



Funded by the
European Union

MODERNIZATION OF MECHATRONICS AND ROBOTICS FOR BACHELOR DEGREE IN UZBEKISTAN
THROUGH INNOVATIVE IDEAS AND DIGITAL TECHNOLOGY
609564-EPP-1-2019-1-EL-EPPKA2-CBHE-JP



Robot control systems

Doc. Dr. Andrius Dzedzickis

Tashkent, Uzbekistan, 15-19 May 2023



This project has been funded with support from the European Commission under the Erasmus+ Programme. The content of this presentation reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- **Programming languages**
- **Robot programming**
 - Methods**
 - Online programming**
 - Offline programming**



Robot Programming



Each type of robot uses its own programming language. For example:

ABB robots are programmed in the RAPID programming language;

Fanuc robots are programmed in the KAREL programming language;

MOTOMAN robots are programmed in the INFORM programming language



Motion programming. Introduction

ABB RAPID:

```
MoveAbsJ jphome,vmax,fine,GR01_W1_Tool\Wobj:=B01_02FX02;
```

```
MoveJ LHP005,vmax,z200,GR01_W1_Tool\Wobj:=B01_02FX02;
```

```
MoveJ LHP010,vmax,z200,GR01_W1_Tool\Wobj:=B01_02FX02;
```

```
! Drop position
```

```
MoveL LBH_02fx02,v500,fine,GR01_W1_Tool\Wobj:=B01_02FX02;
```

```
! Some logic instructions
```

ABB Rapid programming language motion commands



MoveJ - joint movement of separate axes;

MoveL - motion in a straight line;

MoveC - spatial arc movement;

SearchL - line motion searching.;

SearchC - arc motion searching.

ABB MoveJ command



J = Joint - single-axis joint movement; TCP (Tool Center Point) trajectory is not defined; the MoveJ instruction is used;

MoveJ p1, vmax, z30, tool1;

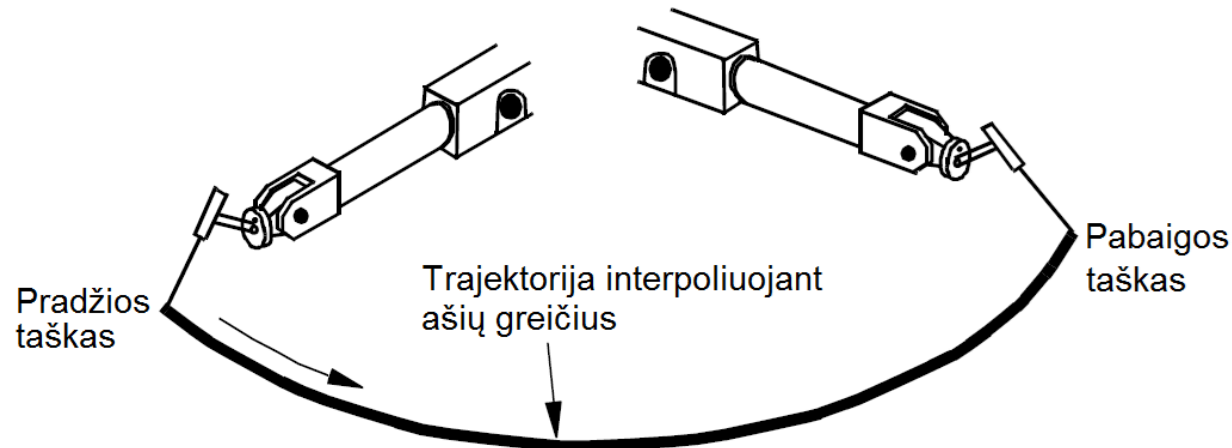
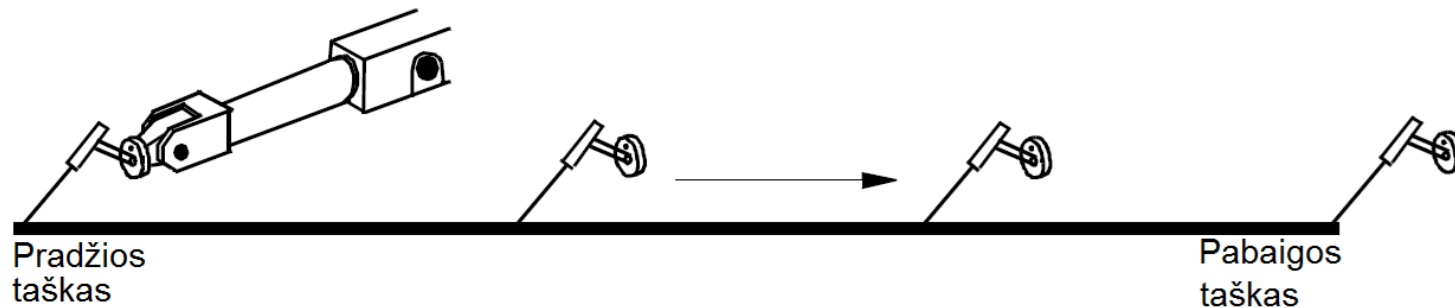


