Sidebar – Programming Commercial Robots

José María Cañas[1], Vicente Matellán[2], Bruce MacDonald[3], and Geoffrey Biggs[4]

[1] Universidad Rey Juan Carlos, Madrid, Spain jmplaza@gsyc.escet.urjc.es
[2] Universidad Rey Juan Carlos, Madrid, Spain vicente.matellan@urjc.es
[3] University of Auckland, New Zealand b.macdonald@auckland.ac.nz
[4] University of Auckland, New Zealand g.biggs@auckland.ac.nz

Lozano–Pérez [LP1986] divided robot programming into methods for guiding, robot–level programming, and task–level programming. A more useful distinction for modern methods is between manual programming and automatic programming, based on the actual method used for programming as this is the crucial distinction for users and programmers.

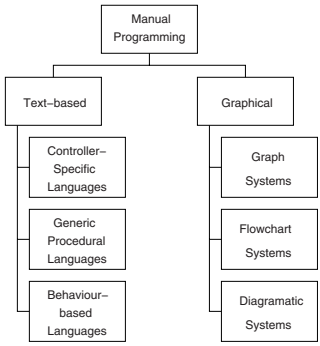

Fig. 1. Categories of manual programming systems. A manual system may use a text-based or graphical interface for entering the program.

Manual systems require the user/programmer to directly enter the desired behaviour of the robot, usually using a graphical or text-based programming language, as shown in Fig. 1. Text-based systems are either controller-specific languages, generic procedural languages, or behavioural languages, which typically differ by the flexibility and method of expression of the system. Graphical languages [BKS2002, BI2001] use a graph, flow-chart or diagram based graphical interface to programming, sacrificing some flexibility and expressiveness for ease of use.

D. Brugali (Ed.): Software Eng. for Experimental Robotics. STAR 30, pp. 125–132, 2007.

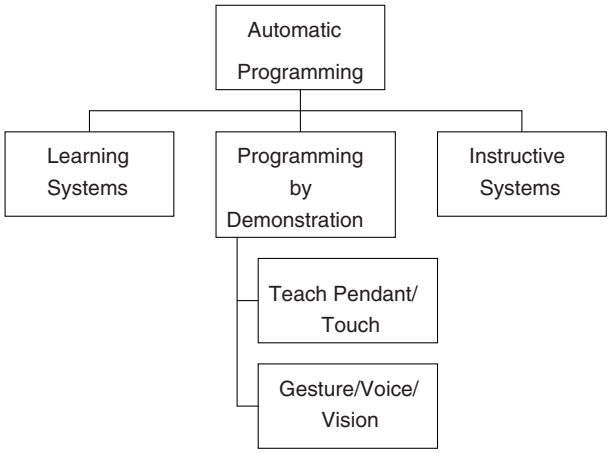

Fig. 2. Categories of automatic programming systems. Learning systems, programming by demonstration (PbD) and instructive systems are all methods of teaching robots to perform tasks.

The user/programmer has little or no direct control over the robot code in an automatic programming system, which may acquire the program by learning, programming by demonstration (PbD), or by instruction, as indicated in Fig. 2. Often automatic systems are used "online," with a running robot, although a simulation can also be used.

In this sidebar we will focus on the characteristics of commercial programming environments. Simple robots can be programmed directly using their own operating systems. More sophisticated robots include SDKs to simplify the programming of their robots. Mobile robots programming environments vs. industrial manipulators are also presented.

1 Industrial Manipulators

Programming systems for industrial manipulators include both manual and automatic methods of programming. Initially manual programming tools were common, in the form of text-based controller-specific languages. Controller–specific languages are designed for a single robot system, for example the system provided by KUKA, shown in Fig. 3.

Coupled with touch screens, graphical languages can enable rapid configuration of industrial robots.

