Picode:
Inline Photos Representing Posture Data in Source Code

Jun Kato **Daisuke Sakamoto** **Takeo Igarashi**
The University of Tokyo, Tokyo, Japan – {jun.kato | d.sakamoto | takeo}@acm.org

ABSTRACT
Current programming environments use textual or symbolic representations. While these representations are appropriate for describing logical processes, they are not appropriate for representing raw values such as human and robot posture data, which are necessary for handling gesture input and controlling robots. To address this issue, we propose *Picode*, a text-based development environment augmented with inline visual representations: photos of human and robots. With *Picode*, the user first takes a photo to bind it to posture data. She then drag-and-drops the photo into the code editor, where it is displayed as an inline image. A preliminary user study revealed positive effects of taking photos on the programming experience.

Author Keywords
Development Environment; Inline Photo; Posture Data.

ACM Classification Keywords
H.5.2. Information interfaces and presentation (e.g., HCI): User Interfaces – GUI; D.2.6. Software Engineering: Programming Environments – Integrated environments.

INTRODUCTION
A programming language is an interface for the programmer to input procedures into a computer. As with other user interfaces, there have been many attempts to improve its usability. Such attempts include visual programming languages to visualize the control flow of the program, structured editors to prevent syntax errors, and enhancement to code completion that visualizes possible inputs [8]. However, programming languages usually consist of textual or symbolic representations. While these representations are appropriate for precisely describing logical processes, they are not appropriate for representing the posture of a human or a robot. In such a case, the programmer has to list raw numeric values or to maintain a reference to the datasets stored in a file or a database.

To address this issue, Ko and Myers presented a framework called "Barista" for implementing code editors which are capable of showing text and visual representations [5]. This framework enhances comments for an image processing method by including an image that shows a concrete example of what the method does. Yeh et al. presented a development environment named "Sikuli," with which the programmer can take a screenshot of a GUI element and paste the image into a text editor [12]. In Sikuli, the image serves as an argument of the API functions. Our goal was to apply a similar idea to facilitate the programming of applications that handle human and robot postures.

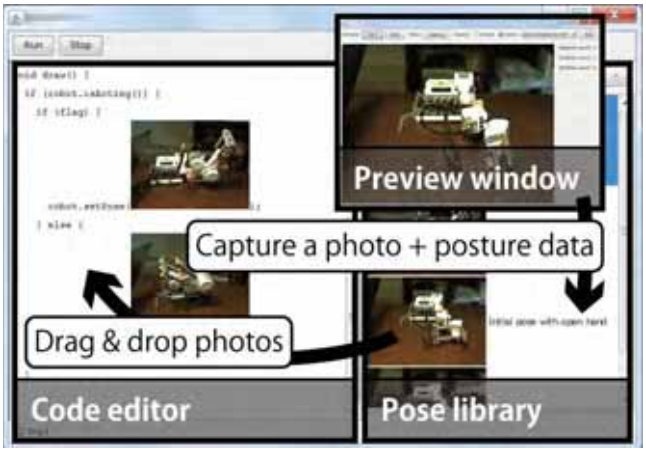


Figure 1. Overview of *Picode*

We propose a development environment named *Picode* that uses photographs of human and robots to represent their posture data in a text editor (Figure 1). It helps the development process of applications for handling gesture input and/or controlling robots. The programmer is first asked to take a photo of a human or a robot to bind it to the posture data. She then drag-and-drops the photo into the code editor, where it is shown as an inline image. Our environment provides a built-in API which methods take photos as arguments. It allows the user to easily understand when the photo was taken and what the code is meant to do.

RELATED WORK
After the Microsoft Kinect and its Software Development Kit (SDK) hit the market, many interactive applications have been developed that handle human posture. At the same time, some toolkits and libraries have been proposed that support the development of such applications. They can typically recognize preset poses and gestures. When the programmer wants to recognize her own poses and gestures, however, she has to record the examples outside the development environment. On the other hand, our development environment is designed to support the entire prototyping process of application development. It fully integrates the recording phase, and the programmer can

CHI 2013, April 27–May 2, 2013, Paris, France.

