

Programming Languages for Robotics: A Comprehensive Review of Applications, Features, and System Integration

Rone*

Abstract

The rapid advancement of industrial automation, artificial intelligence, and autonomous systems has increased the importance of effective programming approaches for modern robotic applications. Robot programming languages provide the computational foundation for controlling robotic motion, interacting with sensors and actuators, implementing autonomous operations, and developing simulation-based solutions. This paper presents a comparative review of major programming languages used in robotics, with particular emphasis on C/C++, Python, Java, C#.NET, and MATLAB. The study examines their programming characteristics, hardware interaction capabilities, development environments, simulation support, software integration, and suitability for different robotic applications. C/C++ is considered in relation to low-level hardware interaction and real-time robotic control, while Python is examined for rapid development, scripting, application programming interfaces, and simulation. Java is discussed for high-level programming and artificial intelligence-oriented applications, whereas C#.NET is considered for integrated development environments and software-level robotic applications. MATLAB is examined primarily from the perspective of mathematical modeling, simulation, algorithm development, and robotic system analysis. The review also distinguishes coding, programming, and computer science concepts within the broader context of robotics education and development. The analysis indicates that no single programming language provides an optimal solution for every robotic application; instead, language selection depends on hardware requirements, computational constraints, development objectives, simulation needs, and system integration. The study provides a structured foundation for understanding the role of programming languages in contemporary robotics and identifies important considerations for developing efficient, adaptable, and intelligent robotic systems.

Keywords: Robotics, robot programming, programming languages, C/C++, Python, Java, C#.NET, MATLAB, industrial robotics, artificial intelligence

INTRODUCTION

The rapid development of robotics, artificial intelligence, and industrial automation has significantly

*Author for Correspondence

Rone

E-mail: rone.somadder@mimt.org

Assistant Professor, Department of Engineering, Mangalmai Institute of Management and Technology, Greater Noida, Uttar Pradesh, India

Received Date: September 26, 2026

Accepted Date: September 28, 2026

Published Date: September 30, 2026

Citation: Rone. Programming Languages for Robotics: A Comprehensive Review of Applications, Features, and System Integration. Recent Trends in Programming Languages. 2026; 13(2): 51–62p.

increased the role of programming in the design, control, simulation, and deployment of robotic systems. Robots are increasingly employed to perform tasks that require precision, repeatability, speed, and controlled operation, particularly in industrial environments. Consequently, effective robot programming has become an important component of modern robotic system development. The present review identifies the growing interest in industrial robotics and emphasizes the increasing requirement for programming expertise as robotic applications expand.

Robot programming involves providing a robotic system with a sequence of instructions or objectives through which it performs the desired operation. Programming can be performed through direct/online approaches or through offline programming environments. In offline programming, higher-level programming languages can be used to define robotic tasks before their implementation on the physical system. The selection of an appropriate programming language therefore influences the development process, hardware interaction, computational performance, simulation capability, and integration of robotic applications. Several programming languages have been employed in robotics, each providing different capabilities and development characteristics. The present study specifically discusses C/C++, Python, Java, C#.NET, and MATLAB. C and C++ provide capabilities for interaction with lower-level hardware and are associated with performance-oriented robotic applications. Python provides a high-level environment that supports scripting, robot programming, simulation, and application programming interfaces, thereby facilitating rapid development and testing. Java is discussed in relation to high-level robotic and artificial intelligence-oriented applications, including machine learning and language processing algorithms. C#.NET provides an integrated software development environment and supports interaction among different programming technologies. MATLAB is particularly relevant to mathematical modeling, simulation, algorithm development, and robotic system analysis.

LITERATURE REVIEW

Robot programming has evolved considerably from conventional teach-pendant and proprietary programming approaches toward flexible, software-based, simulation-assisted, and human-centered programming environments. Early research by Hocken and Morris reviewed the development of offline robot programming systems and highlighted the importance of programming languages, performance measures, and technological barriers associated with offline programming [1]. This early work established the importance of separating robot programming from direct manipulation of physical hardware and provided a foundation for subsequent development of offline programming systems. More recent research has demonstrated that robot programming remains a significant challenge because conventional programming can require substantial technical expertise, time, and production-line interruption [2].

Pan *et al.* presented a comprehensive review of industrial robot programming and classified the major approaches into online, offline, and augmented-reality-assisted programming [2]. Their analysis demonstrated that programming complexity can restrict the adoption of robotics, particularly in small and medium-sized enterprises. The authors further identified CAD/PLM systems, computer vision, sensing technologies, and advanced software environments as important technologies for improving robot programming.

