# A Survey of Robot Programming Systems

**Geoffrey Biggs** and **Bruce MacDonald**
Department of Electrical & Electronic Engineering, University of Auckland

Email: g.biggs at ec.auckland.ac.nz, b.macdonald at auckland.ac.nz

## Abstract

Robots have become significantly more powerful and intelligent over the last decade, and are moving in to more service oriented roles. As a result robots will more often be used by people with minimal technical skills and so there is a need for easier to use and more flexible programming systems. This paper reviews the current state of the art in robot programming systems. A distinction is made between manual and automatic programming systems. Manual systems require the user/programmer to create the robot program directly, by hand, while automatic systems generate a robot program as a result of interaction between the robot and the human; there are a variety of methods including learning, programming by demonstration and instructive systems.

## 1 Introduction

Robots are complex machines and significant technical knowledge and skill are needed to control them. While simpler robots exist, for example the Roomba vacuuming robot from iRobot [2003], in these cases the robots are specifically designed for a single application, and the control method reflects this simplicity. The Roomba robot's control panel allows a user to select different room sizes and to start the vacuuming process with a single button push.

However, most robots do not have simple interfaces and are not targeted at a single, simple function such as vacuuming floors. Most robots have complex interfaces, usually involving a text–based programming language with few high–level abstractions. While the average user will not want to program their robot at a low level, a system is needed that provides the required level of user control over the robot's tasks.

Robots are becoming more powerful, with more sensors, more intelligence, and cheaper components. As a result robots are moving out of controlled industrial environments and into uncontrolled service environments such as homes, hospitals, and workplaces where they perform tasks ranging from delivery services to entertainment. It is this increase in the exposure of robots to unskilled people that requires robots to become easier to program and manage.

### 1.1 Reviewing robot programming

This paper reviews the current state of the art in robot programming systems, in the the related area of robot software architectures, and related trends. We do not aim to enumerate all existing robot programming systems. A review of robot programming systems was conducted in 1983 by Tomás Lozano–Pérez [1982]. At that time, robots were only common in industrial environments, the range of programming methods was very limited, and the review examined only industrial robot programming systems. A new review is necessary to determine what has been achieved in the intervening time, and what the next steps should be to provide convenient control for the general population as robots become ubiquitous in our lives.

Lozano–Pérez divided programming systems into three categories: guiding systems, robot–level programming systems and task–level programming systems. For guiding systems the robot was manually moved to each desired position and the joint positions recorded. For robot–level systems a programming language was provided with the robot. Finally, task–level systems specified the goals to be achieved (for example, the positions of objects).

By contrast, this paper divides the field of robot programming into automatic programming, manual programming and software architectures, as shown in Fig. 1. The first two distinguish programming according to the actual method used, which is the crucial distinction for users and programmers. In automatic programming systems the user/programmer has little or no direct control over the robot code. These include learning systems, Programming by Demonstration and Instructive

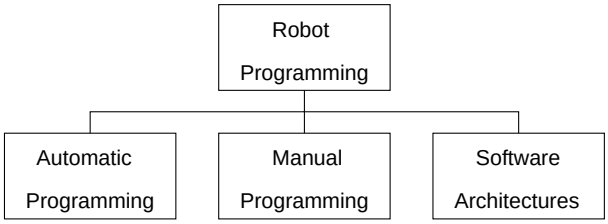


Figure 1: A robot programming system may use automatic or manual programming. Software architectures also play an important role.

Systems. Manual systems require the user/programmer to directly enter the desired behaviour of the robot, usually using a graphical or text–based programming language. Software architectures are important to all programming systems, as they provide the underlying support, such as communication, as well as access to the robots themselves.

Section 2 will concentrate on manual programming systems, while Section 3 will concentrate on automatic programming systems. Section 4 gives conclusions on the trends in robot programming systems. A review of software architectures is beyond the scope of this paper.

## 2 Manual Programming Systems

Users of a manual programming systems must create the robot program by hand, which is typically performed without the robot. The finished program is loaded into the robot afterwards. These are often off-line programming systems, where a robot is not present while programming. It is conceivable for manual programming to control a robot online, using for example an interpreted language, where there are no safety concerns (eg the Lego Mindstorm [2003]).

As shown in Fig. 2, manual programming systems can be divided into text–based and graphical systems (also known as icon–based systems). Graphical programming is not considered automatic programming because the user must create the robot program code by hand before running it on the robotic system. There is a direct correspondence between the graphical icons and the program statements.

### 2.1 Text–based Systems

A text–based system uses a traditional programming language approach and is one of the most common methods, particularly in industrial environments where it is often used in conjunction with Programming by Demonstration (see Section 3.2). Text–based systems can be distinguished by the type of language used, in terms of the type of programming performed by the user. This division can be seen in Fig. 2 and is explained in the remainder of this section.