PbD was developed for industrial robot manipulators, using a teach pendant or similar method to move the manipulator to each position in a task, where the robot's joint positions are recorded for later playback. Recent work in PbD has focussed on creating more flexible robot programs by segmenting demonstrations to identify key actions [EZRD2002, CM1998, CM2000,

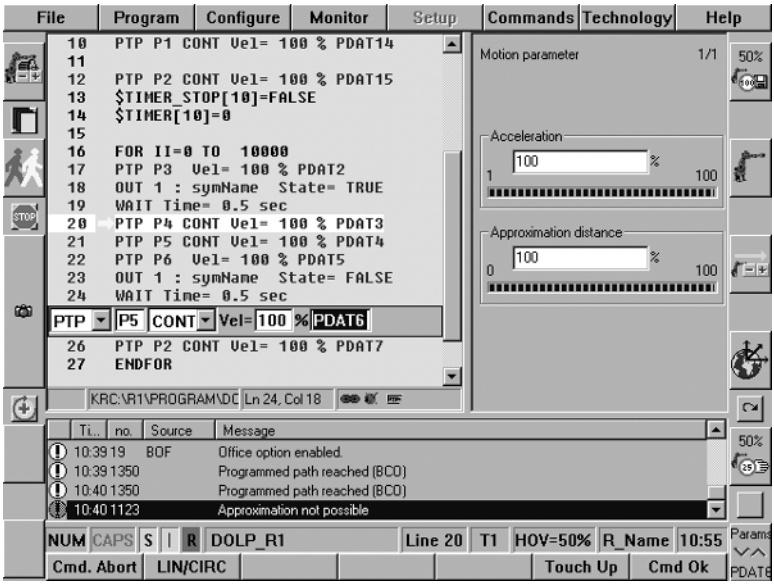

Fig. 3. The KUKA programming environment. [From [KU2005]]

CZ2001, OKKI2002], and on using more natural interaction methods such as voice and touch to perform the demonstrations [GSAH2001, YKY2002, TOKI2002]. Other work includes virtual environments for PbD [OSK2002], finger sensors to detect fine manipulations [ZRDZ2002], and graphical display of demonstration results [FHD1998].

2 Mobile Robots

Mobile robot programming has evolved significantly in recent years, and two approaches are currently found, both manual programming methods. On one hand, application programs for simple robots obtain readings from sensors and send commands to actuators by directly calling functions from the drivers provided by the seller. On the other hand, we have identified many common features across commercial SDKs.

First, they offer a simple and more abstract access to sensors and actuators than the operating systems of simple robots. For example, in a Pioneer with a laser rangefinder, the applications can obtain readings using ARIA or directly through a serial port. Using ARIA, one need only invoke a method and ARIA will take charge of refreshing the variables. Using the operating system directly, the application must request and periodically read the data from the laser through the serial port, and must identify the protocol of the device to compose and analyze the low level messages correctly. The abstract access is also offered for actuators.

Second, the software architecture of the SDK sets the way the application code obtains sensor data, commands the motors, or uses a developed functionality. There are many software options: calling to library functions, reading variables, invoking object methods, sending messages via the network to servers, etc.. Depending on the programming model the robot application can be considered an object collection, a set of modules talking through the network, an iterative process calling to functions, etc.

Third, usually the SDK includes simple libraries and common use functionality, such as robust techniques for perception or control, localization, safe local navigation, global navigation, social abilities, map construction, etc. The robot manufacturers sell them separately or include them as additional value with their own SDK. For example, ERSP includes three packages in the basic architecture: one for interaction, one for navigation and another for vision.

There are several advantages of using the SDKs. First, they favor the portability of applications between different robots. Second, they promote code reuse, shortening the development time and reducing the programming effort needed to code the application as long as the programmer can build the program by reusing the common functionality, keeping herself focused in the specific aspects of her application. And third, the software architecture offers a way to organize code, allowing the handling of code complexity when the robot functionality increases.