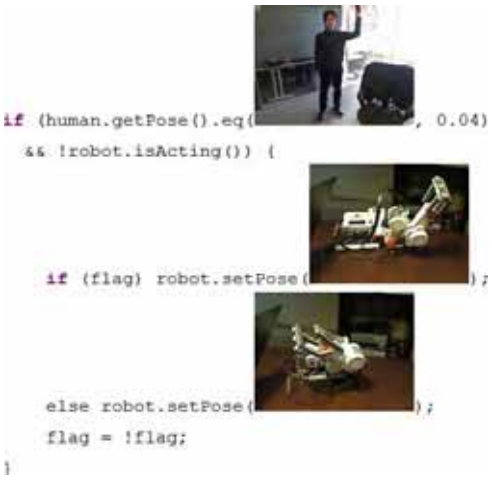


Figure 2. Example code that makes robot swing its hand when user raises her hand

follow the workflow without distraction. Attempts to support a general workflow of domain-specific applications have already been made for many domains including physical computing [3], machine learning [9] and interactive camera-based programs [4].

There is a long history of developing robot applications that deal with robot posture. Typical approaches include Programming by Example (PbE) [1], timeline-based editors to help designers defining transitions from one posture to another [7], and general development environments for textual or visual programming languages [6]. Most of the PbE systems focus on reproducing observed human actions, and the editors focus on creating and editing actions. They both tend to have limited support for handling user input. Conversely, general development environments are more flexible in terms of input handling, but do not display posture data in an informative way. Our objective is to design a hybrid environment, by taking advantages of these approaches.

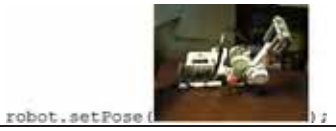
PROGRAMMING WORKFLOW

Our prototype implementation consists of three main components (Figure 1): a code editor, the pose library, and a preview window. First, the user takes a photo of a human or a robot in the preview window. At the same time, posture data are captured and the dataset is stored in the pose library. Next, she drag-and-drops the photo from the pose library into the code editor, where the photo is displayed inline, as shown in Figure 2. Then, she can run the application and distribute the source code bundled with the referenced datasets so that others can run the same application within our development environment.

Taking Photos

To start taking photos, the user clicks the "+" button in the pose library interface and opens the preview window in which the photo preview and posture status are displayed in real time. She can choose the input source of the posture data from Kinect (human) or Mindstorms NXT [6] (robots)

Compare *Pose* with specified error allowance [0-1]



Set current *Pose*

Play series of pose changes by *Action* definition

Table 1. Usage examples of photo-based API

devices. While only one Kinect device can be connected at a time and is automatically detected, one or more Mindstorms NXT devices can be used by entering their Bluetooth addresses. Photos are usually taken from the RGB stream of a Kinect device, but a web camera can be used as an alternative source.

While the preview window is displayed, clicking the "Capture" button triggers the system to take a photo and capture the corresponding posture data. Each captured dataset is automatically named, e.g., "New pose (1)," and stored in the pose library. It can be manually renamed but must be unique. Saying the word "capture" works when the user wants to capture a human posture and cannot click the button because standing in front of the Kinect device. When capturing a robot posture, a torque is applied to each servo motor on a joint to fix its angle. When the user tries to change its angle, however, the torque is set off so that she can move the joint freely. Therefore, the user can set the robot posture by changing joint angles individually. Additionally, she can load an existing posture by right-clicking its photo in the library. This allows the user to easily create a new posture from the existing ones. These interactions for capturing a robot's posture are inspired by the actuated physical puppet [13].

Coding with Photos

The programmer can write code in a programming language that is an extension of Processing [10], with a built-in photo-based API whose methods take photos as arguments. She can drag-and-drop photos from the pose library to the code editor, directly into argument bodies of the methods. Usage examples of currently supported API are shown in Table 1. A human and robot are represented by *Human* and *Robot* classes, whose instance handles communication with the hardware devices. Note that the *Human* instance is capable of sensing but not controlling posture while the *Robot* instance is capable of both.

Running Program

The programmer can compile and run the program by clicking the "Run" button in the main window. After iterative cycles of development, a ZIP archive consisting of source code, referenced photos, and posture data can be made so that others can run the same application.

IMPLEMENTATION

Picode is built on top of Processing core components including its compiler and libraries. The main difference is in the user interface. Therefore, the programmer can benefit from the simple language specification and extensibility provided by many Java-based libraries. Beside the user interface, we modified the compilation process to link every program to our library. We also modified the execution process so that the development environment disconnects from the Kinect device and robots when the program starts, and reconnects to them when it shuts down.