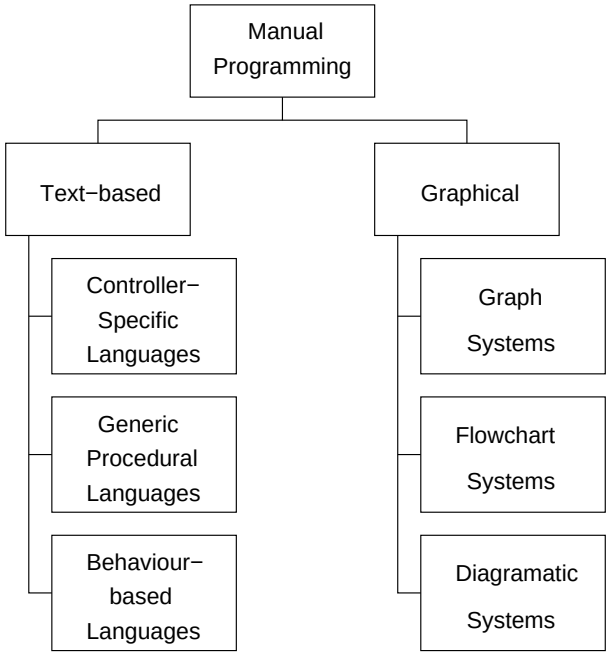


Figure 2: Categories of manual programming systems. A manual system may use a text–based or graphical interface for entering the program.

#### Controller-Specific Languages

Controller–specific languages were the original method of controlling industrial robots, and are still the most common method today. Every robot controller has some form of machine language, and there is usually a programming language to go with it that can be used to create programs for that robot. These programming languages are usually very simple, with a BASIC–like syntax and simple commands for controlling the robot and program flow. A good example is the language provided by KUKA [2003] for its industrial robots, shown in Fig. 3. Programs written in this language can be run on a suitable KUKA robot or tested in the simulation system provided by KUKA.

Despite having existed for as long as industrial robots have been in use, controller–specific languages have seen only minor advances. In one case, Freund and Luedemann-Ravit [2002] have created a system that allows industrial robot programs to be generalised around some aspect of a task, with a customised version of the robot program being generated as necessary before being downloaded into a robot controller. The system uses a "generation plan" to provide the basic program for a task. For example, a task to cut shaped pieces of metal could be customised by the shape of the final result. While such a system can help reduce the time for producing programs for related products, it does not reduce the initial time to develop the robot program.

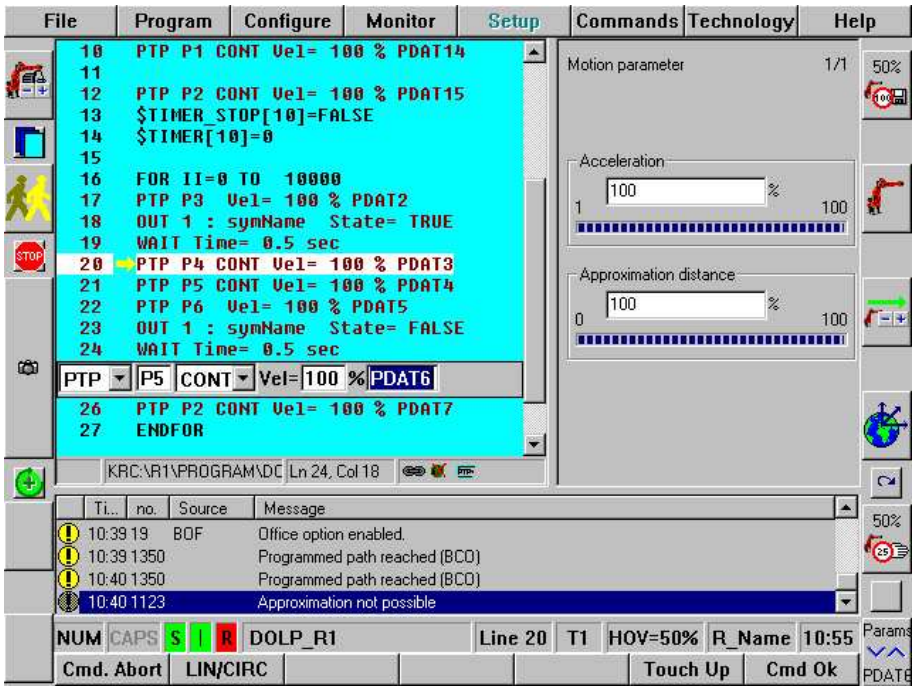


Figure 3: The KUKA programming environment and robot programming language.