ABB MoveL command



L=Linear – - robot TCP moves straight from point to point (linear interpolation), the MoveL instruction is used

MoveL p1, v100, fine, tool1;

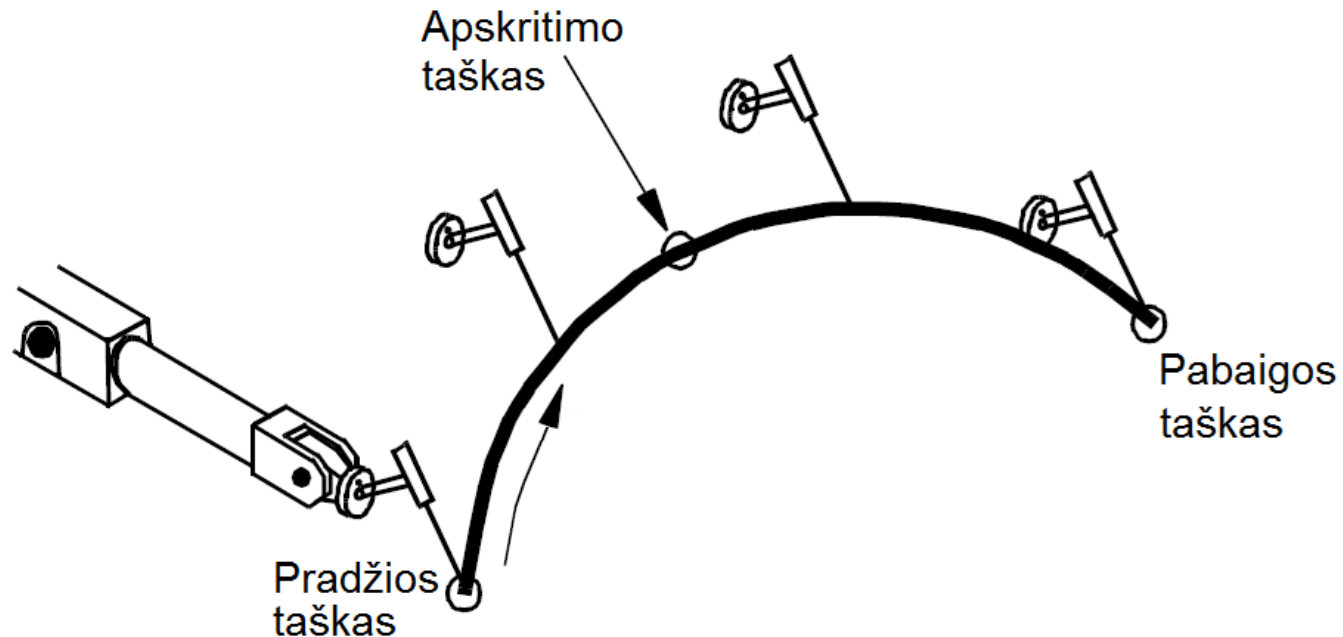


MoveL p120, v200, fine, tool1;
MoveL p130, v200, fine, tool1;
MoveL p140, v200, fine, tool1;

ABB MoveC command

C=Circular – Robot TCP moves in circular trajectory, (circular interpolation), use the MoveC instruction. Two arc points p1 and p2 are specified.

MoveC p1, p2, v100, z20, tool1;



MoveC p320, p330, v100, fine, tool1;



ABB SearchL command



Searching in straight line SearchL - The robot TCP moves from the current point to the specified point in a straight line (linear interpolation) until the sensor becomes active. During operation, the current coordinates are defined and assigned to the variable.

```
MoveL p10, v100, fine, tool1;  
SearchL \Stop, sen1, sp, p20, v100, tool1;
```

In an example, robot TCP moves from p10 to p20.

If the **sen1** signal does not appear, then it returns to p10 and replays the motion.

If the **sen1** signal is generated, the new TCP coordinates are currently assigned to the sp variable, and the robot movement stops (argument \ Stop).



ABB SearchC command

Search by moving in the arc SearchC - The robot TCP moves from the current point to the specified point by circular trajectory (circular interpolation) until the sensor becomes active. During operation, the current coordinates are defined and assigned to the variable.

MoveL p10, v100, fine, tool1;

SearchC \Stop, sen1, sp, cirpoint, p20, v100, tool1;



ABB motion command arguments.

Motion commands **MoveJ**, **MoveL**, **MoveC**, **SearchL** and **SearchC** have a few arguments:

pxxx – (**robtargt** - **Position data**), the coordinate name of the point at which the robot must move

vxxx – the velocity of motion to the specified point;

zxxx – the radius of smoothing when moving to another point;

toolxx – the name of the tool coordinate system.

