Robotics and Autonomous Systems 62 (2014) 545–555

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Targets-Drives-Means: A declarative approach to dynamic behavior specification with higher usability

V. Berenz [a,b,*], K. Suzuki [c]

[a] RIKEN Brain Science Institute, Japan
[b] University of Tsukuba Artificial Intelligence Laboratory, Japan
[c] Department of Intelligent Interaction Technology, Faculty of Engineering, Information and Systems, University of Tsukuba, Japan

ARTICLE INFO

Article history:
Received 16 April 2013
Received in revised form
20 November 2013
Accepted 28 December 2013
Available online 9 January 2014

Keywords:
End-user programming
Usability
Application layer
Behavior specification
Dynamic behaviors
Commercial humanoid robots
TDM

ABSTRACT

Small humanoid robots are becoming more affordable and are now used in fields such as human–robot interaction, ethics, psychology, or education. For non-roboticists, the standard paradigm for robot visual programming is based on the selection of behavioral blocks, followed by their connection using communication links. These programs provide efficient user support during the development of complex series of movements and sequential behaviors. However, implementing dynamic control remains challenging because the data flow between components to enforce control loops, object permanence, the memories of object positions, odometry, and finite state machines has to be organized by the users. In this study, we develop a new programming paradigm, Targets-Drives-Means, which is suitable for the specification of dynamic robotic tasks. In this proposed approach, programming is based on the declarative association of reusable dynamic components. A central memory organizes the information flows automatically and issues related to dynamic control are solved by processes that remain hidden from the end users. The proposed approach has advantages during the implementation of dynamic behaviors, but it requires that users stop conceiving robotic tasks as the execution of a sequence of actions. Instead, users are required to organize their programs as collections of behaviors that run in parallel and compete for activation. This might be considered non-intuitive but we also report the positive outcomes of a usability experiment, which evaluated the accessibility of the proposed approach.

1. Introduction

Several commercially available humanoid robots are ubiquitous programmable tools, rather than being designed to perform specific tasks, including Nao by Aldebaran Robotics, DARwIn-OP by Robotis, and Palro from Fujisoft. End user programming is essential for these products, where users are required to specify behavioral tasks. Humanoid robots are becoming more accessible and this means that the usability of such software is increasingly important. Nao, the humanoid robot sold by Aldebaran robotics, is used in research areas such as education [1], psychology [2] and robot ethics [3]. In these areas, the robot users do not necessarily have backgrounds in programming or robot behavior design.

Visual programming environments are available that allow non-programmers to create robot applications. The current standard paradigm for robot visual programming is based on the selection of behavioral blocks, followed by their connection to obtain the desired flow of action [4,5]. These programs provide very efficient support by allowing users to develop complex series of movements and sequential behaviors. However, the programming of autonomous dynamic behavior remains challenging. This is mainly because the current programming paradigm enforced by visual programming software (the selection and connection of behavior blocks) requires that users manually specify the information flow between components.

Thus, the present study focused on the design of a new programming paradigm, which is suitable for the specification of dynamic robotic tasks, and a usability evaluation was performed. In particular, we considered how the features used by roboticists to handle issues related to dynamic control can be manipulated via the simple association of reusable components. These features include resource management, open intrinsic motivation, asynchronous exchange of data via a centralized memory, schematic representation of information, and the use of dynamic components. Thus, we developed Targets-Drives-Means (TDM), which is an architecture based on a new paradigm for behavioral specification that employs declarative associations of reusable components.

* Corresponding author at: RIKEN Brain Science Institute, Japan. Tel.: +81 9018458572.
E-mail address: vincent@brain.riken.jp (V. Berenz).

0921-8890/$ – see front matter © 2014 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.robot.2013.12.010

Previously, TDM was tested in terms of code reusability [6]. The core engine and the libraries of TDM have been made open source.[1]

The proposed approach has advantages during the implementation of dynamic behaviors, but it requires that users stop conceiving robotic tasks as the execution of sequences of actions. Instead, users are required to organize their program as collections of behaviors that run in parallel and compete for activation. This non-intuitive change in the programming paradigm is demanding and it was not clear whether it would be suitable for inexperienced users of humanoid robots. Thus, we also conducted a usability experiment to evaluate the accessibility of the proposed approach. These tests focused on the usability of the proposed programming paradigm, rather than the usability of a particular graphical implementation.