Freund *et al.* [2001] have also done some work to ease the use of simulation systems in industrial environments. Because of the abundance of control languages, a simulator system must be able to understand the language of each program it is to simulate. Robot manufacturers often provide a simulation system with the programming language, but this once again increases the training time for staff. To enable a single simulation system to be used for multiple languages, translators are typically used. Freund *et al.* created a translator framework that can significantly reduce the time required to develop these translators. It is now in use on the COSIMIR [2003] simulation system in commercial environments.

Controller–specific languages have some drawbacks. The biggest problem is the lack of a universal standard between languages from different robot manufacturers. If a factory uses robots from many different manufacturers then they will need to either train their programmers for each one, or pay the manufacturer to develop the required programs for them. Either method increases significantly the time and costs for developing new robot programs. Commercial systems have concentrated their advances on overcoming this by providing more advanced programming systems that remove the need for the programmer to actually write the robot code by hand. Instead, the programmer can for example select instructions from a list. These systems are designed to significantly reduce programming time, but are generally application-specific. Examples include systems from KUKA [2003] and ABB [2003].

### Generic Procedural Languages

Generic languages provide an alternative to controller–specific languages for programming robots. "Generic" means a high–level multi–purpose language, for example C++, that has been extended in some way to provide robot–specific functionality. This is particularly common in research environments, where generic languages are extended to meet the needs of the research project. The choice of the base language varies, depending upon what the researchers are trying to achieve (for example, procedural or behavioural programming). A language developed in this way may be aimed at system programming or application level programming.

The most common extension to a multi–purpose language is a robot abstraction, which is a set of classes, methods, or similar constructs that provides access to common robot functions in a simple way. They remove the need to handle low–level functionality such as setting output ports high to turn on motors or translating raw sensor data. For example, an abstraction may provide a method to have the robot move a manipulator to a certain position. It might also provide higher–level abstractions, such as methods to make the robot move to a point using path planning. It is common now for a research robot from a manufacturer to provide such a system with their robots. However, these abstractions suffer from the same fault as controller–specific languages for industrial robots. They are still specific to the robot

they are designed for.

To improve this situation, many researches have developed their own robot abstraction systems. Player/stage is a commonly used robot programming system, that provides drivers for many robots and abstractions for controlling them [Vaughan *et al.*, 2003]. Kanayama and Wu [2000] have developed a "Motion Description Language" extension to Java that provides high–level abstractions for mobile robots. To prevent the abstraction from being limited to one robot architecture they use Java classes to provide common abstractions and programming interfaces. Each robot needs a customised version of the *Vehicle* class, because of the specific robot hardware differences. Other services, such as path planning, are also represented by classes.

Others have implemented similar systems, including Hardin *et al.* [2002], who developed a system primarily used on Lego Mindstorms robots [Lego, 2003]. As well as Java, abstractions have been created for many other generic languages, including C++ [Hopler and Otter, 2001, which also provides real-time extensions], [Loffler *et al.*, 2001] and Python, known as Pyro [2003]. Pyro is a particularly extensive system, providing classes and abstractions for robots and algorithms. Even eXtensible Markup Language (XML) has been used for describing robot motion, in a form that can be transmitted over networks and then played back on a suitable robot body [Kitagishi *et al.*, 2002].

It is interesting to note that a abstractions are commonly implemented using an object–oriented methodology. McKee *et al.* [2001] state that a rigorous object–oriented approach to robotics is important to clearly define robotic entities relationships. They describe the MARS model of object-oriented robots, and define robots comprised of "resources," which are then modelled as "modules." They draw clear parallels between object-oriented concepts such as inheritance, and the modules of a robot. A good example is a tool on the end of an arm. Simply by being on the end of an arm, the tool inherits the ability to move.

Thrun [2000] has developed CES, an extension for C++ that provides probabilistic programming support. The use of probabilistic methods allows robot programs to overcome problems caused by such things as sensor noise. However, writing programs with probabilistic methods is difficult. CES provides a set of C++ templates for probability distributions on variable types. It also provides methods for learning, which can be used in conjunction with standard programming practices to create a system where parts are coded by hand while other parts are trained. The system was tested by creating a mail delivery program for a mobile robot. The program required only 137 lines of code and two hours of training. While the use of this language does require a good knowledge of statistical methods, it shows that such a programming system is possible in a general purpose language. If combined with a suitable method to remove the need for low-level robot control, it could be a powerful system for creating learning robots.

**Behaviour–based Languages**

Behaviour–based languages provide an alternative approach to the procedural languages of the previous sections. They typically specify how the robot should react to different conditions, rather than providing a procedural description. For example, a set of behaviours could be to follow a wall from one point to another. A behavioural system is more likely to be used by a robot developer than the end user. The developer would use it to define functionality that the end user would use to perform tasks.