Command syntax then:

MoveL pxxx, vxxx, zxxx, toolxx;



ABB pxxx point

Move, Search command argument **pxxx** is **robtarg** type variable, which consists from 4 components:

CONST robtarget p15 := [[trans], [rot], [robconf], [extax]];

- trans** – TCP point x, y, z coordinates mm;
- rot** – TCP orientation are defined using 4 angles (q1, q2, q3, q4);
- robconf** – robot axis configuration (cf1, cf4, cf6 or cfx), first, fourth and sixth axis orientation. If 0-90° angle, when=0, if 90-180° when 1;
- extax** – position of external axes in mm or degrees.



ABB pxxx point

Move, Search command argument **pxxx** is **robtarg** type variable, which consists from 4 components:

CONST robtarget p15 := [[trans], [rot], [robconf], [extax]];

For example:

**CONST robtarget p15 := [[600, 500, 225.3],
[1, 0, 0, 0],
[1, 1, 0, 0],
[11, 12.3, 9E9, 9E9, 9E9, 9E9]];**

External axes 3-6 are not defined

ABB pxxx point numbers

The coordinates of the points in which the motion instructions perform a motion can be entered into the **robtarget** type variable. By default, the names of these types of variables start with a letter **p** followed by a dot number. For example **p123**, **p2001**.



```
MoveJ p123, vmax, z30, tool1;  
MoveL p2001, v100, fine, tool1;  
MoveC p174, p2, v100, z20, tool1;
```

Programming with teach pendant numbers are automatically assigned. After that, these numbers can be changed if necessary

ABB points without numbers*



If other motion instructions do not use the point coordinates, then the sign * in the movement command can be used * and the coordinates will be remembered in the instruction. This method is used programing with teach pendant.

```
MoveL *1, v100\T:=5, fine, tool1;  
MoveC *, *, v10\T:=5, z20, tool1;
```

ABB the speed of movement, vxxx



The motion command argument **vxxx** specifies the speed of the selected trajectory. You can select variables whose numbers represent the speed in mm/s (**v5**, **v10**, **v20**, **v30**,**v6000**, **v7000**):

MoveL p10, v1000, fine, tool1;

ABB the exact speed of the movement, \V:=xxx



With the additional argument \ V, it is possible to specify the exact desired speed. For example:

MoveL p10, v1000\V:=1150, fine, tool1;

ABB motion time, \T:=xxx



The command \T can specify the time in seconds for the movement. Then the selected speed is only indicative.

MoveL p15, v1000\T:=5, z20, tool1;

ABB independent movement, \Conc



The next line of the robotic program will only be executed when the specified point is reached.

If you want the robot program to continue running until the robot moves to the specified point, the **\Conc** argument can be used. Then, with the start of the movement, the robot does not wait for the end of the movement and performs another command:

MoveL \Conc p10, v1000, fine, tool1;

There can be up to 5 movement commands with this argument.



ABB motion smoothing, zxxx

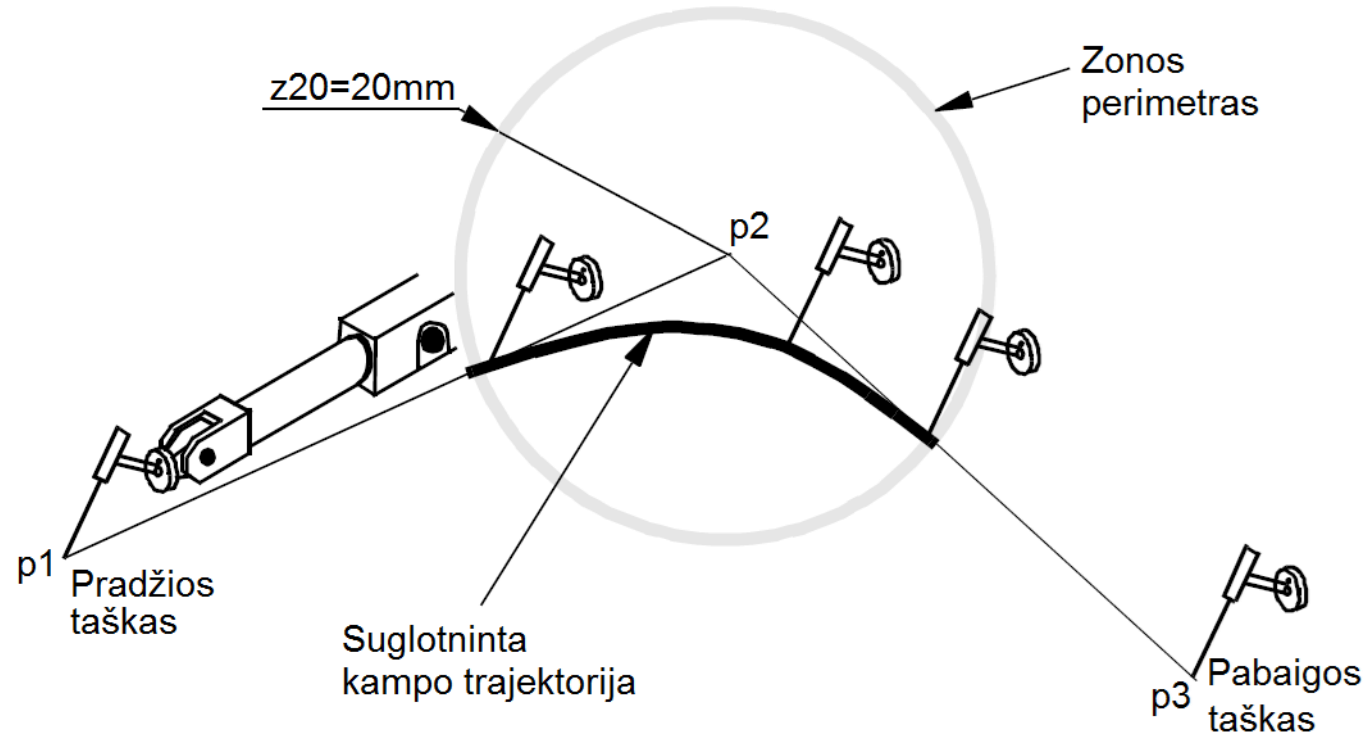
The motion command argument **zxxx** specifies the trajectory smoothing. If in the place **zxxx** is **fine**, then the robot's TCP moves from point to point and stops at each point.