The next sections present some case studies for different mobile robot environments. Most of them are based on *libre*[5] software because it lets us freely explore the underlying technologies.

2.1 LEGO RCX and BrickOS

The RCX[6] in Fig. 5 is sold as a creative and educational toy. It has a central processor, the RCX brick, and a set of LEGO pieces that are assembled to build the body. There are many ways to program it. LEGO offers a graphical programming environment oriented to children, named RCX-code (Fig. 4). Another possibility is NQC [Bau00], a variation of C which includes instructions to access to the sensors and actuators.

The open-source operating system BrickOS[7], developed by Markus L. Noga [Nie00], allows programming the brick in C. And the LeJOS operating system allows the creation of Java applications. These operating systems, including the original LEGO, offer multitasking. Since its sensors and actuators are simple, an SDK is not necessary and applications can be developed without difficulty, programming directly over the API of the operating system.

[5] We use the Spanish term "libre" to avoid the common misunderstanding between free as in "free beer" and free as in "free speech."
[6] http://www.legomindstorms.com
[7] http://brickos.sourceforge.net/

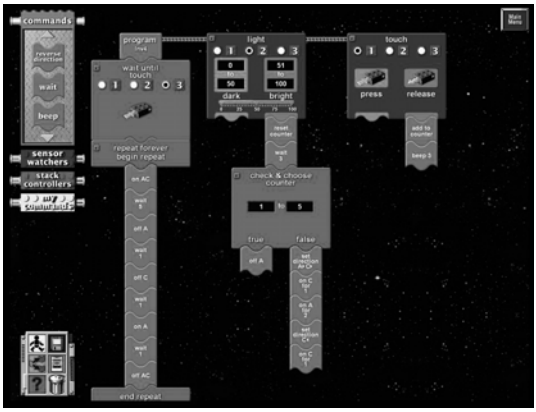

Fig. 4. The Lego Mindstorms graphical programming environment, used to create simple programs for Lego robots.

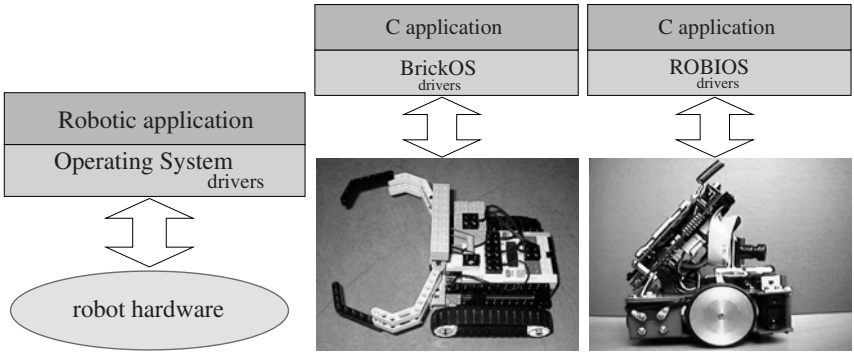

Fig. 5. LEGO (center) and EyeBot (right) are programmed over their Op. Systems

2.2 EyeBot and ROBIOS

The EyeBot [8] [Bra03] is a small robot. Its operating system, ROBIOS, is a meaningful example of an ad-hoc operating system. It allows programs to be loaded through a serial port and executed by pushing buttons. The ROBIOS API includes functions to read the infrared sensors, capture images from the camera and move the motors at a certain speed. It also includes two functions to send and receive bytes through the radio link to other EyeBots or a PC. ROBIOS includes primitives to monitor whether a button is being pushed, and to display images and text on the screen. Concerning multitasking, ROBIOS has primitives to create, pause or kill threads. It offers two ways to share the processor time between threads: with and without preemption. If also offers

[8] http://robotics.ee.uwa.edu.au/eyebot/

locks for the coordination of concurrent execution of these threads, and access to shared variables.