Functional Reactive Programming (FRP) is a good example of a behavioural programming paradigm. In FRP, both continuous and discrete events can be used to trigger actions. Recently, there have been two language extensions of note based on a functional language, Yampa [Hudak *et al.*, 2003] and Frob [Peterson *et al.*, 1999; 2001]. The language used in both cases is Haskell. These systems allow the programmer to specify how the robot reactions using very little code compared with procedural languages. The descriptions are based on behaviours and events. For example, in Yampa it is possible to write a wall following algorithm with just eight lines of code (building on some lower–level functions), shown in Fig.4.

While Yampa focuses mainly on the behavioural aspects, Frob is also designed with modularity in mind. It allows blocks of code to interact through interfaces, thus supporting code reuse. It provides pre-defined controller and sensor interfaces and a communications infrastructure. It also makes use of "tasks" to create sequentiality within the program.

FRP is not limited to languages such as Haskell. Dai *et al.* [2002] have implemented an FRP system in C++. It provides similar functionality to Frob, but also allows existing C++ code. It is simpler to use than Yampa and Frob, both of which require a good knowledge of functional programming.

One obvious trend is the change away from simple, command based languages, and towards higher–level languages that provide more support to the user, which is illustrated by the increasing popularity of behavioural languages. With more intelligent programming systems, the programmer is required to do less work to achieve the same results, increasing productivity.

## 2.2 Graphical Systems

Graphical (or icon–based) programming systems provide an alternative to text–based methods for manual programming. Although they are manual programming

```
rcFollowLeftWall :: Velocity -> Distance -> SimbotController
rcFollowLeftWall v d _ = proc sbi -> do
  let r = rfLeft sbi
  dr <- derivative -< r
  let omega = kp*(r-d) + kd*dr
      kd = 5
      kp = v*(kd^2)/4
  returnA -< mrFinalize (ddVelTR v (lim 0.2 omega))
```

Figure 4: A wall following algorithm implemented using Yampa.

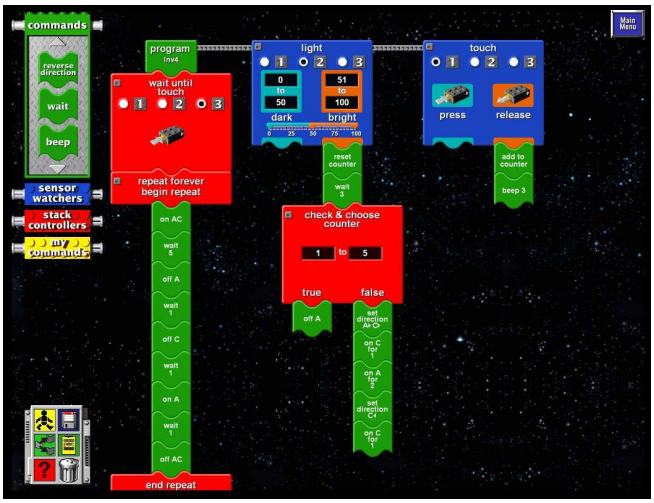


Figure 5: The Lego Mindstorms graphical programming environment, used to create simple programs for Lego robots.

methods, they are a small step closer to automatic programming, as they provide a graphical medium for programming. They require manual input to specify actions and program flow. Graphical systems typically use a graph, flow–chart or diagram view of the robot system. One advantage of graphical systems is their ease of use, which is achieved at the cost of text–based programming's flexibility. They are typically used for robot applications rather than system programming.

Perhaps the most successful graphical system using the flow–chart approach is employed by the Lego Mindstorms robotics kit [Lego, 2003], illustrated in Fig. 5. It is aimed at children, and so is simple by design. Blocks representing low-level actions are stacked like puzzle pieces to produce a sequence of actions. The sequences can be combined together to form a new block that can then be placed in a new stack, thus forming a simple hierarchy. A sequence is either attached to the main robot process, which defines its standard behaviour, or to a sensor where it defines the action taken when that sensor is triggered. While simple, this system allows for the creation of sophisticated behaviours.

In industrial environments, graphical systems enable rapid configuration of a robot to perform a required task. Bischoff *et al.* [2002] have produced a prototype style guide for defining the icons in a flow–chart system based on an emerging ISO standard (15187). Usability tests show that both experts and beginners found the graphical system easier for handling robots, for changing programs and for program overview. Touch screens are becoming popular for programming robots, and graphical systems using icons are ideally suited to this interface.

A graphical system for off-line programming of welding robots has been developed by Dai and Kampker [2000]. The main aim is to provide a user friendly interface for integrating sensor information in robot programs and so increase sensor use in welding robot programs. This is needed to overcome problems such as uncertainty of thermal effects. An icon–oriented interface provides the main programming method, with macros defined for sensor operations in a sensor editor. Macros make it easy to incorporate new sensors. The method could be used with any robot program where sensor information is used to mitigate inaccuracies.