MoveL p10, v100, fine, tool1;

The size of the rounding is selected by a variable whose number indicates the rounding in millimeters: **z1, z5, z10, z15, z20, z30, z40, z50, z60, z80, z100, z150, z200**. For example, 20 mm smoothing:

MoveL p15, v100, z20, tool1;

ABB motion smoothing, fine and zone



MoveL p1, v500, fine, tool1;
MoveL p2, v500, z20, tool1;
MoveL p3, v500, fine, tool1;



ABB motion smoothing, fine and zone

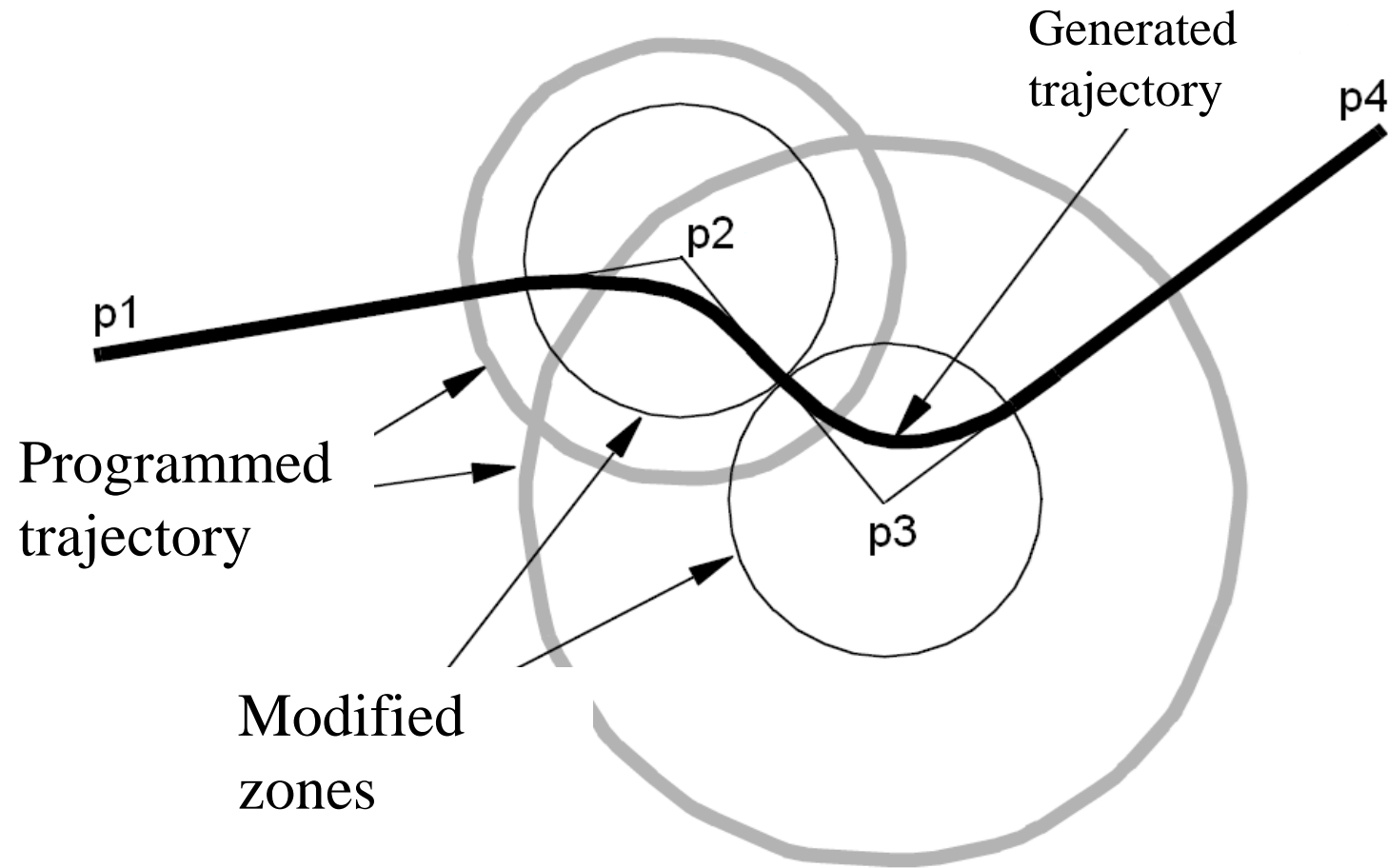
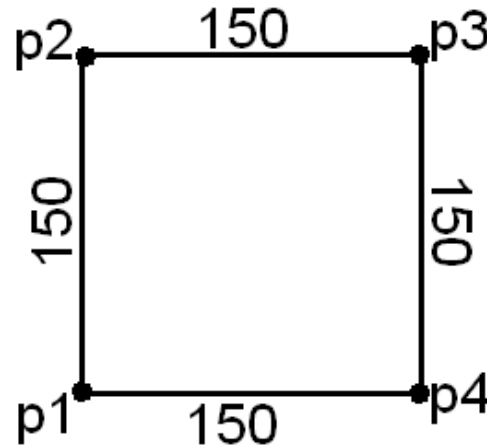