2.3 Aibo and OPEN-R

The Aibo robot[9] in Fig. 6) is sold as a pet, with a program that governs its movements to exhibit dog behaviors (follow a ball, look for a bone, dance, etc.) and learn. Since the summer of 2002 it has been possible to program it, and so the AIBO is useful for research. The operating system in charge of controlling the hardware devices is Aperios, a real-time operating system based on objects. On top of this, Sony provides the OPEN-R SDK [Cor03], which includes many specific C++ objects to access the Aibo hardware. There are objects for basic access to camera images, joint positions, management of the TCP/IP stack and management of the microphone and speaker. Additionally, OPEN-R allows multitasking and event oriented programming.

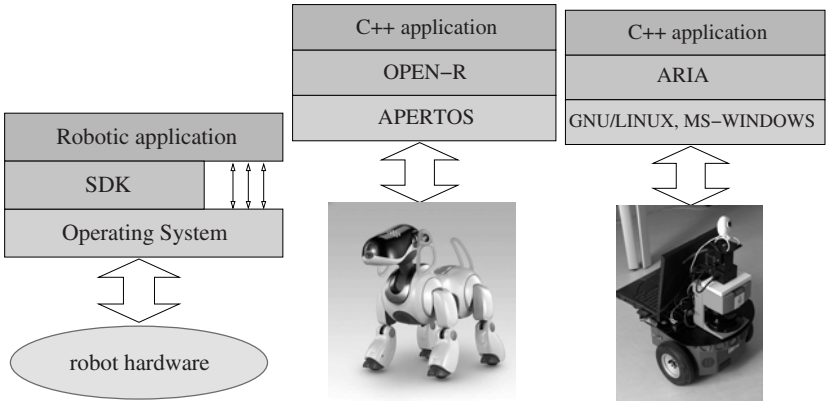

Fig. 6. Two robots programmed using an SDK: Aibo (center) and Pioneer (right)

2.4 Pioneer and ARIA

The Pioneer[10] is a medium size robot, shown in Fig. 6. Its onboard PC is connected to the base microcontroller via a serial port. ARIA (ActivMedia Robotics Interface for Applications) [Rob02] is the manufacturer SDK for the Pioneer robot. ARIA is supported by ActivMedia Robotics, but it is distributed with a GPL license. It offers an object-oriented programming environment, which includes support for multitasking programming and network

[9] http://www.aibo.com/
[10] http://www.activrobots.com/

communication. Applications must be written in C++, and since ARIA runs both on Linux and MS-Windows, the same ARIA application can control robots from either operating system.

For hardware access, ARIA offers a collection of classes, which setup an object based API. The main class *ArRobot* has many relevant methods. There are classes for range sensors and their objects have methods which allow the application to access the data from the proximity sensors. The objects of ARIA are not distributed, ARIA allows the programming of distributed applications using *ArNetworking* to manage remote communications.

Concerning multitasking, the application on ARIA can be programmed as mono-threaded or multi-threaded. In the latter, ARIA offers infrastructure for both user threads and kernel threads, which are a wrapper of the native Linux-pthreads or Win32-threads. For concurrency and synchronization ARIA offers resources such as *ArMutex* and *ArCondition*. It also has basic behaviors such as safe navigation and obstacle avoidance, but it does not include map construction or localization functionalities, which are sold separately. Recently, it has included the open-source simulator Stage (renamed as MobileSim), which has been adapted to work with ARIA. It is a clear example of code reuse in robot applications, involving a private company.

References

[Bau00] Dave Baum, *Dave baum's definitive guide to lego mindstorms*, Apress, 2000.

[Bra03] Thomas Braunl, *Embedded robotics*, Springer Verlag, 2003.

[Cor03] Sony Corporation, *Open-r sdk, programmers guide*, Technical Report 20030201-E-003, 2003.

[Nie00] Stig Nielsson, *Introduction to the legos kernel*, Technical Report, 2000.

