Roboko: Integrating Visual Representations into a Text-based Programming Environment

Jun Kato
The University of Tokyo
i@junkato.jp
http://junkato.jp/roboko

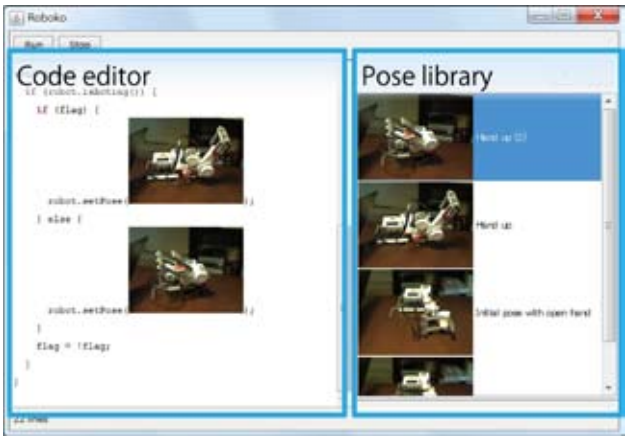
1. Project Goal

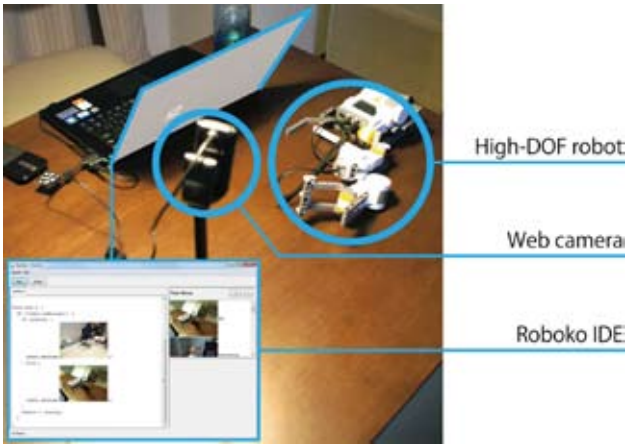
Current programming environments use textual or symbolic (e.g. flow chart) representations. This paradigm has worked well for programming abstract behaviors of software, but is problematic for programming robots that work in the real environment. Robots have a physical embodiment and work in the real world. Textual or symbolic representation does not help people to understand how the robot works in run time. For example, if the programmer wants to program a high degree-of-freedom (high-DOF) robot to reach an arm, each joint angle must be specified in the code. However, it is difficult to understand that the code is for arm reaching by reading the code. Our goal is to address this problem by integrating photograph-based visual representations into a text-based development environment. The integrated development environment (IDE) allows programming-by-example style instructions for robots.

2. Technical breakthrough

Our prototype implementation of the IDE consists of two main components: a code editor and an pose library (Figure 1). Each photograph-based visual representation is shown inline in the editor and is bound to a set of raw pose data. In this section, we will describe the workflow of the programmer with our development environment and briefly explain its implementation.

First, the programmer captures a photograph and a set of raw pose data. He clicks "Capture" button in the pose library and the IDE enters "capture mode." During this mode, he can move every joint of the robot, whose actuators are equipped with rotary encoders, to make a desired pose. When he is satisfied with the current pose, he takes a photograph of the robot with a web camera (Figure 2.) Then, the IDE retrieves corresponding pose data from the robot. Finally, the captured pose appears as the photograph with an arbitrary name (e.g. *"New pose (1)"*) in the pose library and the IDE leaves "capture mode." Within the pose library, the programmer can also rename existing robot pose (e.g. *"Initial pose," "Right hand up,"* etc.,) load existing robot pose to see the robot in action, view raw pose values, or delete existing poses.





Second, the programmer starts coding. While he can write code just as with a standard text-based programming environment, he can also drag and drop a photograph from the pose library to the code editor and the photograph is

shown inline in the code editor. Behind the scene, the IDE inserts textual representation of the photograph (e.g. *Roboko.pose("Right hand up")*) to the dropped position in the source code. Then, it parses the source code to get an abstract syntax tree (AST.) Next, the view is updated according to the AST, showing the corresponding photograph where its textual representation is found. When the parsing is failed or the corresponding pose data is not found, the view is not updated.