ABB motion with increments

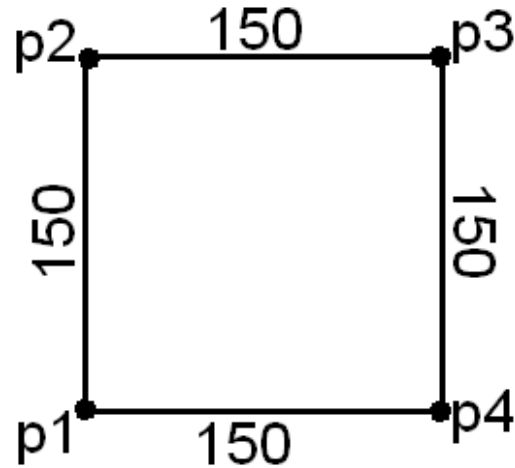
It is often necessary to know only the absolute coordinates of the first point (palletizing, painting, gluing or welding). The coordinates of other points may be indicated by increments relative to the first point. Draw a square showing all the points individually:



```
MoveL p1, v250, fine, tool1;  
MoveL p2, v250, fine, tool1;  
MoveL p3, v250, fine, tool1;  
MoveL p4, v250, fine, tool1;  
MoveL p1, v250, fine, tool1;
```



ABB motion with increments, MoveL Offs



```

MoveL p1, v250, fine, tool1;
MoveL Offs(p1, 0, 150, 0), v250, fine, tool1;
MoveL Offs(p1, 150, 150, 0), v250, fine, tool1;
MoveL Offs(p1, 150, 0, 0), v250, fine, tool1;
MoveL p1, v250, fine, tool1;
  
```



ABB motion with increments, MoveL Offs

The motion with increments instructions syntax:

MoveL Off(px, dx, dy, dz), vspeed, zone, tool

Instructions format for increment:

dx, dy, dz increments according point **px**



ABB motion with increments, MoveL Offs

Increments for motion command can be defined by numeric value, or can be obtained by calculations.

Example:

```
VAR num ilgis := 150;  
VAR num dx := 0;  
VAR num dy := 150;  
VAR num dz := 0;  
...  
dx := ilgis*ix;  
...  
MoveL p1, v250, fine, tool1;  
MoveL Offs(p1, 0, dy, dz), v250, fine, tool1;  
MoveL Offs(p1, dx, 0, dz), v250, fine, tool1;  
MoveL Offs(p1, 0, -dy, dz), v250, fine, tool1;
```