Robot programming has also increasingly incorporated programming by demonstration (PbD) and learning from demonstration (LfD). Calinon *et al.* reviewed robot PbD and demonstrated how robots can acquire task-related behaviors from human demonstrations rather than relying entirely on manually written programs [3]. Argall *et al.* subsequently provided a broad survey of robot LfD, examining demonstration acquisition, policy derivation, and performance evaluation [4]. More recent work by Ravichandar *et al.* showed that LfD has continued to develop as an important approach for reducing the coding expertise required to teach robots new behaviors [5].

The development of human–robot interaction (HRI) has further influenced robot programming. Villani *et al.* reviewed HRI approaches for industrial robot programming and discussed online and offline programming, together with multimodal interaction, virtual reality, and augmented reality [6]. These approaches attempt to make robot programming more intuitive and accessible to users who may not possess extensive conventional programming expertise.

Augmented reality has consequently emerged as an important research direction. Chong *et al.* investigated augmented-reality-based robot programming for planning collision-free robot paths [7]. Ong *et al.* developed an augmented-reality-assisted robot programming system intended to make

programming faster and more intuitive for industrial applications [8]. Their approach incorporated robot motion planning, collision detection, sensor information, and task definition within an AR environment. Zhang *et al.* further examined AR as an alternative to conventional teaching-pendant and offline programming approaches, identifying limitations associated with production-line interruption and the requirement for detailed CAD models [9].

Recent research has extended AR programming toward virtual representations of complete robotic workspaces. Thoo *et al.* proposed an AR interface capable of supporting both online and offline robot programming through a virtual representation of the workspace, allowing operators to develop and reuse robot tasks without necessarily remaining co-located with the physical robot [10]. Lambrecht and Krüger also investigated spatial programming using natural communication and AR, reporting reductions in programming duration and programming errors compared with conventional approaches in their user study [11].

Another major development concern is the robot software framework and middleware. Quigley *et al.* introduced the Robot Operating System (ROS), an open-source framework designed to facilitate software development for robotic systems through reusable tools, libraries, communication mechanisms, and hardware abstraction [12]. ROS and related middleware approaches have contributed to the integration of different programming languages, robotic hardware, sensors, simulation tools, and intelligent algorithms into unified development environments.

The choice of programming language remains an important component of this software ecosystem. The present review identifies C/C++, Python, Java, C#.NET, and MATLAB as major programming environments associated with robotics. C/C++ are particularly relevant where direct hardware interaction and computational performance are important. Python provides a high-level development environment suitable for scripting, simulation, APIs, and rapid application development. Java has been associated with high-level software and AI-oriented applications, whereas C#.NET provides an integrated software development environment. MATLAB is primarily useful for mathematical modeling, simulation, algorithm development, and analysis of robotic systems. The literature also demonstrates a close relationship between programming and robot motion planning. Khatib established the artificial-potential-field concept for real-time obstacle avoidance and robot motion control [13]. Kavraki *et al.* introduced probabilistic roadmaps for motion planning in high-dimensional configuration spaces [14], while Zucker *et al.* developed CHOMP for optimizing robot motion trajectories [15]. These studies demonstrate that robot programming increasingly involves not only explicit motion commands but also algorithms capable of planning, optimization, collision avoidance, and adaptation.

Programming approaches based on demonstration have also been applied specifically to robotic assembly. Zhu and Hu reviewed robot LfD for assembly applications and identified task representation, demonstration acquisition, generalization, and adaptation as important research challenges [16]. Such approaches are particularly relevant to flexible manufacturing because they can potentially reduce the amount of manually developed task-specific code. The increasing demand for flexible automation has also motivated research into low-code and graphical robot programming. Recent research categorizes low-code robot programming according to application scenarios and user groups and identifies graphical programming and human-machine interaction as important directions [17]. This direction is significant because conventional programming can create a barrier for operators who have strong manufacturing knowledge but limited software-programming expertise.

METHODOLOGY

Research Design

This study adopts a structured comparative review methodology to examine programming languages and programming approaches used in robotic systems [18]. The methodology was developed from the programming concepts and languages discussed in the present review, particularly C/C++, Python,

Java, C#/.NET, and MATLAB. The literature review considers research addressing robot programming, offline programming, PbD, human–robot interaction, augmented-reality-based programming, robot middleware, simulation, and emerging programming approaches [19].

Classification of Programming Languages

The selected programming environments are classified according to their primary role in robotic system development:

- C/C++—Hardware interaction, performance-oriented programming, and robot control.
- Python—High-level programming, scripting, APIs, and rapid development.
- Java—High-level software development and AI-oriented applications.
- C#/.NET—Integrated software development and application programming.
- MATLAB—Modeling, simulation, algorithm development and analysis.

