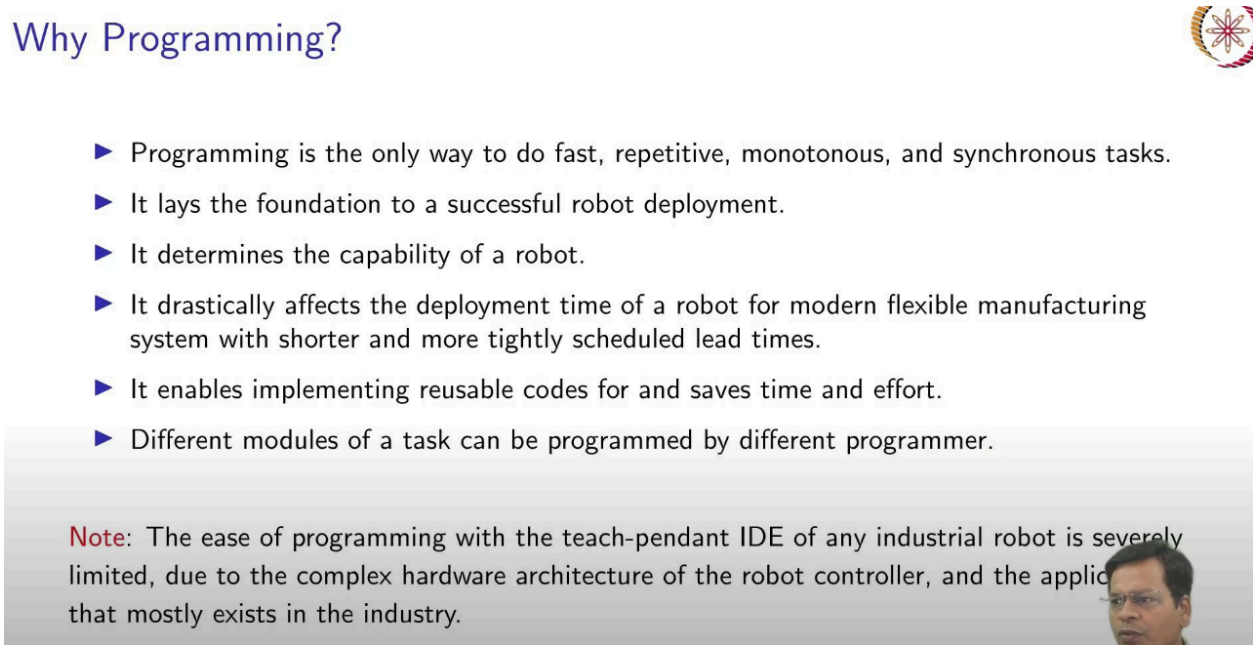
NPTEL Online Certification Courses
Industrial Robotics: Theories for Implementation
Dr Arun Dayal Udai
Department of Mechanical Engineering
Indian Institute of Technology (ISM) Dhanbad
Week: 12
Lecture 49

Industrial Robot Programming

Welcome to the last lecture of the course Industrial Robotics, Theories for Implementation. Thanks for everything so far. You are now an expert learner of industrial robots who can understand kinematics, dynamics, calibration, safety and norms. With everything in place now you need to run your robot to its film capacity. So, let us begin with that in today's lecture Industrial Robot Programming.



So, why do we need programming first of all? Programming is the only way to do fast, repetitive, monotonous and synchronous tasks. So, you understand industrial robots are into tasks which are mostly quite repetitive in nature. Every second it is doing plenty good amount of tasks which are quite monotonous as well. If at all it comes to a person who has to do it, he may not be able to do it also. So, let us say if it is tightening a bolt, a whole lot of days it is tightening the bolt only. So, every second you see it is picking up a bolt, putting it in place, tightening it up. Again, for the next one also, it is doing the same. Let us say it has to assemble a wheel. Maybe it has 36 number of bolts in a single wheel that is to be tightened for a truck, and you see, it has similar things

which are very monotonous in tasks, very repetitive, and it has to be done quite faster also and sometimes, along with other robots also. So, it requires a good amount of synchronisation. So, that is only possible if robots are programmed. So, that is the reason programming is very, very important. The same is the reason for programming to be there for any task which we do normally with the computers also.

So, the reason over here is similar. It lays the foundation for a successful robot deployment. If programming is capable of implementing quite a good amount of features in it. Let us say it can do circular trajectory with just a single command. It can do a linear trajectory in just a single command. If robots teach pendants, the programming tool can suffice with all those instructions in place, and you can implement them in no time and more precisely, you can do it. Instead of defining a circle using points points points points like that, it is better to have a parametric definition of a circle within the compiler itself, within the robot interpreter itself. So, that it can quickly create the code, we can quickly program it, and we can use it. So, having all those features in place, mostly industrial robots teach pendant they have it. So, that lays the foundation for successful robot deployment.

It determines the capability of the robot. If a robot is capable of being calibrated by just touching an external tip from three different points, find out the tool dimensions the way you saw it earlier in our chapter calibration. So, you saw an external point was placed in the robot workspace, and you were made to touch the robot from different directions, and the robot teach pendant, the controller can automatically identify the dimension of your tool. Similar was the case when you could identify the linear rail transformation matrix. So, that makes the robot quite capable, quite implementable for multiple applications, you see. Rather than having a raw robot which you can program through Python or any programming language of your choice, it is better to have an industrial robot which has all those features inbuilt into it. So, that makes it very, very capable for those specific applications. It drastically affects the deployment time. Having all those features in place, all the commands in place, all the calibration tools in place, everything is already built into your controller programming front end.

So, it affects the deployment time. You see, nowadays, with a flexible manufacturing system, you have very short lead times, very scheduled, very tightly scheduled lead times. You see, you have to change your programming every month, second week, you have to change. So, that is how, because of the type of goods it is manufacturing, it gets changed. It is very much customer-specific. Even on a single assembly line in a Toyota plant, you see there are multiple cars which are getting manufactured and multiple types of cars are getting manufactured in a single assembly line. You see, in those cases, in the case of a flexible manufacturing system, deployment time matters a lot. Your robot should be capable of handling such variations in your assembly line also. So, that is just an application. There are many similar applications if it comes to industrial applications.