Human postures and the corresponding images are retrieved using a standalone GUI-less program implemented with Kinect for Windows SDK, which is automatically executed when needed. It communicates with the development environment and all programs that run on the environment through a TCP/IP connection. Robot postures are retrieved by reading values of a motor encoder or set by transmitting Bluetooth commands that are officially supported by the Mindstorms NXT firmware.

Code Editor Supporting Inline Photos

The code editor is implemented in the Model-View architecture, where the model is the source code in string format and the view is its GUI representation. Each photo has its string representation, which is a call to the specific photo-based API *Pose.load(key)* where *key* is a unique name of the corresponding posture data. When the photo is dropped to the code editor, the string is inserted into the source code. Every change in the source code triggers the language parser in order to build an abstract syntax tree. Then, the view is updated for syntax highlighting and every call to the photo-based API is replaced with photos.

API with Photo Arguments

Each posture dataset represented by a photo is instantiated as a *Pose* class instance. A *Pose* class is currently extended using *KinectHumanPose* and *MindstormsNXTPose* classes to support platform-dependent implementation and can be further extended to support more types of robots, such as humanoids, or more ways of detecting poses such as with a motion capture system. The posture data and the photo are saved as a text file and a JPEG file with its unique name (e.g. *Hand up.txt* and *Hand up.jpg*) in the same directory. The text file starts with its corresponding *Pose* class name followed by raw numerical values.

The equality test between *Pose* instances always returns false if their types are different. When their types are the same, the system calculates the Euclid distance between the vectors consisting of the absolute difference between joint angles (e.g. absolute difference in elbow angle, knee angle, …) and normalizes it between 0 and 1. The equality test returns true if the distance is less than the specified threshold, otherwise it returns false.

PRELIMINARY USER STUDY AND DISCUSSION

We asked two test users to try our development environment together for about three hours. The goal was to verify two hypotheses on the benefit of embedding photos in the source code. The first hypothesis was that photos contain rich contextual information other than mere posture information, which helps the programmer recall the situation. The other was that the inline photos can involve a non-programmer in the software development process since they can be basically taken and understood by anybody. While one test user knew Processing and was familiar with basic programming concepts, the other did not know about programming except for basic HTML coding. We had them work together since we expected our environment to establish a new relationship between programmers and non-programmers (users). First, we thoroughly explained the workflow of our programming environment with the example code for an hour. Then, we asked them to make their own program for the remaining two hours.

After two hours of free use, the participants could write a program that uses gesture input to control robot posture. The robot basically tried to mimic the user input, e.g., when the user waved her hand, the robot waved its hand back. By putting the robot in front of the keyboard, the participants also had it operate the PC with its mechanical hand, which reminded us of mechanical hijacking [2].

Contextual Information in Photos

When the participants were asked to read existing code, they seemed to benefit from contextual information in photos, which was missing in the numerical posture data. The programmer commented that he might also benefit from the information when he reads the code he had written a long time ago since the photo can remind him of the situation. According to this observation, there were two types of contextual information. The first type tells the user about what the subject (human or robot) in the photos was doing. For example, photos would make it easy to distinguish when a user is drinking a glass of juice from when she is raising her hand to greet, while raw posture data will be the same (or very similar). A robot hand grasping a small ball and a large cube falls within the same issue. Additionally, each photo of the robot helps the users remember the proper hardware configuration. Prototyping robot applications often requires many iterations, and the photos taken during the development process might work as revision history for the hardware setup. The second type tells the user about the surrounding context for which the program was designed. For instance, the optimal parameters

for a mobile robot that runs on the floor differ according to the material, such as carpets or laminated flooring.