ABB motion with increments MoveL Offs

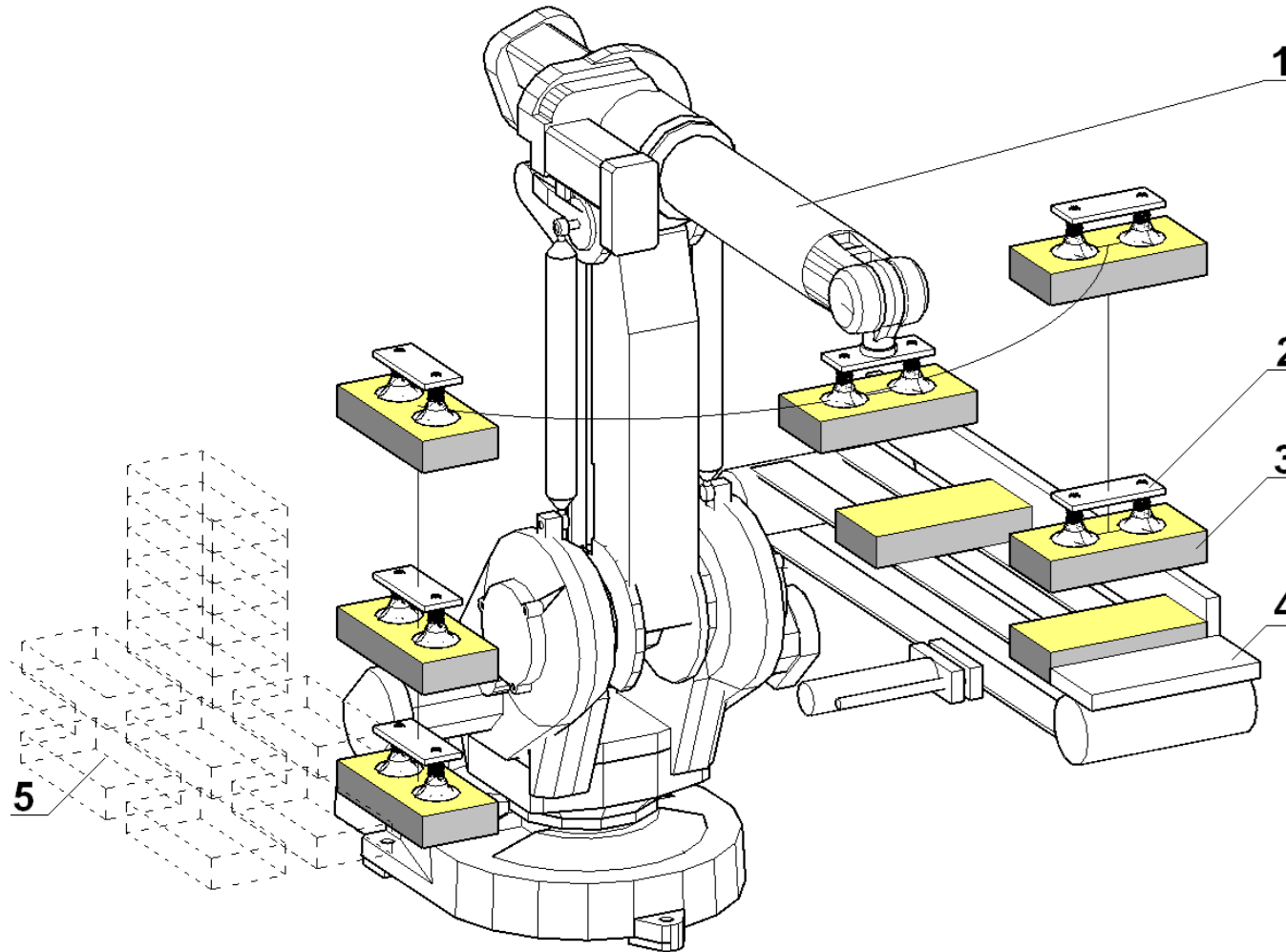