In the next section, we clarify our research focus by analyzing the difficulties of programming dynamic behavior and we justify the need for a novel solution. In Section 3, we describe our proposed approach and Section 4 presents the results of the usability tests. Finally, we discuss the results obtained in this study.

2. Programming dynamic behavior

Programming dynamic behaviors for small commercial humanoid robots is a complicated task for inexperienced roboticists, where the users of robots must achieve the following.

- Organize the information flow between modules that run at different rates.
- Ensure robustness and the continuity of actions for possibly unreliable sensory modules.
- Implement memories of the existence and positions of detected objects.
- Organize the information flow from sensory inputs to distinguish between objects.
- Implement the desired logic for action selection, while considering the requirement for action correction in case of failure.

In this section, we explore these difficulties and provide further details based on an example. We also justify the need for a new solution by showing how the standard flowchart approach fails to support users in overcoming these difficulties.

2.1. Difficulties of dynamic control

Typically, beginners will view tasks as a succession of sub-tasks. For example, ''kicking the closest ball'' could be split into the following sequence: 1. search for the balls, 2. evaluate the distance, 3. select the closest ball, 4. walk to the selected ball, and 5. kick the selected ball. However, this description ignores the reactive aspects, which are essential for ensuring the robustness of execution. For example, the walking parameters should be updated online to compensate for errors and to ensure that an appropriate reaction occurs if the ball is moved. This implies that there must be online re-estimation of the position of the ball and a head tracking system should be implemented to ensure that the ball remains in the field of vision. This raises the issue of data exchange between modules which must run in parallel at different rates.

Another issue is ensuring the continuity of behavior if the ball is out of sight. This may occur if the head tracking system and/or the computer vision program are unreliable. Thus, occasional updates of the position of the target ball are required based on odometry. But also the logic for differentiating between non-detection due to temporary limitation of the detection system and the robot definitely losing track of the ball must be implemented, and the necessary actions have to be programmed (e.g. stopping the walking and searching for the ball again).

If several balls have been detected, the robot must re-evaluate in a continuous manner the ball to be kicked. Thus, the positions of balls that are not currently in the field of vision must also be maintained in a memory and updated using odometry. The simultaneous detection of several balls implies that there is a risk of the robot switching repeatedly between walking toward one ball to walking toward another. The system must enforce a notion of object permanence that differentiates between these two balls and the appropriate information flow needs to be managed between sensing and acting modules. This implies a system that systematically treats the continuous uniform information input from the sensory module and process it into knowledge about distinct objects. This system must be applicable to an unknown number of balls. Advanced features (e.g. obstacle avoidance) require the robot to share its attention between several objects without stopping walking, meaning complex head motion suitable for multi-object head-tracking must be performed.

Furthermore, dynamic behaviors can never be described fully by a succession of fixed sub-tasks. For example, if a ball is removed while the robot is walking toward it, the expected action sequence is interrupted and the robot must either redirect itself to another ball or restart the ball-searching routine.

2.2. Solutions for dynamic control

Solutions have been proposed that address the problems related to dynamic control in a systematic manner. For example, URBI [7] and TDL [8] are programming languages, which were developed specifically for robot programming. URBI provides a syntax that allows the organization of code into different processes that either run sequentially or in parallel, and it also provides tools to monitor their execution. In TDL, the code is organized into task trees, which encode the hierarchical decomposition of tasks as well as the synchronization of constraints between tasks. A large range of robotic architectures have been proposed, including solutions based on hierarchical control architectures [9], reactive architectures [10], and hybrid systems [11–14]. More recently, specialized operating systems have been proposed, such as ROS [15], which organize code into asynchronous modules that exchange data using a data subscription protocol. Hierarchical organization of behaviors and modularity are also being investigated [16–18].

These solutions manage the integration and communication between different types of hardware and software and support the implementation of reaction, as well as behavioral specification. However, these programs have been produced by robot developers and are targeted at this community, which means that they are not intended to be used by non-roboticists and a solid background in computer science/robotics is required for their use.

2.3. Software that facilitates greater usability

An approach that provides flexible architectures is the integration of configurable modules. Two examples of these architectures are SAFSR for service robots [19] and B3IA for the control of robots, which are used as intervention tools for children with autism spectrum disorders [20]. SAFSR and B3IA are configurable but are also domain-specific.

2.4. Emotionally Grounded Architecture (EGO)

EGO was developed for managing behaviors in Sony Aibo and QRIO [21]. In EGO, behaviors are specified by organizing modules

[1] http://www.ai.iit.tsukuba.ac.jp/research/tdm/.