[Rob02] ActivMedia Robotics, *Aria reference manual*, Technical Report (1.1.10), 2002.

[USEK02] Hans Utz, Stefan Sablatnog, Stefan Enderle, and Gerhard Kraetzschmar, *Miro – middleware for mobile robot applications*, IEEE Transactions on Robotics and Automation, Special Issue on Object-Oriented Distributed Control Architectures **18** (2002), no. 4, 493–497.

[LP1986] Tomás Lozano-Pérez. Robot programming. Technical Report Memo 698, MIT AI, December 1982, revised April 1983 1982. Also published in Proceedings of the IEEE, Vol 71, July 1983, pp.821–841 (Invited), and IEEE Tutorial on Robotics, IEEE Computer Society, 1986, pp.455–475.

[BKS2002] R. Bischoff, A. Kazi, and M. Seyfarth. The MORPHA style guide for icon-based programming. In *Robot and Human Interactive Communication, 2002. Proceedings. 11th IEEE International Workshop on*, pages 482–487, 2002.

[BI2001] A. Bredenfeld and G. Indiveri. Robot behavior engineering using DD-Designer. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 1, pages 205–210, 2001.

[KU2005] KUKA Automatisering + Robots N.V. http://www.kuka.be/, June 2005.

[EZRD2002] M. Ehrenmann, R. Zollner, O. Rogalla, and R. Dillmann. Programming service tasks in household environments by human demonstration. In *Robot and Human Interactive Communication, 2002. Proceedings. 11th IEEE International Workshop on*, pages 460–467, 2002.

[CM1998] J. Chen and B. McCarragher. Robot programming by demonstration-selecting optimal event paths. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 1, pages 518–523, May 1998.

[CM2000] J.R. Chen and B.J. McCarragher. Programming by demonstration - constructing task level plans in hybrid dynamic framework. In *Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA '00)*, volume 2, pages 1402–1407, apr 2000.

[CZ2001] J.R. Chen and A. Zelinsky. Programming by demonstration: removing sub-optimal actions in a partially known configuration space. In *Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA '01)*, volume 4, pages 4096–4103, May 2001.

[OKKI2002] K. Ogawara, J. Takamatsu, H. Kimura, and K. Ikeuchi. Generation of a task model by integrating multiple observations of human demonstrations. In *Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA '02)*, volume 2, pages 1545–1550, May 2002.

[GSAH2001] G. Grunwald, G. Schreiber, A. Albu-Schaffer, and G. Hirzinger. Touch: The direct type of human interaction with a redundant service robot. In *Robot and Human Interactive Communication, 2001. Proceedings. 10th IEEE International Workshop on*, pages 347–352, 2001.

[YKY2002] Y. Yokokohji, Y. Kitaoka, and T. Yoshikawa. Motion capture from demonstrator's viewpoint and its application to robot teaching. In *Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA '02)*, volume 2, pages 1551–1558, May 2002.

[TOKI2002] J. Takamatsu, K. Ogawara, H. Kimura, and K. Ikeuchi. Correcting observation errors for assembly task recognition. In *Intelligent Robots and System, 2002. IEEE/RSJ International Conference on*, volume 1, pages 232–237, 2002.

[OSK2002] H. Onda, T. Suehiro, and K. Kitagaki. Teaching by demonstration of assembly motion in vr - non-deterministic search-type motion in the teaching stage. In *Intelligent Robots and System, 2002. IEEE/RSJ International Conference on*, volume 3, pages 3066–3072, 2002.

[ZRDZ2002] R. Zollner, O. Rogalla, R. Dillmann, and M. Zollner. Understanding users intention: programming fine manipulation tasks by demonstration. In *Intelligent Robots and System, 2002. IEEE/RSJ International Conference on*, volume 2, pages 1114–1119, 2002.

[FHD1998] H. Friedrich, J. Holle, and R. Dillmann. Interactive generation of flexible