in the real world.

Some other researches introduce actions associated with sensing, as we also do so. A work of Chen et al. [15], which intends human-robot collaboration (unlike ours), can treat rather large-scale actions (e.g., ‘goto a location’ and ‘pick-up an item’) as atomic actions, which can include sensing. However, it does not directly mention the issues in the real world described in Sec.1, such as I/O disturbances, and failures in actions caused by them. Besides, we think that ‘goto a location’ is too large to be treated as an atomic action; it should be better to regard it as a subgoal for which we can choose a plan by deliberation.

KNOWROB by Tenorth et al. [16] is an action-centered knowledge representation system which can learn action models. It can build complex actions including sensing as action classes. It also mentions handling of uncertainties such as sensor noise. However, since it is a knowledge representation system, It does not mention how to overcome such uncertainties in itself.

7 Conclusions

We present an architecture for motion planning, which can respond to dynamic changes in the environment and also is able to deal with uncertainties in the real world. We incorporate sensing into atomic actions because we have to take the robot’s embodiment into consideration. Atomic actions are similar to everyday actions done using sense data.

Though our experiment is still in the basic stage and we are dealing with a small goal of reaching the single target, we expect that our method will be able to handle larger scale problem solving. We plan to clarify the practicality of our method through more pragmatic problems such as handling multiple goals and utilizing beliefs gained by past perceptions. Increasing the precision of atomic actions is another future issue.

While there are many works about planning, not many reports about planning have been published that deal with difficulties which come from embodiment such as mentioned in Sec.1. We hope that this kind of research increases as a result of ours.

Acknowledgments

This work was supported by Nara Women’s University Intramural Grant for Young Women Researchers.

9 As mentioned in Sec.2, it is a feature of our method that the atomic actions need not be accurate enough. However, increasing the precision of them can improve the behaviors of robots.
References

1. Pfeifer, R., Bongard, J.C.: How the Body Shapes the Way We Think: A New View of Intelligence. The MIT Press (2006)
2. Malik, G., Nau, D., Traverso, P.: Automated planning: theory and practice. The Morgan Kaufmann Series in Artificial Intelligence Series. Elsevier/Morgan Kaufmann Publishers (2004)
3. Ayan, N.F., Kuter, U., Yaman, F., Goldman, R.: HOTriDE: Hierarchical Ordered Task Replanning in Dynamic Environments. In: Principles and Practices for Planning in Execution. (2007) 31–36
4. Sánchez-Garzón, I., Fdez-Olivares, J., Castillo, L.: Monitoring, Repair and Replanning Techniques to support Exception Handling in HTN-based Therapy Planning Systems. In: International Workshop on Artificial Intelligence in Healthcare and Biomedical Applications (CAEPIA 2011). (2011)
5. Langley, P., Choi, D.: Learning recursive control programs from problem solving. Journal of Machine Learning Research 7 (2006) 493–518
6. Bishop, C.M.: Sparse kernel machines. In: Pattern Recognition and Machine Learning. First edn. Springer (2007) 325–358
7. Quigley, M., Conley, K., Gerkey, B.P., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source robot operating system. In: ICRA Workshop on Open Source Software. (2009)
8. Wieliemaker, J., Schrijvers, T., Triska, M., Lager, T.: SWI-Prolog. Theory and Practice of Logic Programming 12(1–2) (2012) 67–96
9. Chang, C.C., Lin, C.J.: LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology 2 (2011) 27:1–27:27
10. Pozos, P., Yescas, E., Vásquez, J.: Planning using situation calculus, Prolog and a mobile robot. In: Latin-American Workshop on Non-Monotonic Reasoning, Proc. of the LAMR07 Workshop. Volume 286 of CEUR Workshop Proceedings. (2007)
11. Nalepa, G.J., Ziecik, P.: Integrated embedded Prolog platform for rule-based control systems. In: Proc. of MIXDES 2006: MIXed DESign of integrated circuits and systems. (2006) 716–721
12. Matyasik, P., Nalepa, G.J., Ziecik, P.: Prolog-based real-time intelligent control of the Hexor mobile robot. In: Proc. of KI 2007: Advances in Artificial Intelligence, 30th Annual German Conference on AI. Volume 4667 of Lecture Notes in Computer Science. (2007) 485–488
13. Qureshi, F., Terzopoulos, D.: Intelligent perception and control for space robotics: Autonomous satellite rendezvous and docking. Machine Vision and Applications 19(3) (2008) 141–161
14. Levesque, H.J., Reiter, R., Lespérance, Y., Lin, F., Scherl, R.B.: GOLOG: A logic programming language for dynamic domains. Journal of Logic Programming 31(1–3) (1997) 59–83
15. Chen, X., Ji, J., Jiang, J., Jin, G., Wang, F., Xie, J.: Developing high-level cognitive functions for service robots. In: Proc. of AAMAS 2010: 9th Intl. Conf. on Autonomous Agents and Multiagent Systems. (2010) 989–996
16. Tenorth, M., Beetz, M.: KnowRob – knowledge processing for autonomous personal robots. In: Proc. of IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems. (2009) 4261–4266