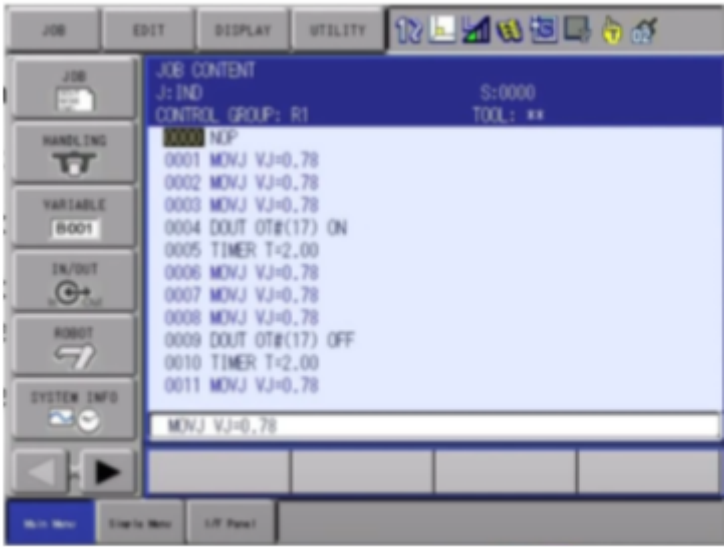
So you see, that is the reason it reduces the deployment time drastically. It enables the implementation of reusable codes. If it is code, there can be various snippets of the code which

are useful for various other purposes. Let's say you are taking data from a sensor, you have a filter which is there in place, and then it is handling something which is quite much maybe having a set of for next loop which handles quite good amount of tasks. So, the part of the code which handles the task and the part of the code which handles the filters are two different modules which can be clubbed with other programs also.

Maybe you have a different set of tasks which is to be performed with the same set of sensors. So, you may require the same filtering code again in some other code. So, this is just an example again where you have two at least reusable codes which can be put into different programs. So, this saves quite a lot amount of time and effort that goes into it, and different modules of a task can be programmed by different programmers.

Each programmer may be a specialist in something else. Let's say in the earlier example also where I gave an example of filtering signal data which is coming from a sensor. A programmer may be very much a specialist in that. He will code for that particular filtering code. Whereas a programmer who is very much expert in programming to do multiple things simultaneously using robot pick and place kind of thing maybe. So, he may be given a task. So, you see, multiple groups of programmers can work together, do things in modules and put them all together to make a set of jobs handy. So, that is the way it is programmed in industry also. It is not always a single programmer who does all the programming for any particular complex application in a system.

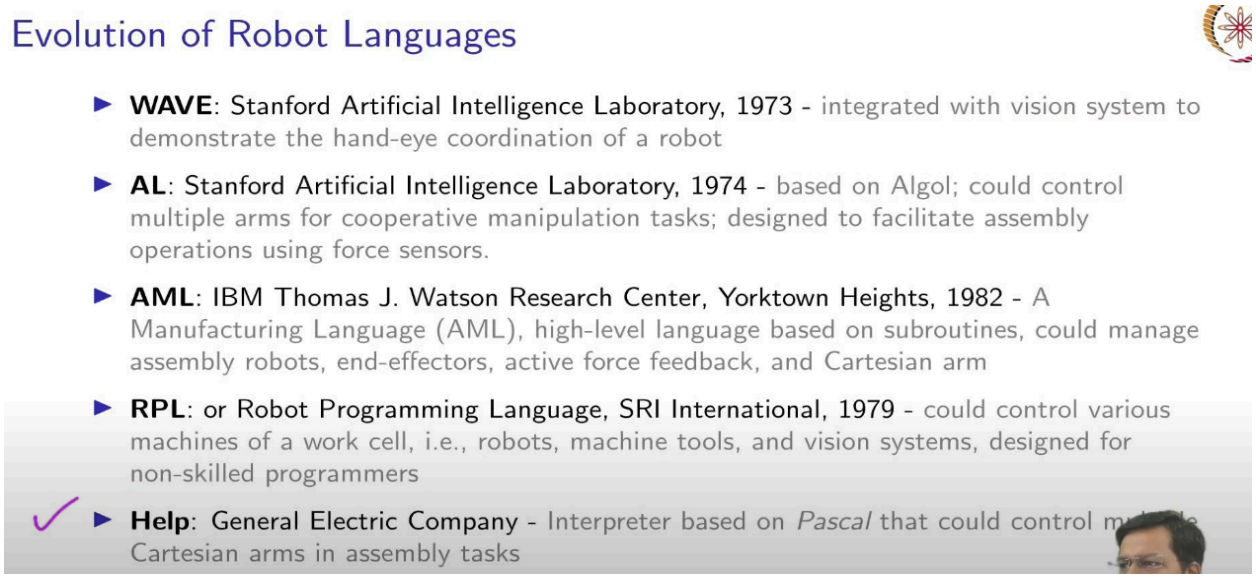
Teach Pendant



Programming Interface

However, the ease of programming with the teach pendant. The teach pendant is a tool which comes with the robot controller. So, that is the teach pendant. It has an integrated development environment IDE for any industrial robot is severely limited due to the complex hardware architecture of the robot controller. So, it is not like a standard PC, which does software programming and keeps it within. It has to be integrated with a lot amount of external devices

also, and those devices are many. And they are manufactured by different makers. So, not like computers, which have to handle just a mouse, keyboard or a printer or a fax machine or something like that. So, in this case, each and every piece of hardware is very, very complicated. They are not updated the way it happens in the software world. Not as in the case of our PC or a laptop. These architectures have been very much constant for quite a long amount of time, and that makes things very, very complex. You see you have multiple number of device all has to be interfaced. Even the architecture of the robot controller is quite complex because it has to do things in real-time. Time is very much crucial here. Not like our laptop or a PC, where handling a particular thing at a specific time or handling a trajectory is not there. Your time can run independently other than the rest of the programs if it is there. That kind of complexity is there in the case of industrial robot controllers. So, that makes this programming tool, that is, the teach pendant, a very, very complex thing and finally, when it comes to programming, that programming cannot be developed and updated every second day. But it is not like our Android phone, where applications are updated quite frequently. These architectures are not updated so frequently, and that is the reason because of their complexity. It is very difficult to design a device driver for all the attachments which these kinds of controllers are handling. So, you see, it is very, very complex, and that is the reason the programming end is also very rugged, and it is quite much difficult to code in that. It is not as flexible as our standard C, C++, and Python programming languages. Although quite a lot of them are built over this but, the flexibility is a kind of very much limited. Just like as soon as you start typing in the case of standard IDE, it will automatically complete the intended commands that you want to key in. Those features are not there in the case of industrial robots, either and mind it, these are made literally very simple so that you don't need a very skilled programmer for industrial robots. So, that is also one of the specific reasons why industrial robot programming is made very simple. And it doesn't have too much flexibility the way we have in our standard integrated development environment.