This classification follows the scope of the present review

Evaluation Parameters

Each programming environment is examined using common parameters so that the comparison remains consistent:

1. Programming level—high-level or lower-level programming capability.
2. Hardware interaction—ability to interface with robotic hardware, sensors, and actuators.
3. Computational performance—suitability for performance-sensitive robotic operations.
4. Development complexity—relative programming effort and learning requirements.
5. Simulation capability—support for modeling and testing robotic systems before physical implementation.
6. AI and machine learning support—suitability for implementing intelligent robotic algorithms.
7. Software integration—compatibility with robotics frameworks, APIs and external software.
8. Industrial applicability—relevance to industrial automation and robotic applications.
9. Development flexibility—suitability for modifying, extending and adapting robotic applications.

Robot Programming Approaches

In addition to programming languages, this study considers the principal approaches through which robotic systems can be programmed. These include online/direct programming and offline programming. The methodology also considers developments reported in the wider literature, including PbD, augmented-reality-assisted programming, simulation-based programming, and software-framework-based development. These approaches are examined as complementary methods rather than replacements for individual programming languages [20].

Comparative Analysis

Comparative analysis does not assign a universal ranking to programming languages. Instead, the characteristics of each language are interpreted according to the requirements of a particular robotic application. Hardware-oriented and performance-sensitive applications are examined separately from rapid application development, simulation, AI implementation, and mathematical modeling.

COMPARATIVE ANALYSIS OF PROGRAMMING LANGUAGES FOR ROBOTICS

Robotics Market- Overview

Human efforts to perform advanced tasks become easier and more precise with the help of robots. With this in mind, the field unit of many corporations has shown interest in robot and field unit finance because of its advantages. "Per research and market, the industrial AI market is projected to grow to US \$48,166.9 million by 2025, from US \$16 to 847.6 million in 2018. While robot field units have historically been used in structured environments, industrial robots have been gaining momentum in

recent years, with well-known and controlled inputs and outputs. With this, individual sectors show interest in programming, and people with some expertise in programming need to take advantage of robot programming. It is expected that the demand for robot programmers is projected to increase over the remainder of the last decade.

Programming of Robot

The robot is programmed by directing or offline programming to perform a selected action. Most economic mechanisms are programmed by purposefully directing a robot through the associated stages in operation, with every objective within the robot's system. Robots receive directions via PC commands and can be seen as manipulator-level offline programming. The use of offline programming includes high-level languages, within which robotic tasks are outlined by tasks or objectives.

Robot programmers must have data on a variety of programming languages because shifting from computer to robot is not a pleasant transition that many developers/programmers can imagine.

Different Types of Robot Programming Languages

Programming machines have a perceptual effect on robot integration in industrial machines. The sky is the limit for robotic industrial applications. The robot software package plays an important role in the task of advanced operation and correct functionality. To start programming in artificial intelligence, you want to start learning a variety of languages to create robot work. Although artificial intelligence can be an advanced subject, learning these programming languages can help you build a thesis so that associate degrees can be used easily.

There are more than 1500 programming languages in the world that are used for learning and implementation. Here are the most popular and most liked programming languages in robotics:

- C/C ++ and C++ have to be told that this is the best start for robot enthusiasts as it can be a general-purpose artificial language that includes mandatory, object-oriented, and generic programming options.
- Why is C ++ that there is no 1 artificial language in robotics?
- C++ allows interaction with low-level hardware and provides additional real-time performance. C and C++ are mature programming languages. To ensure the simplest performance of an automobile, C++ should be used.
- As AI is incredibly involved in critical real-time performance, C and C++ are the simplest options for robotics Figure 1

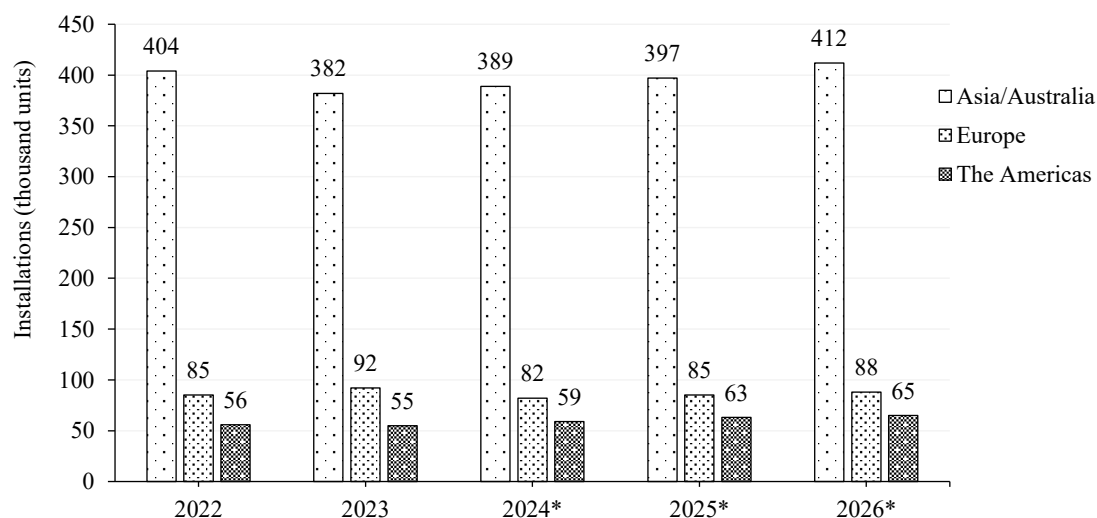


Figure 1. Industrial robots' statistics.