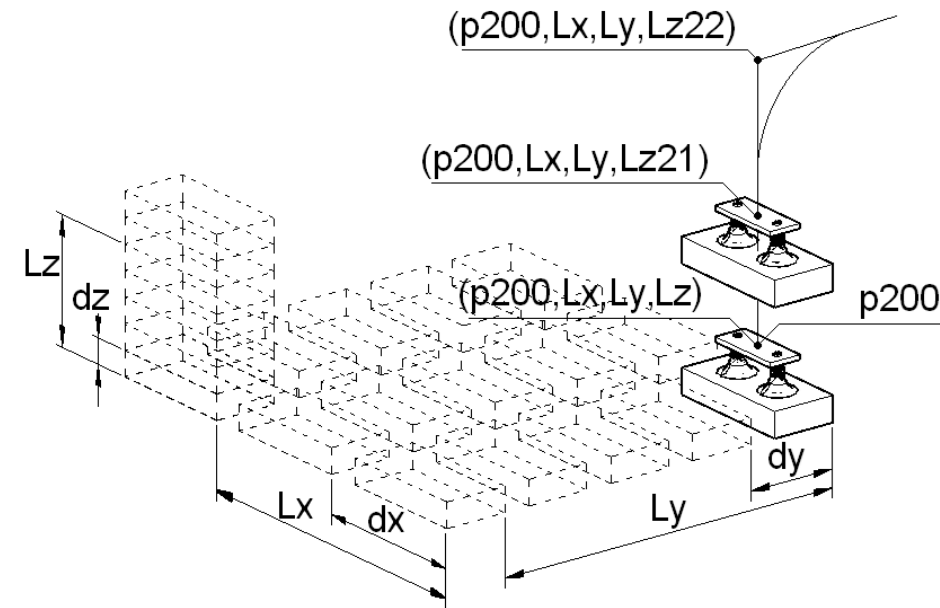
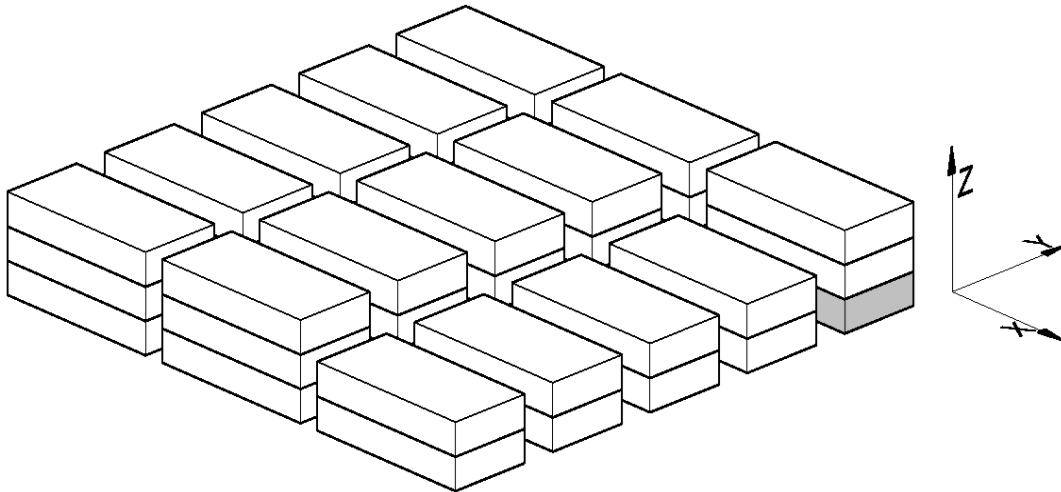