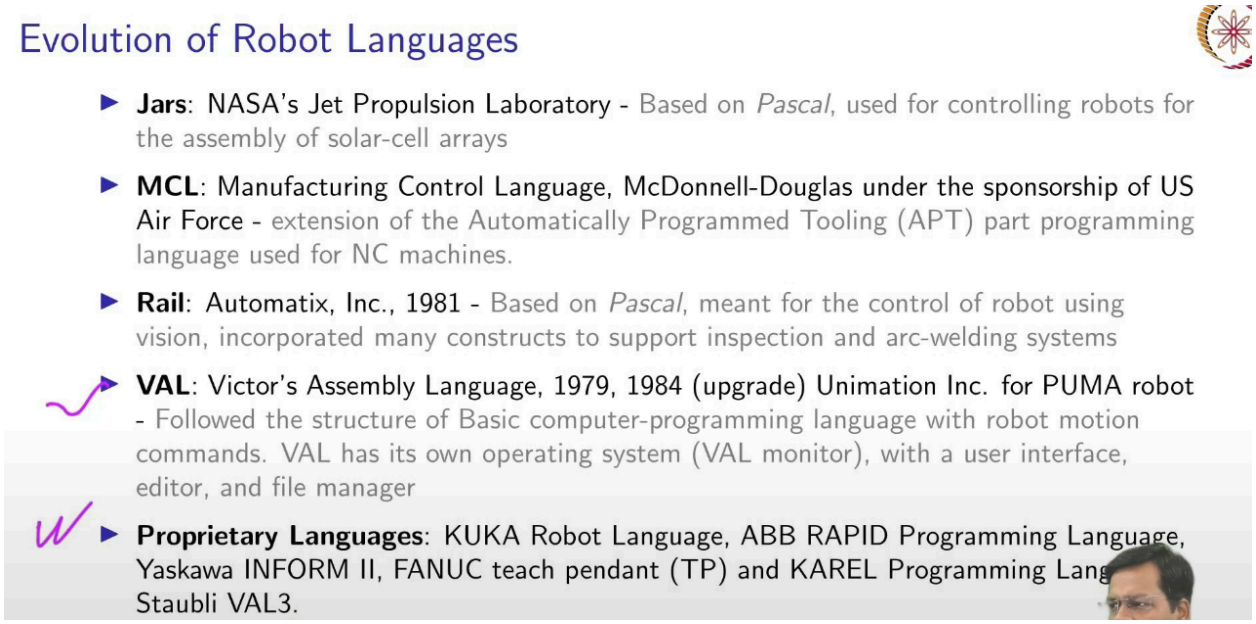
The way industrial robot languages have evolved, we will just go through that now. So, the first one was the WAVE which was developed by Stanford Artificial Intelligence Laboratory in 1973.

They could integrate a vision system to demonstrate the hand-eye coordination of a robot. So, you see, Stanford University has done quite a lot of jobs in robotics in those days when it was just evolving. You see, quite a good amount of robots were evolved there. It was developed there. Finally, it came into industrial systems. So, in those days in 1973, Stanford AI lab was very, very developed, and they could develop such language, the first language that was known as WAVE. Then came a similar one that was based on the Algol programming language in 1974. It was known as AL. It could control multiple arms. It could do cooperative manipulation tasks designed to facilitate assembly operations using force sensors. You see compliant manipulations handling tasks using force sensors. So, those kinds of compliant manipulation tasks were the kind of research which was going on in those days in the Stanford AI lab and that is the reason this language was evolved also. They cannot just do pick-and-place tasks. So, this type of assembly robot was force-compliant. It was not just a standard robot which is position controlled. They were working on robots which were force-controlled torque controlled also. So, that is one big area of industrial robotics, which is now termed collaborative robots and cobots. So, that kind of collaborative robot evolved from here only. The next one is AML, which was developed by IBM, Thomas J. Watson Research Center, Yorktown Heights, 1982- a manufacturing language AML. It is a high-level language which has subroutines which are there in our modern programming also and it could manage assembly robots just like AL and effectors definitely.

Most of the robots that can do that and active force feedback control. So, this is again in line with this and the Cartesian arm. The gantry arm that I showed you earlier showed different kinds of robots and their classification. So, you see, IBM has also come up with AML. It is not just in 1982 they have come up. They earlier had many other languages for their own system. RPL is yet another language, Robot Programming Language, developed by SRI International. It could do various machines of a work cell. Now, the concept of a work cell has come where it can handle robots, machine tools, and vision systems, and it is designed for non-skilled programmers. Now, it is quite much of an attention is given to non-skilled programmers.