A graph approach has been taken by Bredenfeld and Indiveri [2001] for their Dual Dynamics (DD–) Designer system, which takes a behaviour-based approach to controlling groups of mobile robots. The graphical programming interface uses a data processor hyper-graph, which is made up of data processing elements connected together by states. This approach allows the interaction between robot system elements to be specified.

## 3 Automatic Programming Systems

Automatic programming systems provide little or no direct control over the program code the robot will run. Instead, robot code is generated from information entered into the system in a variety of indirect ways. Often a robot system must be running while automatic "programming" is performed, and these systems have been referred to as "online" programming systems. However, automatic programming may also be performed on simulated or virtual robots, for example in industrial robotic CAD systems. In this case the real robot is off-line but

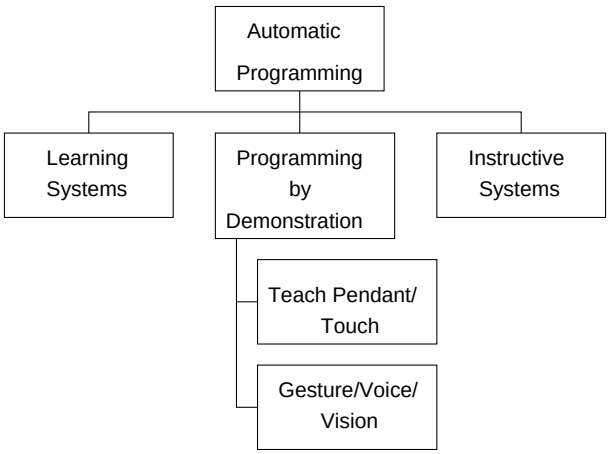


Figure 6: Categories of automatic programming systems. Learning systems, programming by demonstration and instructive systems are all methods of teaching robots to perform tasks.

the virtual robot is online. For example, the IGRIP [2003] system provides full simulation capabilities for creating and verifying robot programs.

Fig. 6 shows the three categories that automatic systems can be placed into: learning systems, programming by demonstration (PbD) and instructive systems. These are discussed in the following sections.

### 3.1 Learning Systems

Learning systems create a program by inductive inference from user provided examples and self–exploration by the robot. In the long run it will be crucial for a robot to improve its performance in these ways. A full review is beyond the scope of this paper. Examples include a hierarchy of neural networks developed for learning the motion of a human arm in 3D [Billard and Schaal, 2001], and a robot that can learn simple behaviours and chain these together to form larger behaviours [Weng and Zhang, 2002]. Smart and Kaelbling [2002] propose reinforcement learning for programming mobile robots. In the first phase the robot watches as the task is performed. In the second phase the robot attempts to perform the task on its own.

### 3.2 Programming By Demonstration

This is the most common method of automatic programming. Fig. 6 shows that PbD systems may use touch/pendants for the demonstration, or they may use other, more natural communication methods such as gestures and voice.

A traditional PbD system uses a teach-pendant to demonstrate the movements the robot should perform. This technique has been used for industrial manipulators for many years. The demonstrator performs the task (for example, an assembly task) using the teach pendant. The position of the pendant is recorded and the results used to generate a robot program that will move the robot arm through the same motions. Alternatively, the demonstrator may move the robot arm through the required motions either physically or using a controller. In this case, the joint positions are recorded and used to generate the robot program. Though simple, this type of system has been effective at rapidly creating assembly programs. Myers *et al.* [2001] describe an implementation by Intelligent Automation, Inc. It uses PbD to demonstrate subtasks, which are then grouped into sequential tasks by a programmer.

There are two current PbD research directions. The first is to produce better robot programs from the demonstrations, for example by combining multiple demonstrations and breaking down the information collected into segments. The second is to enhance demonstrations through the use of multi-modal communications systems.

Significant work has been conducted in recent years to develop PbD systems that are able to take the information produced from a demonstration, such as sensor and joint data, and extract more useful information from it, particularly for industrial tasks. Traditional PbD systems simply record and play back a single demonstration with no variation to account for changes or errors in the world. Much current research aims to introduce some intelligence to PbD systems to allow for flexible task execution rather than pure imitation.

Ehrenmann *et al.* [2002] describe a method for segmenting demonstrated data into moves (found by segmenting between grasps) and grasps (specifically, the actions performed during a grasp). The results of the segmentation can be stored for later playback on a robot. [Chen and McCarragher, 1998; 2000; Chen and Zelinsky, 2001] describe the progressive development of a similar system in which multiple demonstrations are used to build a partial view of the robot's configuration space. Optimal paths are generated between steps in a task. The motivation is that demonstrations rarely contain the best path between steps. This introduces significant flexibility to task performance. For example, the task can be biased towards maximum execution speed or maximum accuracy.