Forecast for World Robotics 2024, values in thousands of units.

Source: International Federation of Robotics (IFR), World Robotics 2024, press conference 2024, p.26.

```

#include "WPIlib.h"

class RobotDemo : public SimpleRobot
{
public:
1 RobotDemo(void)
  {
    //put initialization code here
  }
2 void Autonomous(void)
  {
    //put autonomous code here
  }
3 void OperatorControl(void)
  {
    //put teleop code here
  }
};
START_ROBOT_CLASS(RobotDemo);

```

Figure 2. Basic structure of a WPILib robot program.

```

class RobotDemo : public SimpleRobot
{
1 RobotDrive myRobot; // robot drive system
  Joystick stick; // only joystick

public:
  RobotDemo(void):
2   myRobot(1, 2), // these must be initialized in the same order
   stick(1)       // as they are declared above.
3 {
   myRobot.SetExpiration(0.1);
};

```

0.1

ROBOT SYSTEM CONFIGURATION (0.1)

Figure 3. Robot drive system and joystick initialization.

Example: Default Simple Robot Template

Now, let us locate the code contained within each section (Figure 2 and 3) There are three main parts: constructor:

This can be wherever the data format code runs to create a range. In this case, it will run before 1 program starts, before the robot is enabled. The builder helps initialize sensors and create optional WPILib objects. While starting, we do not have a tendency for actors to use positioners because the motors will not operate. Autonomous Function: This method is employed to run code in autonomous mode. Autonomous Methodology can run at any time when the robot is placed in autonomous mode. A coder must be very careful while running this method, as its execution is governed by the robot's autonomous operating mode.

Operator Control Methodology: This method refers to the situation in which the robot becomes part of the match in teleoperation. The code includes a loop that reads the operator interface values (joystick and switch) and operates the actuators until the teleop amount is exhausted.

Defining variables for a sample robot:

The sample robot includes a joystick on USB port 1 for arcade drives and two motors on PWM ports 1 and 2. Here, we create objects of type robotDrive (myRobot) and joystick (stick). This part of the code performs three tasks:

1. Defines variables as members of our RobotDemo class.
2. Initializes variables as part of the constructor using an initialization list.
3. Robot initializes (in this case, myRobot sets the security timer expiration for the object).
4. Configures the motor-safety timer associated with the robot- drive object.

Python

Python is a high-level programming language that plays an important role in the creation and testing of robots. Python robots can be a sensible platform for modifying, teaching, and posting programs. Many of us choose this programming language because we are able to write scripts that record and emulate a complete robot program instead of having to manually teach each statement to a robot. This helps to quickly visualize the answer in the simulation as it refines the program and its logic.

Why is Python the Most Commonly Used Programming Language in Robotics?

- Compared to C, C ++, and Java, you can write fewer lines of code in Python.
- Learning Python is an essential step in building autonomous mobile robots using the popular robot software framework.
- This language is used in designing embedded systems and is an integral part of robotic operating systems.
- The major focus of Python is ease of use, and less time is required for defining programs and casting variable types.
- When you need to implement some basic functionality, there are a large number of free libraries for this, meaning you do not have to reinvent the wheel.

Example Program

This example shows that all objects are spread mechanically through Python. We are only able to programmatically place any object because of the Python API. Incorrect use of the Python API: We are able to create, modify, or delete any object. Additionally, we are able to programmatically move mechanisms within the same Python script, allowing the North American nation to mechanically create system programs Figure 4 and 5