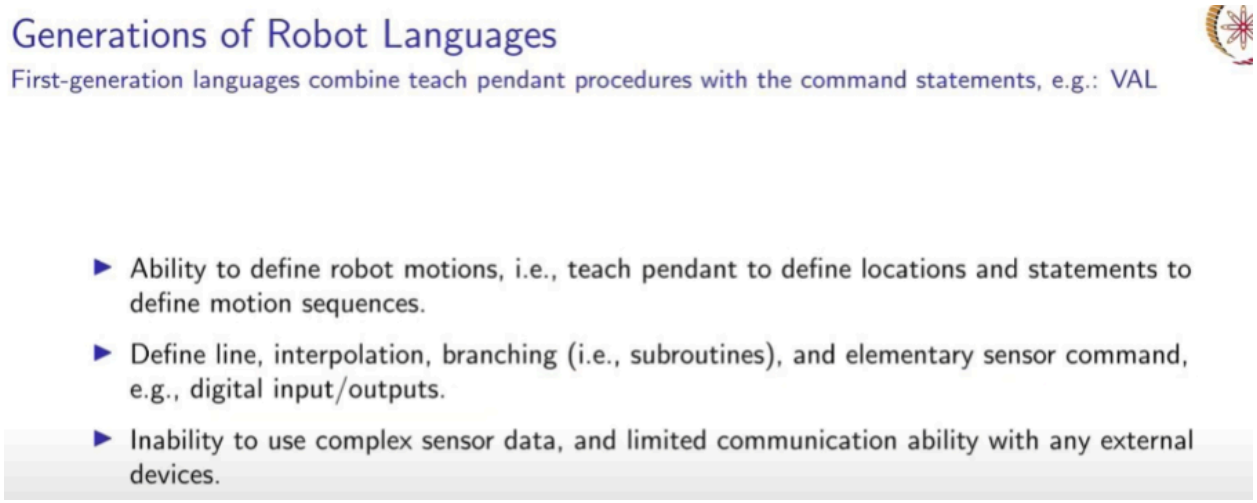
In earlier days, robots were mostly used by skilled programmers. Mostly, who had very good advanced knowledge of industrial robots, programming and the intrinsic of the industrial robot and how it is made. When it came to industry, it was realised it is not going to be programmed all the time by those programmers. It has to be multiplied in numbers for industrial software, and in that case, such programmers were not readily available. So, attention was given to non-skilled programmers also.

Next came the HELP, which was developed by General Electric Company. It was again an interpreter-based language that had evolved over Pascal and could handle multiple Cartesian arms in assembly tasks. So, it is quite complex. It can handle multiple Cartesian arms, and it can do assembly tasks. So, it is quite advanced, you see.



Then came NASA's Jars. So, that was developed at Jet Propulsion Lab over there. This is again based on Pascal. Used for controlling robots for the assembly of solar cell arrays by NASA. So, it is very specific to them. So, it was not much commercialised after that. So, MCL came later on that is Manufacturing Control Language that was developed by McDonnell Douglas under the sponsorship of the US Air Force. This is, again, very specific to them. Extension of Automatically Programmed Tooling, APT program that is nowadays used for NC machines also. So it is just an extension of that. Next came the RAIL that was developed by Automatrix Inc. in 1981 again based on Pascal. You see, Pascal was a foundation for quite a good amount of programming languages which is there. It is meant to control the robot using vision. Vision is again there. Incorporated with many constructs to support inspection. Now, see the specific application add-on was there for the arc welding system also. So, these are some of the advancements which were in line with modern-day compilers or programming IDE that you see. VAL is the next one. Vector's assembly language was in 1979. Later on, an upgrade came in 1984. It was developed by Unimation Inc., which also developed the first robot Unimation and PUMA robot. This followed the structure of basic computer programming language. So, it is not Pascal now with robot motion commands that are always there that are to be there. VAL has its own operating system. Now, the concept came for its own operating system. It is not just the programming front end. It is also the OS which is developed by the robot manufacturer now. It has its own user interface, editor and file manager. So, you see, it is in line with the OS and the complete hardware GUI that you can see in the robot stitch pendant. The whole of that is developed by the robot manufacturer. So, this is one of the most modern one, which is in line with the current-day robot programming language, and then this follows. So, you have proprietary languages which were developed by KUKA. KUKA has its own language known as KRL, KUKA robot language. ABB has its own, which is known as a RAPID Programming Language. Yaskawa has INFORM II and FANUC- they have their own teach pendant with KAREL Programming Language. Staubli uses VAL3 in line with this. So, you see, other than

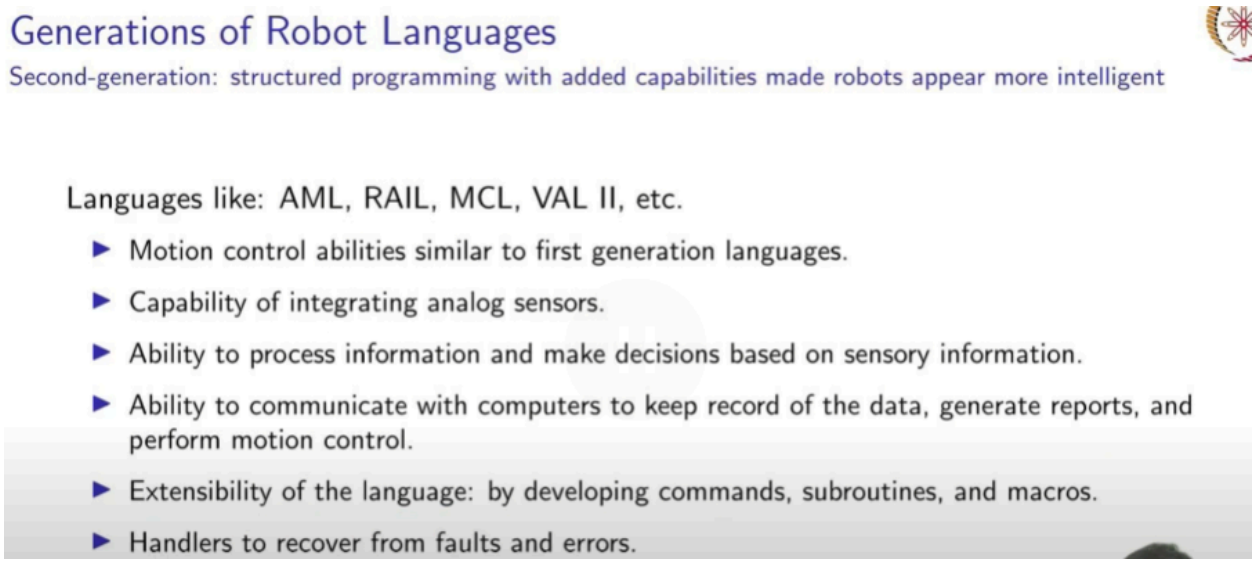
standard programming language which was developed over the years, proprietary languages took over. So, each robot manufacturer came up with their own programming languages. Although the features were very much in line with them, it has its own OS. It has its, maybe, a collaborative real-time engine that runs behind and then it has its own OS or sometimes collaborative OS also they have and then they developed the GUI in front of them. Like in the case of KUKA, it has a real-time engine which runs behind the front-end OS of Windows. And then it has its own GUI which actually does the integration of all these three. So, user interface, editor, file manager, so everything is there in these programming languages also. So, this finally took over. These are proprietary languages, so there is no generic language which applies to all the robots. So, they don't follow the way C++, C sharp, or Python. It runs on all platforms. It is not like that. So, in the case of robots, each robot manufacturer has its own programming language. So, it is very, very difficult to switch a user from one robot to the other one. If he is a specialist in one of the robots, he may not be equivalent in other robots also. Right? It is very, very difficult. So, this has to be evolved in the modern years to come.