Ogawara *et al.* [2002] developed a system that integrates observations from multiple demonstrations. The demonstrated data is segmented to find important states such as grasps and moves. The multiple demonstrations are used to determine which segments are important to the task, and from this a flexible task model is built, for later execution on the robot.

The modality of the demonstration is also important; it may be touch, vision, gestures or voice. All have seen

active research in recent years. Grunwald *et al.* [2001] developed a method for natural touch in PbD. Rather than having to grip a robot arm at a certain point to move it for the demonstration, the demonstrator may hold the robot arm at any point, much as they would hold a human arm when indicating the movements that should be performed for a task. Without a requirement that the robot be gripped in a certain place, the robot becomes easier and more natural for a non-technical person to use.

Vision is also an important method of receiving demonstration information. However, it is difficult to produce a robust vision–based system that can operate in the typically cluttered environments of the real world. Special markers often must be used to indicate which objects the robot should be paying attention to during the demonstration, and the data from the demonstration is not as accurate as data from sensors on the robot. Yokokohji *et al.* [2002] developed a system to use cameras mounted close to the demonstrator's viewpoint, for acquiring the demonstration data. Both the demonstrator's hand motion and head motion are captured by tracking landmarks. Tests included the task of retrieving a CD from a CD rack, which showed that the task could be reproduced with sufficient accuracy. However, markers are required on all objects of interest in order to find landmarks.

Takamatsu *et al.* [2002] describe a method of producing more robust programs by correcting possible errors in demonstrated data from vision–based PbD. Contact states are checked to ensure they don't create such problems as having two objects in the same place. This ensures that incorrect results from a vision system do not produce erroneous programs.

There have been other advances in PbD systems. Onda *et al.* [2002] developed a virtual environment where demonstrations are performed. Contact state information can easily be retrieved from such an environment. Standard contact states may be replaced with special behaviours to overcome differences between the virtual world and various iterations of the real world. Instead of simply attempting to push a peg into a hole, the system will make the robot perform a search pattern to ensure the peg is correctly aligned with the hole and then move it in such a way that it goes into the hole smoothly. This is an imitation of how humans perform such a task, that is visually lining up the peg and the hole before moving it around until it goes in.

Other advances include the use of sensors on the fingers to detect fine manipulation of objects, for example turning a screw [Zollner *et al.*, 2002]. Friedrich *et al.* [1998] allow the results of the demonstration to be graphically viewed once the demonstration is complete. This allows the programmer to see what the robot will do as a result of the demonstration, and also allows different parts of the demonstration to be edited, moved around, and even used separately, producing code reuse for a PbD system.

Traditional robot CAD programming systems also provide a virtual, simulation environment in which a user may manipulate a robot to perform a task, and this is a form of PbD. Although the robot is off-line, the robot simulation is online.

The key trend in PbD is the increased intelligence of the programming system. Rather than just playing back a single demonstration, as was originally done, PbD systems are now capable of interpreting a demonstration and then using the interpreted data to produce robust, flexible programs capable of handling complex, changing worlds. PbD methods may include learning; some of the task description may be acquired by learning from the demonstrations.

### 3.3 Instructive Systems

Instructive systems are given a sequence of instructions, usually in real–time. The technique is best suited for commanding robots to carry out tasks that they have already been trained or programmed to perform; it could be considered the highest level of programming. Typically, gesture recognition or voice recognition is used.

Voyles and Khosla [1999] explored gesture–based programming using "Gesture Interpretation Agents." This is integrated into a PbD system. Steil *et al.* [2001] investigated the use of gestures for controlling the vision based robot GRAVIS. Gestures are used to direct the attention of the robot, and so enable its vision system to more easily find objects that are specified in instructions. This is useful for overcoming the problems caused by clutter in human environments. Strobel [2002] *et al.* used hand gestures for controlling a domestic cleaning robot. Static hand and arm gestures are captured with the robot's stereo vision system, or dynamic gestures are captured with a magnetic tracking system. Spatial knowledge is used to help determine the intent behind the gesture. The user could point to a surface that should be cleaned. Gesture recognition is useful for indicating objects in a scene that instructions apply to.

Language–based communication is the most natural method for humans to communicate instructions to one another, so it is a good candidate for robot instruction. A natural language system for providing directions to a robot is described by Lauria *et al.* [2002]. Natural language is used to teach the robot how to move to different locations by specified routes. It has fourteen motion primitives that are linked to natural language constructs. Unknown commands may be used by the user at some point, and some form of clarification and learning system would be needed.