During the coding phase, the programmer can benefit from a software library tightly coupled with the IDE. The library provides a set of API that enables easy control of robots. The main functions are shown below:

Pose Robot.getPose() – returns the current pose data.

float Pose.distance(Pose pose) – calculates the distance between the two poses. [0.0-1.0]

boolean Pose.eq (Pose pose, float threshold) – returns whether the two poses can be thought of as identical or not. (returns whether the distance between the two poses is less than the threshold or not.)

boolean Robot.setPose(Pose pose) – set the current pose to the specified data.

Action Robot.action() – start building an action for the robot.

Action Action.pose(Pose pose) – add this pose to the end of this action.

Action Action.wait(int ms) – wait for the specified time at the end of this action.

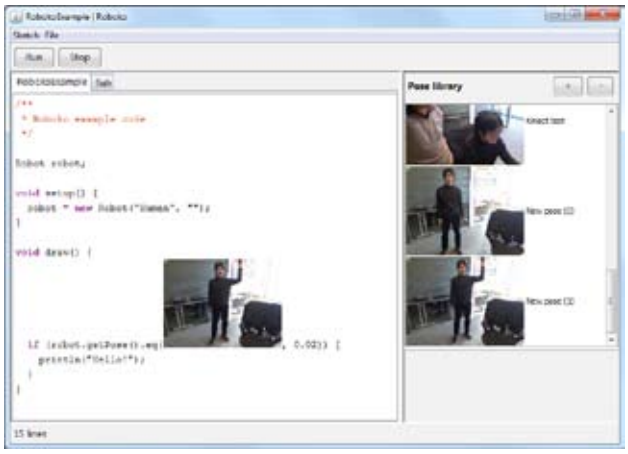
ActionResult Action.play() – play this action.

When the coding is finished, the programmer runs the program by clicking "Run" button. The simple workflow described in this section supports iterative cycle of capturing poses, programming and running applications. It is expected to enable prototyping process not only for professional programmers but also for novice programmers without prior knowledge of robotics.

3. Innovative Applications

While the initial goal of this project focused on programming robots, developing the prototype of the IDE made us aware that our method is also applicable to Kinect programming. We extended the prototype IDE to support Kinect-based skeletal tracking and allow the programmer to easily make pose-based applications (Figure 3.)

Given that the behavior of robot and Kinect-based applications is easy to understand because of the physical embodiment, our IDE might be a good starting point for programming learners. Robot and Kinect are both very hot topic among interaction designers, and thus, our IDE can help their casual development as well.



4. Academic Achievement

We plan to submit this work to CHI'13: ACM SIGCHI 31th Conference on Human Factors in Computing Systems, a top-tier conference in the Human-Computer Interaction research community. [CHI13]

5. Achievement in Talent Fostering

The principal investigator of the project, Jun Kato, is a Ph.D student. He attended an internship at Microsoft Research Asia for 3 months.

6. Collaboration with Microsoft Research

During his internship, Jun collaborated with Dr. Xiang Cao to investigate potential of this research project. He developed Kinect extension introduced at 3. Innovative Applications. Additionally, he started another project and submitted the result to UIST'12: ACM 25th Symposium on User Interface Software and Technology.

7. Project Development

The project is still ongoing and there are much work to be done including user study and further investigation.

Promising direction includes the extension of the IDE to help broader activity related to the hardware development. Current contribution of the project is about the IDE whose text editor is capable of showing photographs captured by a camera, each of which represents a pose of a robot or a person. It frees the programmer from struggling with raw numerical parameters. It is a kind of "what you see is what you get" interface and is expected to be used by novice programmers. Thereupon, using photographs benefits not only programming robot applications but also remembering how the robot is constructed. For instance, there is an example application with a LEGO robot in which the robot waves its hand. Though the LEGO robot can be constructed in many ways with LEGO bricks, photos bound to raw pose data tell the others how the robot is constructed.

During the development of a robot application, the programmer often changes form factors of the robot. It is a kind of iterative cycle of building a robot (hardware) and writing its controlling code (software). In such development process, hardware and software evolve together. Though, while there is much support for tracking changes of code such as Subversion and Git, there's no apparent support for tracking changes of hardware. When the IDE takes snapshot photos of the hardware (and their corresponding pose data) during the development process, it can help the iterative cycle more effectively compared to the existing development environment.

In addition, when the application development is finished, the IDE can not only build the executable binary but also output photo-based step-by-step guidance that tells how the robot can be constructed. The binary and the guidance can be distributed together so that other people can run the application at their home.

8. Publications

Paper publication

1) Roboko: Integrating Visual Representations into a Text-based Programming Environment, to be submitted to CHI'13: ACM SIGCHI 31th Conference on Human Factors in Computing Systems