So, yes, the generation of programming languages, as it evolved, the first generation language, combines teach-pendant procedures with the command statements, the way VAL has. It has the ability to define robot motions. Motion is the most important stuff which has to be there in any robot programming language. Mostly, it is done using a teach-pendant to define locations. There are jog buttons. I will show you what the teach-pendant looks like also. So, it has a jog button which you can press and take the robot to a place. You can teach the robot that particular location, take it to that location and create some statements to create some kind of motion sequences.

Define line, interpolation, branching, subroutines, and elementary sensor commands to acquire digital and analogue data inputs. Right? Analog was not there with the first-generation language. Only digital input was there. The next is the inability to use complex sensor data. It doesn't have the ability to use complex sensor data. Limited communication ability was there with any

external devices. It cannot communicate much with too many devices which are there in the industry.



So, the second-generation language basically overcomes the shortcomings of the first-generation language. They were more structured programming that was used and added capabilities that made the robot appear quite intelligent. That includes languages like AML, RAIL, MCL, VAL2, etc. Motion control abilities were similar to first-generation languages. The capability to integrate analog sensors has come. Now, you can use analog sensors also, not just the digital on off type of controller like proximity switches. It can also handle analog sensors like force sensors, and pressure sensors like. So, the ability to process information now and make decisions based on the sensor information. So, it has multiple if then else kinds of commands which were there so that you can, based on the inputs you can, control the robot. So, that made the robot appear quite intelligent over here. So, the ability to communicate with computers now. To keep a record of the data, generate reports and perform motion control as well. Motion control is definitely the prerequisite apart from that, it can record the data on a PC which is connected to it. It can generate reports of how many bolts it has tightened to say. So yes, these are some of the capabilities which are there in second generation robots. Robot programming languages. The extensibility of the language is there. This language can evolve. It has provisions to include more commands through subroutines. Macros definition, the way we do it in Python and all. So, those features were also there in the robot programming languages. So, these are very advanced features which make complex two things to be defined in a macro way and then you just call that routine macro routine, and it will just do that job in no time. Instead of writing the whole procedure every time, we can simply call those macros. Just like subroutines. Hope you understand. So, handlers to recover from faults. So, it can also have handlers if any fault occurs, and it can do a set of jobs and stop the robot in case of any serious error. So, those can be defined as well.

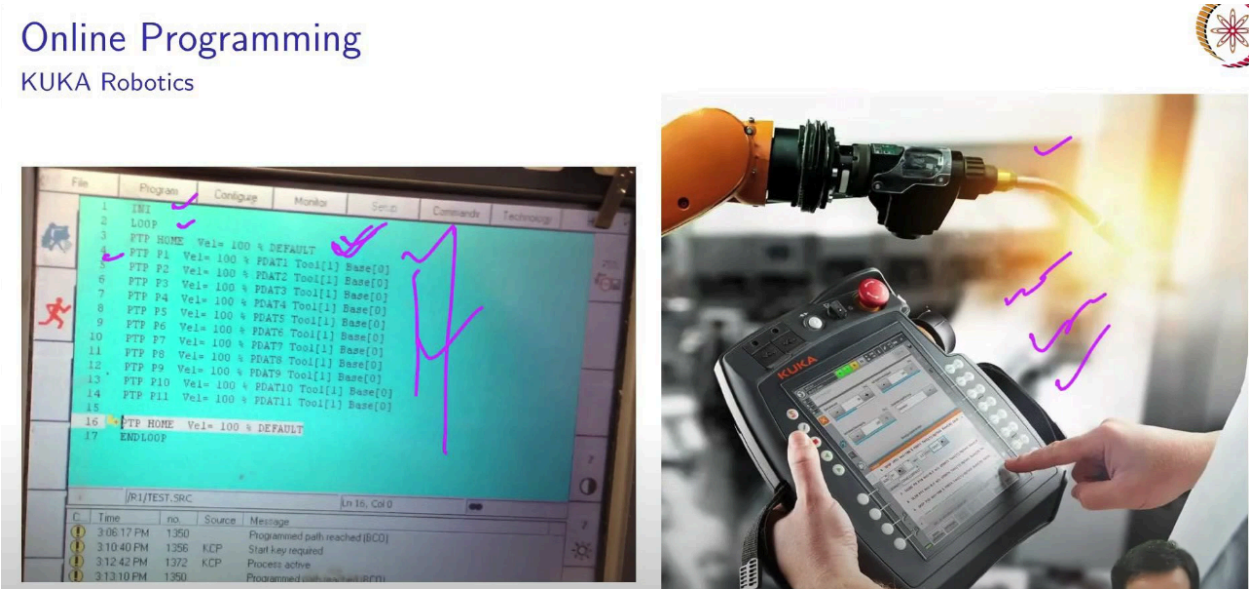
Standard Functions of a Robot Programming Language

- Instructions for Motion Control: Position, Primitive motions (Circle, Linear, Spline) with Speed, etc. in different calibrated frames (tool, world, object, workspace, etc.)
- Instructions for Tool Control: Gripper closing or opening, Spot welding, etc. through dedicated add-ons and I/O channels.
- Standard programming instructions: Looping, decision making, Branching (if..then..else, do..while, for..next, while, switch case, etc.), math functions, string functions, etc.
- Input and Output Instructions: Digital and Analog.
- Communication Instructions: Ethernet and Serial normally, Field-Bus communication is performed using mapped system variables or addresses.
- Advanced functions: PID control, Signal Filter, etc.
- Miscellaneous: controller parameter settings, program selection, fault diagnosis, etc.