Multi-modal communication has potential for simple robot programming. Vision systems provide a view of the world, and are used for gesture recognition (for example, gesturing commands or pointing at an object in the world). Gesture recognition and natural language recognition are used to give and clarify instructions to a robot. McGuire *et al.* [2002] describe a continuation of the work in [Steil *et al.*, 2001], mentioned earlier. The authors argue that a multi-modal system is a necessity for robots aimed at "more cognitively oriented environments" such as homes. They aim for human-like interaction. Information from all sources (vision, gestures and voice) may be used. For example, an instruction to pick up "that cube" could be given with voice while a gesture is used to indicate the cube to pick up, and the vision system provides a location for the cube.

Instructive systems have great potential for providing a high-level control method for robots. However, they still rely on the underlying trained or programmed abilities. These can be implemented only using other programming systems such as manual programming and through training with PbD systems.

## 4 Conclusions

Robot programming systems have become significantly more powerful and intelligent, moving beyond basic, text-based languages and record-and-play programming by demonstration, to more intelligent systems that provide considerable support to the user/programmer. Text-based languages are becoming higher–level, reducing the work required to implement systems. Graphical and automatic systems are becoming more powerful, allowing people with little or no technical skills to program robots.

The strongest trend is the addition of intelligence to programming systems to remove some of the burden from the programmer, both for manual programming systems and automatic programming systems. Text–based systems often supply much of the required low–level support, and programming by demonstration systems are becoming capable of building flexible task plans from demonstrations rather than just playing back the recorded data. Instructive systems are useful for providing a final, high level of control.

The development of these systems needs to be driven forward, beginning with solutions for robot developers that can then be scaled up to consumer solutions. The aim of these systems should be an environment that provides a consistent, simple interface for programming robots. Such a system would allow the general population to program robots with ease.

## References


[ABB, 2003] The ABB group. `http://www.abb.com/`, Oct 2003.

[Billard and Schaal, 2001] A. Billard and S. Schaal. Robust learning of arm trajectories through human demonstration. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 2, pages 734–739, Nov 2001.

[Bischoff *et al.*, 2002] 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.

[Bredenfeld and Indiveri, 2001] 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.

[Chen and McCarragher, 1998] 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.

[Chen and McCarragher, 2000] 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.

[Chen and Zelinsky, 2001] J.R. Chen and A. Zelinsky. Programming by demonstration: removing suboptimal 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.

[COSIMIR, 2003] COSIMIR. `http://www.cosimir.com/`, 2003.

[Dai and Kampker, 2000] Wenrui Dai and M. Kampker. User oriented integration of sensor operations in a offline programming system for welding robots. In *Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA '00)*, volume 2, pages 1563–1567, apr 2000.

[Dai *et al.*, 2002] Xiangtian Dai, G. Hager, and J. Peterson. Specifying behavior in C++. In *Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA '02)*, volume 1, pages 153–160, May 2002.

[Ehrenmann *et al.*, 2002] 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.

[Freund and Luedemann-Ravit, 2002] E. Freund and B. Luedemann-Ravit. A system to automate the generation of program variants for industrial robot applications. In *Intelligent Robots and System, 2002. IEEE/RSJ International Conference on*, volume 2, pages 1856–1861, 2002.

[Freund *et al.*, 2001] E. Freund, B. Ludemann-Ravit, O. Stern, and T. Koch. Creating the architecture of a translator framework for robot programming languages. In *Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA '01)*, volume 1, pages 187–192, May 2001.

[Friedrich *et al.*, 1998] H. Friedrich, J. Holle, and R. Dillmann. Interactive generation of flexible robot programs. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 1, pages 538–543, May 1998.

[Grunwald *et al.*, 2001] 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.

[Hardin *et al.*, 2002] D. Hardin, M. Frerking, P. Wiley, and G. Bolella. Getting down and dirty: device-level programming using the real-time specification for java. In *Object-Oriented Real-Time Distributed Computing, 2002. (ISORC 2002). Proceedings. Fifth IEEE International Symposium on*, pages 457–464, 2002.

[Hopler and Otter, 2001] R. Hopler and M. Otter. A versatile C++ toolbox for model based, real time control systems of robotic manipulators. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 4, pages 2208–2214, Nov 2001.

[Hudak *et al.*, 2003] Paul Hudak, Antony Courtney, Henrik Nilsson, and John Peterson. Arrows, robots, and functional reactive programming. In *Summer School on Advanced Functional Programming 2002, Oxford University*, Lecture Notes in Computer Science. Springer-Verlag, 2003. To Appear.

[IGRIP, 2003] Igrip and ultra products page. `http://www.deneb.com/products/igrip.html`, Oct 2003.

[iRobot, 2003] Roomba robotic vacuum cleaner. `http://www.roombavac.com/`, 2003.