Java

To perform human-like tasks, artificial intelligence systems must be attached to tasks through programming languages. To replace robots with these functions, the Java language will be used as it offers a collaborative Java provides libraries and APIs that can support AI- oriented robotic applications. Command-and-control recognition, dictation systems, and Speech-related functionality can be implemented using the Java Speech API, while Java- based multimedia facilities can support related processing requirements.

```
1. from robolink import * # API to communicate with RoboDK
2. from robodk import * # basic matrix operations
3.
4. # Any interaction with RoboDK must be done through
5. # Robolink()
6. RL = Robolink()
7.
8. # get the robot item:
9. robot = RL.Item('KUKA KR 6 R900 sixx')
10.
11. # get the home target and the welding targets:
12. home = RL.Item('Home')
13. target = RL.Item('Target 1')
14. # get the pose of the target (4x4 matrix):
15. poseref = target.Pose()
16.
17. # move the robot to home, then to the center:
18. robot.MoveJ(home)
19. robot.MoveJ(target)
20.
21. # make an hexagon around the center:
22. for i in range(7):
23.     ang = i*2*pi/6 #angle: 0, 60, 120, ...
24.     posei = poseref*rotz(ang)*transl(200,0,0)*rotz(-ang)
25.     robot.MoveL(posei)
26.
27. # move back to the center, then home:
28. robot.MoveL(target)
29. robot.MoveJ(home)
```

Figure 4. Robot control and movement using RoboDK.

```

import javax.speech.*;
import javax.speech.recognition.*;

// Create recognizer and allocate resources
Recognizer recognizer = Central.createRecognizer(null);
recognizer.allocate();
/* Add engine listener - to notify when speech engine
 * is stopped, started, etc.
 */
recognizer.addEngineListener(engineListener);
// Read-in grammar file
File gf = new FileReader(grammarFile);
RuleGrammar rules = recognizer.loadJSGF(gf);
/* Add result listener. VoiceListener then called when a
 * particular grammar word is recognized. The associated
 * event tag is passed to it
 */
rules.addResultListener(new VoiceListener());
// Tell recognizer to commit changes
recognizer.commitChanges();
// Request focus of microphone away for other apps.
recognizer.requestFocus();
// Start listening
recognizer.resume();
// Remember to catch exceptions

```

Figure 5. Speech recognition setup and voice command processing in Java.

Why is Java the most used programming language in robotics?

- Java includes all the high-level features required in the robotics industry, especially when it comes to artificial intelligence.
- Using Java, you can create highly efficient algorithms for machine learning, search, and neural algorithms, language processing, and more.
- Interprets instructions at the Java Virtual Machine runtime.
- The Java virtual machine enables Java users to use the same code on many different machines.
- This provides all the high-level features required to deal with various aspects of robotics.
- Example code to create a robotics voice listener using a grammar format file.

C#/.NET

C# is a general-purpose programming language developed by Microsoft and commonly used with the .NET platform for application development. Examples include Netduino, FEZ Rhino, and others. This provides aspiring programmers with a solid foundation on which they can branch out into different fields. C#/.NET is commonly used in port- and socket-level programming Figure 6.

Why C#/.NET is the most used programming language in robotics?

- C#/.NET allows multiple languages to be used and has horizontal scalability.
- .NET provides an integrated development environment in which applications can be developed using languages such as C#, Visual Basic, and F#
- All tools and IDEs are pre-tested and readily available in the Microsoft Developer Network.
- Language integration is seamless, as you can call methods from C# to VB.NET.

Examples:

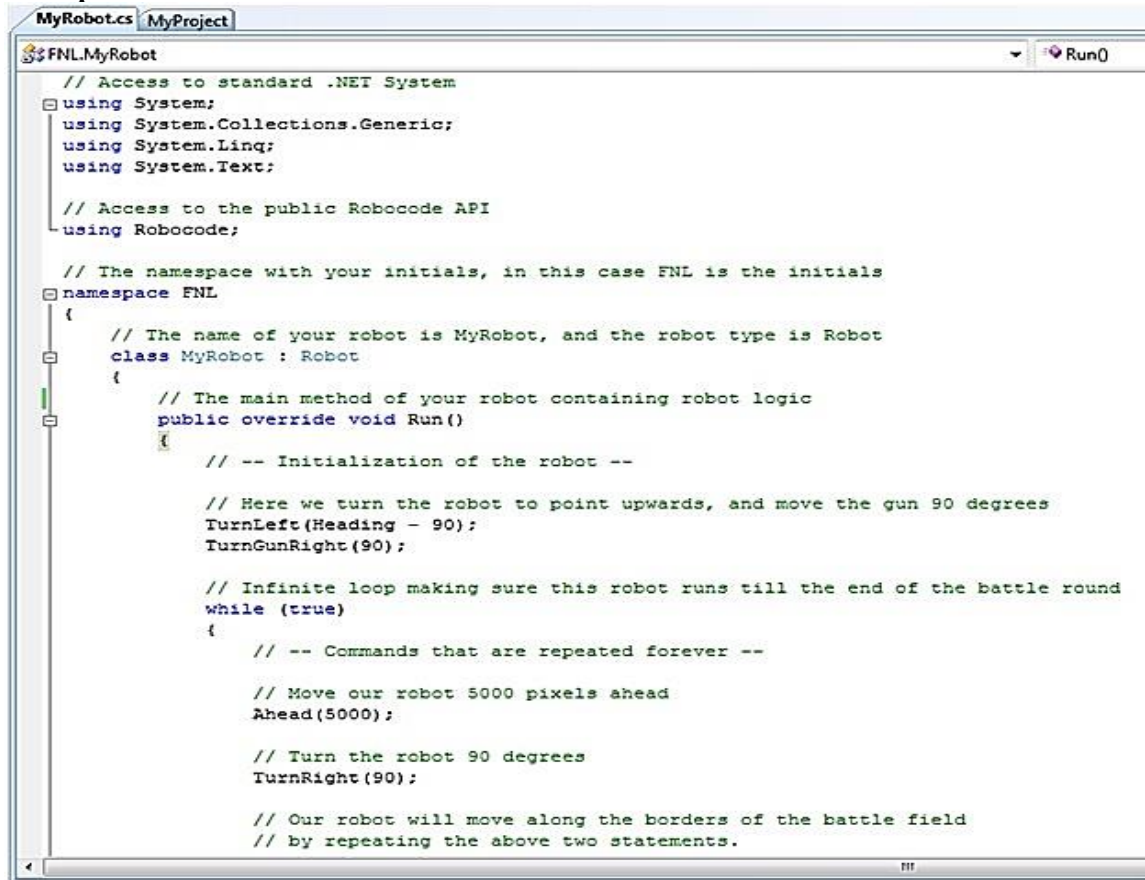


Figure 6. C# code for a robocode robot.

Source: RoboWiki

MATLAB

Technically speaking, MATLAB is not an artificial language, although it is a tool with which you will realize engineering solutions supported by arithmetic. Robot developers must learn MATLAB if they need to research information, create advanced graphs, or implement management systems Figure 7

MATLAB and its open-source relative, such as Octave, are used by very few robot engineers for analyzing information and developing management systems.

Programming for an automaton requires a controller that controls the automaton's behavior. However, modeling and simulation have become important for interacting with perception, dynamics, and interactions around the controller and robot.

Why is MATLAB the most used programming language in robotics?

- MATLAB is highly useful in designing entire robotic systems.
- It is widely used in the robotics industry because it is deeply rooted in the foundation and development of robots.
- This is a simulation tool by which you can provide your algorithm or design, and it simulates the result.
- Simulation helps the system to refine system prototypes and eliminate errors before developing hardware prototypes.

Example: modeling and simulation of a KUKA KR6 robot Figure 7.