So, what should be the standard functions of any standard programming language which deals with robots. So, it should have instructions for motion control that is a prerequisite. It has to do those motions. How is it done? It can command the position. It can command primitive motions like a circle. It can generate lines. It can define spline by using control points and maybe a standard cubic spline can be fitted. So, spline commands are built into this program. Instead of writing the whole routine for the spline, which is really very complex. So, it has an inbuilt feature in which you can define spline also and with different speeds. So, on those paths, it can run at multiple speeds and in different calibrated frames. You see, you have, and if you can calibrate your tool, you can take the tooltip to any trajectory. You know that already. That is the reason why we calibrated the industrial robot earlier in one of the modules. It can be calibrated to run on the world coordinate system, object coordinate system, workspace coordinate system, external circular rotary table you can use. So, those are external frames that you can define and put that tool to move in those particular calibrated frames also. So, that is what is motion control capabilities. That is what is desirable in any robot programming language. So, instructions for tool control also like gripper opening and closing spot welding operation. So, there can be an add-on to standard programming language.

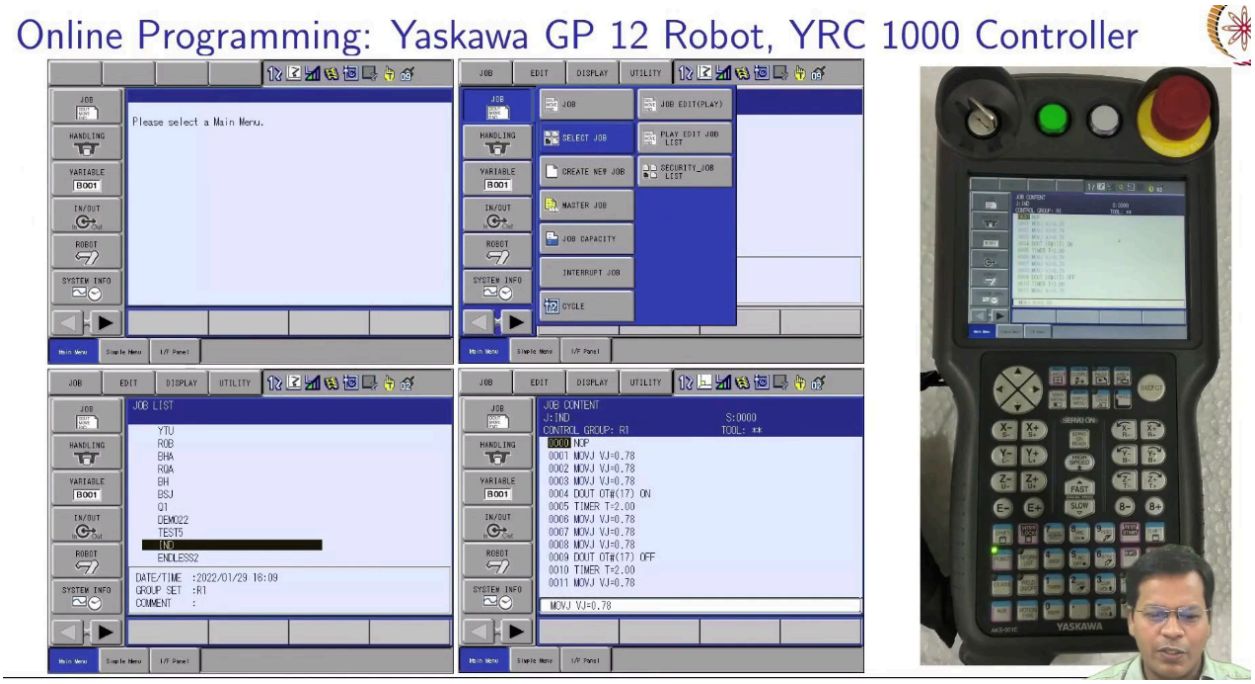
Normally, industrial robots come with a basic thing. Then, depending on the applications, you can add on certain tools like spot welding thing painting thing, right? So, those add-ons can be added to create those features. Even to handle a gripper, sometimes there are gripper add-ons to the standard programming. Through dedicated add-ons and the I O channels also. So, tools can be controlled directly by IO channels or using those add-ons. Standard programming instructions are what we normally feel we should be having in our standard programming IDE. So, looping, decision making, branching, if-then-else kind of statement, do-while, for-next, while, switch case statement. These basically increase the capability of the robot programming front end. Math functions like using sine a cosine functions. String functions when it acquires data from a remote system. It comes like a string, and now you have to handle those strings. String manipulation can

be there in any standard programming language. So, there are standard instructions to do these kinds of kinds of stuff, right? So, input and output instructions like to handle digital and analog inputs if it is there or output if it is there. So, there should be a ready-made command for that. You should not handle straight away from the registers. You should have a map memory address, maybe from where you can collect those data. Communication instructions like to communicate through Ethernet and serial. These two are normally there in an industrial robot. Field-bus communication is performed using mapped system variables or addresses. So, Field-bus like Profibus, Ethercat, and Ethernet, those communications can happen through mapped addresses, okay? Where the data can be put and it simply can go from that address, right? So advanced functions are also there like PID control. These are the most widely used kind of controls which are there in industrial robots. So, that is the reason, based on the input-output relation, you can have a PID kind of relation. It has ready-made commands to handle signal filters. Butterworth first-order filter, to say. There can be a ready-made command where you have parameters to control the kind of filter you want. You can design your own filter based on the parameters. So, these are some of the advanced functions which can be directly embedded in your standard programming language or it may come like an add-on, right? So that is, there and miscellaneous commands which are there to control parameter settings. Let's say if it is a PID; you can have parameters to gain control, to control its gain, right? So, there you have controller parameter setting, program selection, and fault diagnosis. There are miscellaneous commands which are there, functions which are there in the standard robot programming languages.