[Kanayama and Wu, 2000] Y.J. Kanayama and C.T. Wu. It's time to make mobile robots programmable. In *Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA '00)*, volume 1, pages 329–334, apr 2000.

[Kitagishi *et al.*, 2002] I. Kitagishi, T. Machino, A. Nakayama, S. Iwaki, and M. Okudaira. Development of motion data description language for robots based on extensible markup language - realization of better understanding and communication via networks. In *Intelligent Robots and System, 2002. IEEE/RSJ International Conference on*, volume 2, pages 1145–1151, 2002.

[KUKA, 2003] KUKA automatisering + robots N.V. `http://www.kuka.be/`, Aug 2003.

[Lauria *et al.*, 2002] S. Lauria, G. Bugmann, T. Kyriacou, and E. Klein. Mobile robot programming using natural language. *Robotics & Autonomous Systems*, 38(3–4):171–181, March 2002.

[Lego, 2003] Lego Mindstorms. `http://mindstorms.lego.com/eng/products/ris/rissoft.asp`, 2003.

[Loffler *et al.*, 2001] M.S. Loffler, D.M. Dawson, E. Zergeroglu, and N.P. Costescu. Object-oriented techniques in robot manipulator control software development. In *American Control Conference, 2001. Proceedings of the 2001*, volume 6, pages 4520–4525, June 2001.

[Lozano-Pérez, 1982] 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.

[McGuire *et al.*, 2002] P. McGuire, J. Fritsch, J.J. Steil, F. Rothling, G.A. Fink, S. Wachsmuth, G. Sagerer, and H. Ritter. Multi-modal human-machine communication for instructing robot grasping tasks. In *Intelligent Robots and System, 2002. IEEE/RSJ International Conference on*, volume 2, pages 1082–1088, 2002.

[McKee *et al.*, 2001] G.T. McKee, J.A. Fryer, and P.S. Schenker. Object-oriented concepts for modular robotics systems. In *Technology of Object-Oriented Languages and Systems, 2001. TOOLS 39. 39th International Conference and Exhibition on*, pages 229–238, 2001.

[Myers *et al.*, 2001] D.R. Myers, M.J. Pritchard, and M.D.J. Brown. Automated programming of an industrial robot through teach-by showing. In *Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA '01)*, volume 4, pages 4078–4083, May 2001.

[Ogawara *et al.*, 2002] 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.

[Onda *et al.*, 2002] 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.

[Peterson *et al.*, 1999] J. Peterson, G.D. Hager, and P. Hudak. A language for declarative robotic programming. In *Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA '99)*, volume 2, pages 1144–1151, May 1999.

[Peterson *et al.*, 2001] J. Peterson, G. Hager, and A. Serjentov. Composable robot controllers. In *Computational Intelligence in Robotics and Automation, 2001. Proceedings 2001 IEEE International Symposium on*, pages 149–154, 2001.

[Pyro, 2003] Pyro, Python Robotics. `http://emergent.brynmawr.edu/pyro/?page=Pyro`, 2003.

[Smart and Kaelbling, 2002] W.D. Smart and L. Pack Kaelbling. Effective reinforcement learning for mobile robots. In *Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA '02)*, volume 4, pages 3404–3410, May 2002.

[Steil *et al.*, 2001] J.J. Steil, G. Heidemann, J. Jockusch, R. Rae, N. Jungclaus, and H. Ritter. Guiding attention for grasping tasks by gestural instruction: the gravis-robot architecture. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 3, pages 1570–1577, Nov 2001.

[Strobel *et al.*, 2002] M. Strobel, J. Illmann, B. Kluge, and F. Marrone. Using spatial context knowledge in gesture recognition for commanding a domestic service robot. In *Robot and Human Interactive Communication, 2002. Proceedings. 11th IEEE International Workshop on*, pages 468–473, 2002.

[Takamatsu *et al.*, 2002] 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.

[Thrun, 2000] S. Thrun. Towards programming tools for robots that integrate probabilistic computation and learning. In *Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA '00)*, volume 1, pages 306–312, apr 2000.

[Vaughan *et al.*, 2003] Richard T. Vaughan, Brian P. Gerkey, and Andrew Howard. On device abstractions for portable, reusable robot code. In *Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems (IROS03)*, pages 2421–2427, Las Vegas, Nevada, October 2003.

[Voyles and Khosla, 1999] R.M. Voyles and P.K. Khosla. Gesture-based programming: a preliminary demonstration. In *Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA '99)*, volume 1, pages 708–713, May 1999.

[Weng and Zhang, 2002] Juyang Weng and Yilu Zhang. Action chaining by a developmental robot with a value system. In *Development and Learning, 2002. Proceedings. The 2nd International Conference on*, pages 53–60, 2002.

[Yokokohji *et al.*, 2002] 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.

[Zollner *et al.*, 2002] 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.