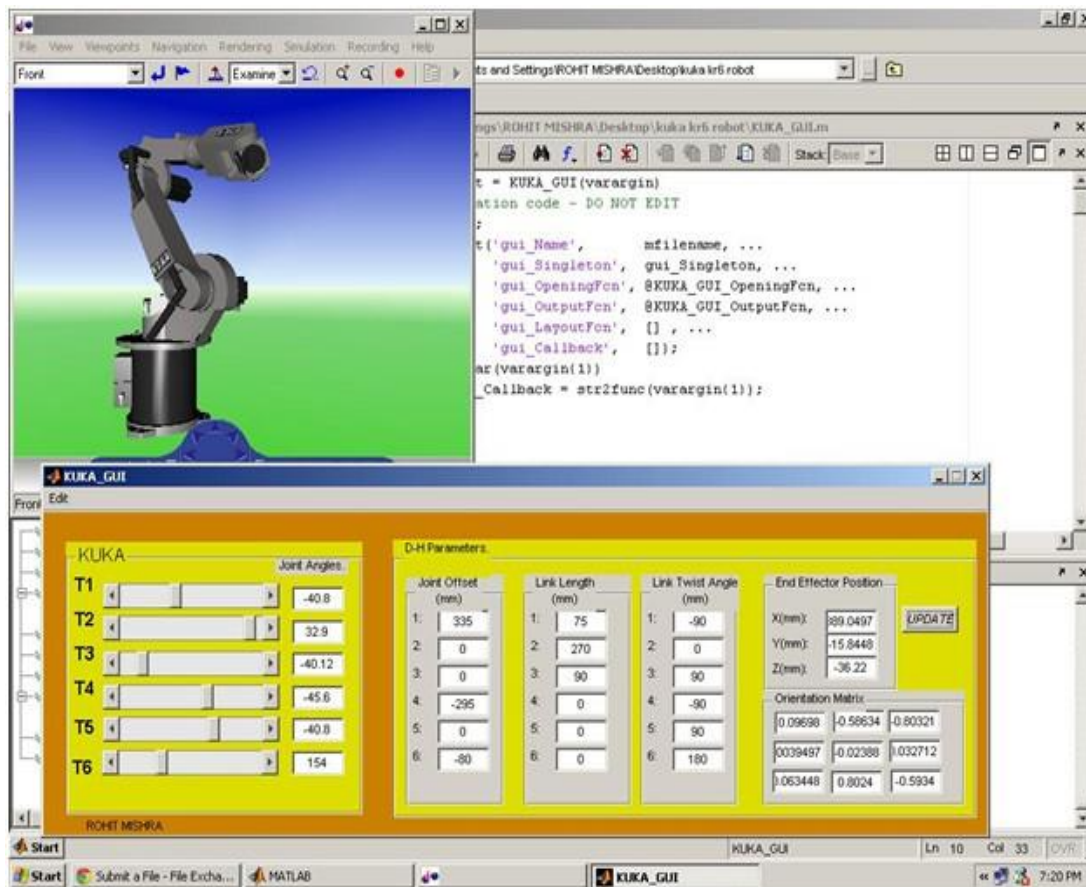


Figure 7. MATLAB GUI for a KUKA robot simulation.

Which one of them is the best?

Learning these programming languages can help you develop a robot or gain employment within the artificial intelligence business. However, learning C/C++ on a priority basis can help you, as they are the simplest programming languages for artificial intelligence.

Although some applications have adopted newer programming approaches, C and C++ continue to be widely used in robotics because of their mature toolchains, libraries, and performance-oriented capabilities. Many companies opt for C and C++ to develop robots because they have a lot of tools and library functions. These two language area units will be developer friendly, and the developers will modify line by line. However, learning these programming languages can help you develop writing skills, and they will help you create a tremendous ultimate Intelligent Java.

Coding, Programming and Computer Science are not Equal

You have found, in all probability, the push to produce students with cryptography and programming skills as a way to prepare them for the 21st century and future careers. Several firms such as code.org, Code Academy, Tinker, Programming Basics, RobotLab, and others provide a proliferation of learning exercises to show cryptography and programming to students in a very fun and direct way. There are also online platforms for youth under five years of age.

Nevertheless, many differences exist between cryptography and programming, and it is important to understand what makes them distinctive. Each student incorporates a completely different goal for their learning; the selected ability, and whether or not to enhance their career path or interact with their passion, should ensure that they are required to become a cryptography or technologist.

Computer science is the study of almost everything associated with computers: theory, hardware design, algorithms for resolution of a variety of issues, logical structures, constrained rules that operate and manage automated data processing systems, and so on. This actually provides a theoretical explanation of working computers. It is about locating processes that are capable of detecting processes, the set of instructions to complete tasks, and what computers are and do not seem capable of doing. Programming and cryptography are the only significant elements of the broader field, however, that we have in computing decision-making tendencies.

There is a lot of confusion between programming and cryptography, and each is very suggestive. The above programming is primarily a method of creating bugs or collaborative applications that run once, which should result in results. In the broader sense, the term programming refers to all aspects of activity and creating a program, including in-depth analysis of a problem, developing logic, springing with the right algorithms, and maintaining a program, testing, debugging, and maintaining it. This is all part of programming, whereas cryptography is simply an implementation of an algorithmic program in a highly specialized artificial language (C++, Python, Java, etc.). It is translating logic into real code that makes it understandable to humans and machines. Let us provide this collaborative example: imagine building a library. The library has an individual, accountable, collaborative designer to come up with the development. Before developing a system, a designer may consider factors such as functionality, safety, security, appearance, and available space. He can verify specific processes to realize his goal. A technologist is an Associate Designer who creates a program for the World Health Organization. A software developer or programmer implements the planned design by translating the required logic and algorithms into executable code. Cryptography is simply the collaborative act of writing code.

CONCLUSION

This study reviewed and comparatively examined major programming languages used in robotics, with emphasis on C/C++, Python, Java, C#.NET, and MATLAB. The analysis demonstrates that robot programming has developed from conventional, controller-oriented programming toward more flexible approaches involving high-level languages, simulation, software frameworks, artificial intelligence, and interactive programming.

C/C++ are associated with hardware interaction and performance-oriented robotic programming, whereas Python provides a high-level environment suitable for scripting, rapid development, APIs, and simulation. Java and C#.NET provide additional options for high-level software development and integration, while MATLAB is particularly relevant to mathematical modeling, simulation, algorithm development, and robotic system analysis.

The review indicates that programming language selection is application-dependent. Factors such as hardware interaction, computational requirements, development complexity, simulation capability, artificial intelligence integration, software interoperability, and intended robotic application need to be considered collectively. Therefore, the analysis does not establish a single programming language as universally suitable for robotics.

This study also highlights the continuing transition toward offline programming, PbD, augmented-reality-assisted programming, simulation-based development, and emerging low-code approaches. These developments can reduce programming complexity and provide alternative mechanisms for defining robotic tasks.