So, you see, it looks like this. Standard programming front-end is normally done on teach pendant. It is very, very inconvenient. It has to be held in hand, and you have a touch screen sometimes. Sometimes, you may attach an external keyboard, or this display can be replicated to any external bigger display with a keyboard so that you can do programming the way you normally do on your standard PC, but that is not always doable in the case of industrial software. You have to do programming over here only. Because you have limited things to do, you cannot

go and start programming genetic algorithm kind of complex thing over here. You don't need it actually in the industry. For research this is not a platform to do research. There are other ways to bring out research using these kinds of robots. I will discuss that in the few slides to come. So, it looks very much like this. You see the programming. This is from KUKA of GUI, programming front-end. It has, let's say, has INI function. It has loop PTP point-to-point ohm 100% velocity to use, right? And then you have PTP to P1 location velocity 100%. Data for that is stored in p.1. There are two files, basically. One is the SRC file, which is the source file, which you can see here and there is a file that is behind this. That has all the data corresponding to the points that you have recorded, right? p.1, p.2 like that. So, each robot has its own way of storing this data with respect to tool 1. That is, if there are multiple tools. You can call one of the tools as tool 1 or tool 2. They are stored in the array. So, over here, it is tool 1 that you want to take to point p1, and then you have the base, which is defined here. So, these are some of the calibrated tools and the base set in which you want to operate. So, this is just an example I have shown here. So, this is how it goes. You can take the robot to different locations using the keys which are here in the teach pendant.



So, even in the case of the Yashikawa robot. You see, it has a similar front end, which looks like this. You have Z plus, Z minus, Y plus, Y minus. Right? X plus, X minus, like that. So, these are jogging keys, similar to roll pitch and yaw. You can move them. Right? This is to jog all these. There are menu-holding buttons also. With which you can handle the cursor. Right? So, this is the front GUI. You see it is very much similar to KUKA programming. MOV with velocity and then you are handling on and off certain input. Right? So it looks very much similar and this is the front menu you have gone to select the particular job. Selected one of the jobs. That job is open here. Got it? So this is the kind of menu I have just shown. This is with the Yashikawa

GP12 robot with a YRC 1000 controller. So, basically the GUI depends on the type of controller it is there. Not on the type of robot which is connected to the controller. So, mostly, this teach pendant is a generic teach pendant for all Yoshikawa robots, which can be connected to the controller YRC 1000. So, there are other teach pendants also with other compatible controllers. So, this is online programming. So, online programming means a robot is connected live. You have to stop the robot and start programming.

Offline Programming

Programming while the robot is already operating, possibly in an environment different to the intended program

The robot is programmed in a virtual environment created using CAD or 3D scanned data.

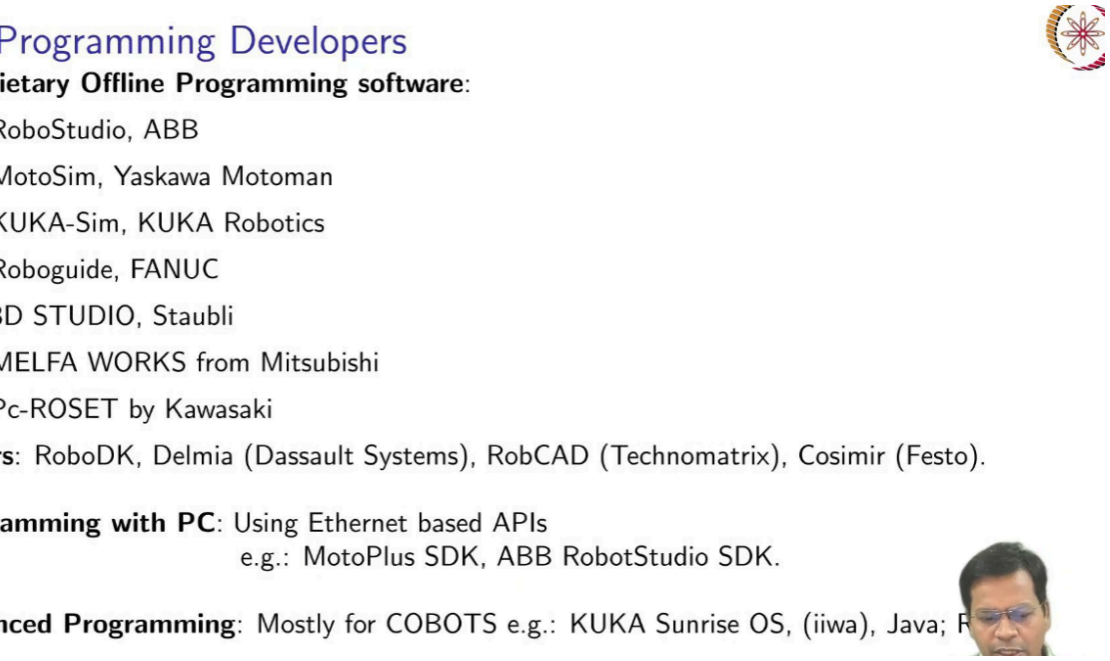
- Planning the system and robot operation quickly and conveniently: Allows optimizing workspace layouts by rerunning the **simulations**.
- Easy to **present** the animated solutions to the customers thereby, increasing the success in sales and the revenue.
- Easy to design the work cell concepts in advance with reachability and collision tests checks, precise cycle times (micro-motion, idle states) for increased planning reliability.
- Allows **virtual commissioning** (Sensors, External axis, PLC connectivity) to verify the cell processes to increase the planning reliability and **safety**.
- **Modular**: The existing work cell arrangement can be integrated with different applications, like arc welding, spot welding, punching, etc.
- Offline program generation for different robot-controller combinations, **Optimizing energy** consumption.

Note: Needs expert programmers, and additional investment.