Source Code as Communication Medium

The meaning of the inline photos could be understood by both the programmer and the non-programmer, and the photos worked as a communication medium between them. The non-programmer said that she felt involved in the application development process and was never bored. She stated two reasons for this feeling. First, she could take part in the development process by taking photos. Simple algorithms that handle posture data often require parameter tuning depending on the environment in which the code runs. In our environment, this can be done by replacing the existing photo with a new one. Through the replacement, she started to take ownership of the source code. With the inline photos, the source code became not only for the programmer but also for the non-programmer. Second, she could guess what the code was doing by recognizing the inline photos. For non-programmers, text code sometimes looks like a series of non-sense words. In *Picode*, however, they can understand the meaning of the code in relation to its nearby photos. When she asked a question about the code to the programmer, the programmer often started the explanation by pointing to the related photo. She also mentioned that the photos were easy to see in the plain text code, which made it easy to locate particular lines of the code. The idea of making meaning of code transparent (more understandable) was also discussed in Victor's recent essay about learnable programming [11]. Inline photos can be a good starting point for learning programming.

FUTURE WORK AND CONCLUSION

We foresee three enhancements that can make our development environment more effective: support for machine learning, comparison between partial posture data, and recording videos instead of taking photos. First, the current API only supports comparison between one posture dataset with another, which makes it difficult to recognize more general postures. For example, when the programmer wants to recognize the human posture of raising the right hand regardless of the height of the hand, she must write several "if" statements. Support for machine learning might solve this issue, treating multiple posture datasets as correct examples and others as false examples. Second, the current API cannot compare partial data, which makes it difficult to recognize the posture of the right hand and ignore the other body parts. With Kinect, *Picode* might allow the programmer to mask certain areas of the body on the photo to ignore the corresponding joints. Third, recording videos instead of taking photos might allow interesting programming experiences, by combining *Picode* code-based approach with the flow paradigm of DejaVu [4]. Videos can be used for learning human gestures or for replaying robot actions. The programmer might be able to change the replaying speed to make robot actions faster or slower.

We introduced *Picode*, a development environment that integrates photos into a code editor. It supports the programming workflow with the posture data: recording examples by taking photos, coding, and running the program. Photos were found to be interesting media that enhance the programming experience. *Picode* is open-source and available at http://junkato.jp/picode/.

ACKNOWLEDGEMENTS This work was supported in part by Microsoft Research 7th collaborative research program and JSPS KAKENHI Grant Number 23-9292, 24700112.

REFERENCES

1. Billard, A., Calinon, S., Dillmann, R. and Schaal, S. Robot programming by demonstration. In *Handbook of Robotics*, Springer (2008), 1371-1394.
2. Davidoff, S., Villar, N., Taylor, A.S. and Izadi, S. Mechanical hijacking: how robots can accelerate UbiComp deployments. In *Proc. UbiComp 2011*, 267-270.
3. Hartmann, B., Klemmer, S.R., Bernstein, M., Abdulla, L., Burr, B., Mosher., A.R. and Gee, J. Reflective physical prototyping through integrated design, test, and analysis. In *Proc. UIST 2006*, 299-308.
4. Kato, J., McDirmid, S. and Cao, X. DejaVu: Integrated support for developing interactive camera-based programs. In *Proc. UIST 2012*, 189-196.
5. Ko, A.J. and Myers, B.A. Barista: An implementation framework for enabling new tools, interaction techniques and views in code editors. In *Proc. CHI 2006*, 387-396.
6. LEGO Mindstorms NXT. http://mindstorms.lego.com/
7. Nakaoka, S., Kajita, S. and Yokoi, K. Intuitive and flexible user interface for creating whole body motions of biped humanoid robots. In *Proc. IROS 2010*, 1675-1682.
8. Omar, C., Yoon, Y., LaToza, T.D. and Myers, B.A. Active code completion. In *Proc. ICSE 2012*, 859-869.
9. Patel, K., Bancroft, N., Drucker, S.M., Fogarty, J., Ko, A.J. and Landay, J. Gestalt: integrated support for implementation and analysis in machine learning. In *Proc. UIST 2010*, 37-46.
10. Processing. http://processing.org/
11. Victor, B. Learnable Programming. http://worrydream.com/LearnableProgramming/
12. Yeh, T., Chang, T.H. and Miller, R.C. Sikuli: using GUI screenshots for search and automation. In *Proc. UIST 2009*, 183-192.
13. Yoshizaki, W., Sugiura, Y., Chiou, A.C., Hashimoto, S., Inami, M., Igarashi, T., Akazawa, Y., Kawachi, K., Kagami, S. and Mochimaru, M. An actuated physical puppet as an input device for controlling a digital manikin. In *Proc. CHI 2011*, 637-646.