Overall, the comparative framework developed in this study provides a structured basis for understanding the capabilities and application domains of major programming environments in robotics. Future research can extend this work through quantitative benchmarking using identical robotic tasks, evaluating execution time, computational overhead, memory requirements, development effort, simulation accuracy, and hardware-response characteristics across different programming environments.

REFERENCES

1. Hocken R, Morris G. An overview of off-line robot programming systems. *CIRP Ann.* 1986;35(2):495-503. doi:10.1016/S0007-8506(07)60198-X.
 2. Pan Z, Polden J, Larkin N, Van Duin S, Norrish J. Recent progress on programming methods for industrial robots. *Robot Comput Integr Manuf.* 2012;28(2):87-94. doi:10.1016/j.rcim.2011.08.004.
 3. Calinon S. *Robot Programming by Demonstration*. Lausanne: EPFL Press; 2009.
 4. Argall BD, Chernova S, Veloso M, Browning B. A survey of robot learning from demonstration. *Robot Auton Syst.* 2009;57(5):469-483. doi:10.1016/j.robot.2008.10.024.
 5. Ravichandar H, Polydoros AS, Chernova S, Billard A. Recent advances in robot learning from demonstration. *Annu Rev Control Robot Auton Syst.* 2020;3(1):297-330. doi:10.1146/annurev-control-100819-063206.
 6. Villani V, Pini F, Leali F, Secchi C, Fantuzzi C. Survey on human-robot interaction for robot programming in industrial applications. *IFAC-PapersOnLine.* 2018;51(11):66-71. doi:10.1016/j.ifacol.2018.08.236.
 7. Chong JWS, Ong SK, Nee AYC, Youcef-Youmi KB. Robot programming using augmented reality: an interactive method for planning collision-free paths. *Robot Comput Integr Manuf.* 2009;25(3):689-701. doi:10.1016/j.rcim.2008.05.002.
 8. Ong SK, Yew AWW, Thanigaivel NK, Nee AYC. Augmented reality-assisted robot programming system for industrial applications. *Robot Comput Integr Manuf.* 2020;61:101820. doi:10.1016/j.rcim.2019.101820.
 9. Zhang F, Lai CY, Simic M, Ding S. Augmented reality in robot programming. *Procedia Comput Sci.* 2020;176:1221-1230. doi:10.1016/j.procs.2020.09.119.
 10. Thoo YJ, Maceiras J, Abbet P, Racca M, Girgin H, Calinon S. Online and offline robot programming via augmented reality workspaces [Preprint]. 2021. arXiv:2107.01884.
 11. Lambrecht J, Krüger J. Spatial programming for industrial robots: efficient, effective and user-optimised through natural communication and augmented reality. *Adv Mater Res.* 2014;1018:39-46. doi:10.4028/www.scientific.net/AMR.1018.39.
 12. Quigley M, Conley K, Gerkey BP, Faust J, Foote T, Leibs J, et al. ROS: an open-source robot operating system. In: *ICRA Workshop on Open Source Software*. Kobe, Japan; 2009. p. 5.
 13. Khatib O. Real-time obstacle avoidance for manipulators and mobile robots. *Int J Robot Res.* 1986;5(1):90-98. doi:10.1177/027836498600500106.
 14. Kavraki LE, Svestka P, Latombe JC, Overmars MH. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans Robot Autom.* 1996;12(4):566-580. doi:10.1109/70.508439.
 15. Zucker M, Ratliff N, Dragan AD, Pivtoraiko M, Klingensmith M, Dellin CM, et al. CHOMP: covariant Hamiltonian optimization for motion planning. *Int J Robot Res.* 2013;32(9-10):1164-1193. doi:10.1177/0278364913488805.
 16. Zhu Z, Hu H. Robot learning from demonstration in robotic assembly: a survey. *Robotics.* 2018;7(2):17. doi:10.3390/robotics7020017.
 17. Wang P, Wang Z, Duan J, Miao Y, Tian W. Review of the development status of low-code programming software for robots. *Digit Eng.* 2025;4:100039. doi:10.1016/j.dte.2025.100039.
 18. Schulman J, Ho J, Lee AX, Awwal I, Bradlow H, Abbeel P. Finding locally optimal, collision-free trajectories with sequential convex optimization. In: *Robotics: Science and Systems IX*. Berlin, Germany: Robotics: Science and Systems Foundation; 2013. doi:10.15607/RSS.2013.IX.031.
 19. Biggs G, MacDonald BA. A survey of robot programming systems. In: *Proceedings of the Australasian Conference on Robotics and Automation*. Sydney, Australia; 2003. p. 1-6.
 20. Fang HC, Ong SK, Nee AYC. Interactive robot trajectory planning and simulation using augmented reality. *Robot Comput Integr Manuf.* 2012;28(2):227-237. doi:10.1016/j.rcim.2011.09.003.
-