So, there are other ways to do programming also which is known as offline programming. So, programming while the robot is already operating. You can program the robot while the robot is in use somewhere else. You can do the programming. You can dump that code to the robot when you are ready with the code. So, it is done in an environment which is different to the intended program. The program may be there for some other application with some different set of environments which is there. Whereas this robot is currently mounted in a different environment. Your program may be for a different environment. So, this is quite possible. So, the robot is programmed in a virtual environment created using the CAD model or 3D scan data. So, quite a lot of offline programming tools they support 3D scan data also. So, what are the capabilities which are there and what you can achieve with that? So, let us discuss that. So, planning the system and robot operation quickly and conveniently. So, you see, this is very, very convenient. While a robot is already in use, you cannot program it. So, you have to wait for that much time until your robot is off and you can program it. So, you are saving time here. If you can do programming while the robot is already in use you are saving time. You are doing programming in your own way. You are convenient mostly to do programming on the standard PC. So, it is very, very convenient to program on your PC rather than on the teachpendant. So, complex programs can be generated. This allows for optimising different workspace layouts by running

the simulation. So, the front end of this offline programming system allows you to simulate the robot. See virtually how my robot will behave. What could be the best fit for everything? Where you have to locate the object, where you have to locate the toolset. Right? So, everything can be optimised over here. By re-running those, you can see the one which has the best-optimised system of all the robot environments. You can directly deploy that to the robot. So, it is easy to present. Now, these front ends also allow you to create animated files like mp4 files or a zip file animated solution. This helps you to present how you are going to make your robot work on the industry soft floor. Like, so in that case you can show this animation to your customer. They will be impressed by looking at it. Right? They can appreciate things better rather than explaining things in the air. So, in that case, this increases the success in sales and, finally, the revenue of a company. So, it is easy to design the work cell concepts. In this case, you have multiple robots that can work together. You have multiple robots. Here each one of them can be made with a different program. They can run with a different program, and in advance, you can make them do something. In advance, you can analyse its reachability. Collision tests can be done when all the robots will work, whether they will collide or not. You can easily check offline only. So, it saves quite a lot of hazards. So, the environment will be saved, and money will be saved. If at all, you can prevent any accident from happening. So, this is very, very useful. You can make a micro-motion study. When it is picked from here, it will put it there. How much time will it take, and how long the system is in an idle state? So, those can be optimised over here. So, it has increased planning reliability also. It can take, do precise things with less cycle times also. So, that is the benefit of having offline programming. The next is virtual commissioning. In this case, you don't just program the robot. You need it to interface with maybe a real hardware somewhere. PLC connectivity may be with external access. With external sensors where the robot is virtual, the rest of the things can be real. So, those interfaces can be there. Or even these can be there in soft. You don't even need that. Right? You don't even need the PLC. You have the whole set in virtual, and then you can generate your code and finally put it to use. To verify the cell processes. To increase planning reliability again and the safety also. So, this is very, very important. And it is very much modular in nature. So, existing work cell arrangements can be integrated with different applications. A robot with the same kind of table with the same robot and a controller. So, you can have different things for this robot to do now. Different tasks for this robot to do. It can be put for arc welding applications, spot welding, and punching. So, whatever that comes in front of it, it can handle that. Provided you can program it offline and dump them. So, actually, how it happens. You first program the way you program it, instead of taking the robot to a physical location, teaching it. You just define a point as a parameter and later take the actual robot to those locations and teach that location. Got it? So, you already have a code, a logic set, algorithm in place. And then you actually take those robots, jog them to the real location. Teach those points. What you have put in your algorithm. So, this is how it works. So, that modularity is there. You can take a part of this code and put it somewhere else. The way we can do this is with multiple programmers doing a single, developing single program. Right? So, offline program generation for different robot controller combinations. The same program

can be put in for different robots with different controllers. Optimising energy consumption. So, while doing so you can even have tools that can optimise the energy which this robot will take up when it is actually put to you. So, in the long run, it saves quite a lot of revenue. So yes, there are definitely some short-comes for this also. It needs some expert programmers and additional investment is required to buy these tools also. Offline programming tools are very, very expensive for the industry. So, yes, but with this, you can save a lot of money and time. Definitely, programming is a day-to-day job, but it is not very frequently done. So, this can be used. So, you can have an expert programmer here who can do one program. This system runs for a while or a few months and then you can again call him back and do the next program. So, in that case, it can be done.



So, there are various offline programming developers also. Apart from proprietary ones, which are mentioned here. ABB has its own roboStudio. Yaskava, it is known as MotoSim. Kuka, Kuka-Sim. FANUC has Roboguide, and Staubli they have 3D STUDIO. Mitsubishi has MELFA WORKS. Kawasaki has Pc-ROSET. So, various proprietary developers are there. They basically make the simulation system for their own robots. Mostly, you will have to be very much in line with these only instead of open source and those tools. In industry it is not practiced because of safety reasons. Although you have many other open-source developers, it is not an open source. Here, I mean this. This can handle multiple robots of different makes. RobotDK can handle ABB robots, Fanuc robots, and many other robots. Whereas KUKA-Sim can only handle KUKA robots. So, yes, it must be quite obvious. These are inexpensive, and they can handle multiple robots. But you cannot rely on them 100%. That is the reason, the industry they don't go by these. They normally prefer using the simulator, which is designed by the robot manufacturer themselves. Due to the safety reasons and the amount of cost which is involved in creating those industry systems. Delmia by Dassault Systems, RobCAD by Technomatrix, and Cosimir by

Festo. So, there are many open developers also. We have programming with PC features and also using APIs. They are supported by many manufacturers robot manufacturers. So, they have API, which can be called on any standard programming language like Python, C, C++, Visual Studio and those kinds of environments. So, you can call these APIs and they have device drivers which can communicate with your standard robots industrial robots as well. This again, is not preferred in the industry. Due to again safety reasons and real time issues also. There are a few examples, like MotoPlus SDK. It is by Yaskawa, ABB RobotStudio SDK. This is by AB Robotics. So, you see, there are a few which work like this also and definitely there are advanced programming also. Mostly used for COBOTS. COBOTS have a programming language, which is very, very similar to standard programming language. They have APIs inbuilt into the programming they have. Like KUKA iiwa. It has its own Sunrise operating system which is very much like Java. And definitely, you have ROS, a Robot Operating System. You must have heard of it. That is hardly used in any industrial soft flow. Because they are mostly for research purposes. So, somebody who is interested in robots, using industrial robots for your research. You can definitely pick up a few of these. But in industry, these are used. So, hope you got the idea of how industrial robots can be programmed. Different ways it can be programmed. What features normally should be there? How has it evolved?

So, I think with this we can end today's lecture. In the next class, we will just discuss concluding the whole course and today, I will finish this lecture and the course also. That's all. Thanks a lot for being with me. Thank you for watching.