ABB motion with increments, MoveL Offs

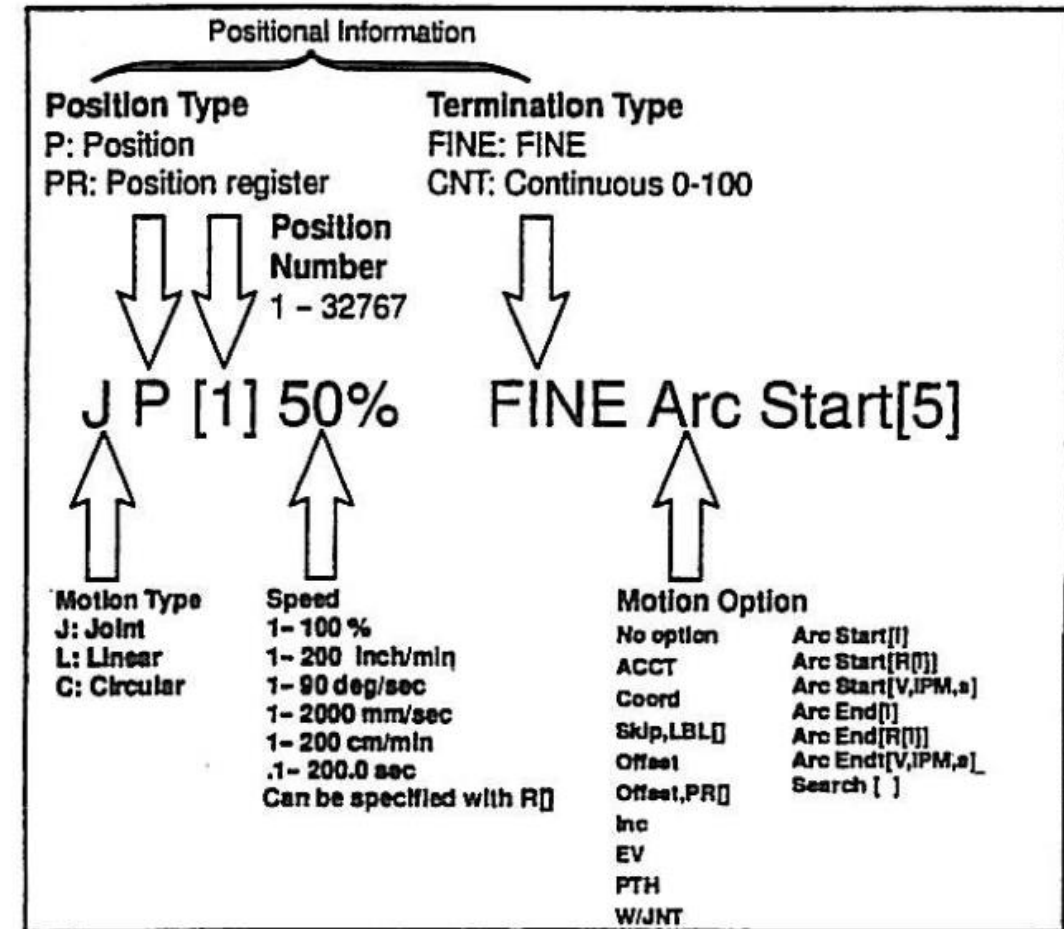


Three-dimensional palletizing

FANUC motion programming



The motion control instruction format is as follows:

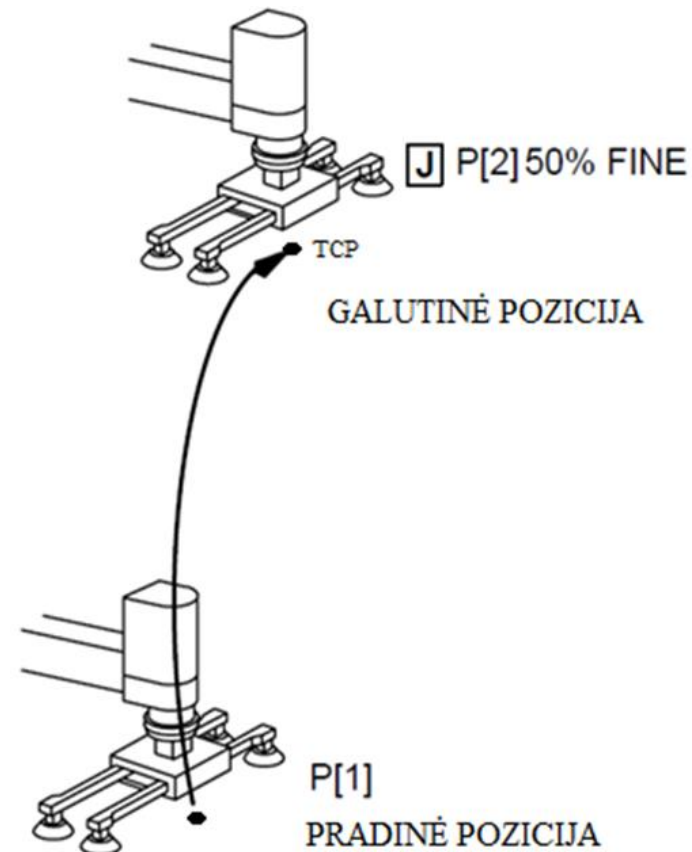


FANUC. Axes Joint movement

All axes move together so that the movement is completed all at once. The shape of the trajectory is unpredictable.



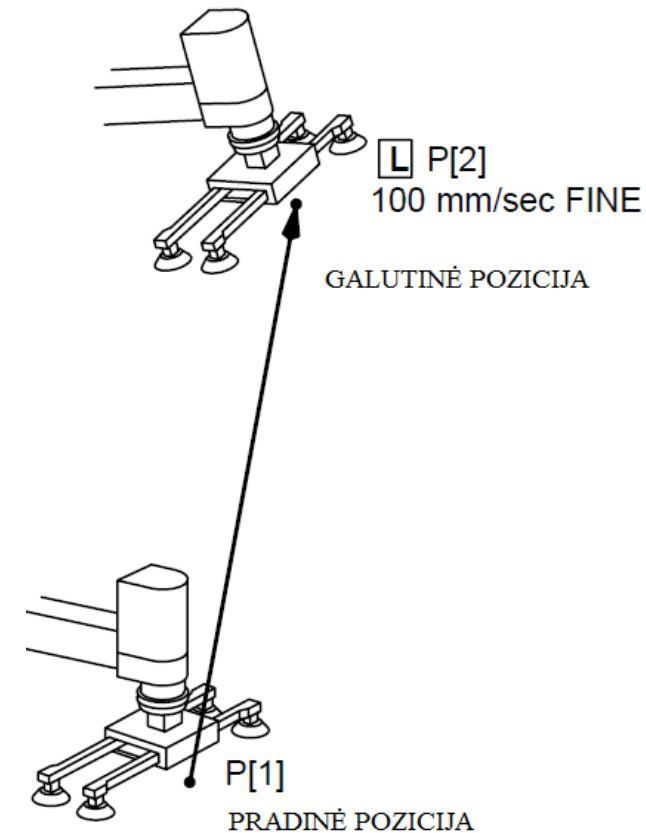
J P[2] 50% FINE;



FANUC, Linear movement

From the point to the point TCP moves
in straight line.

L P[2] 100 mm/sec FINE;

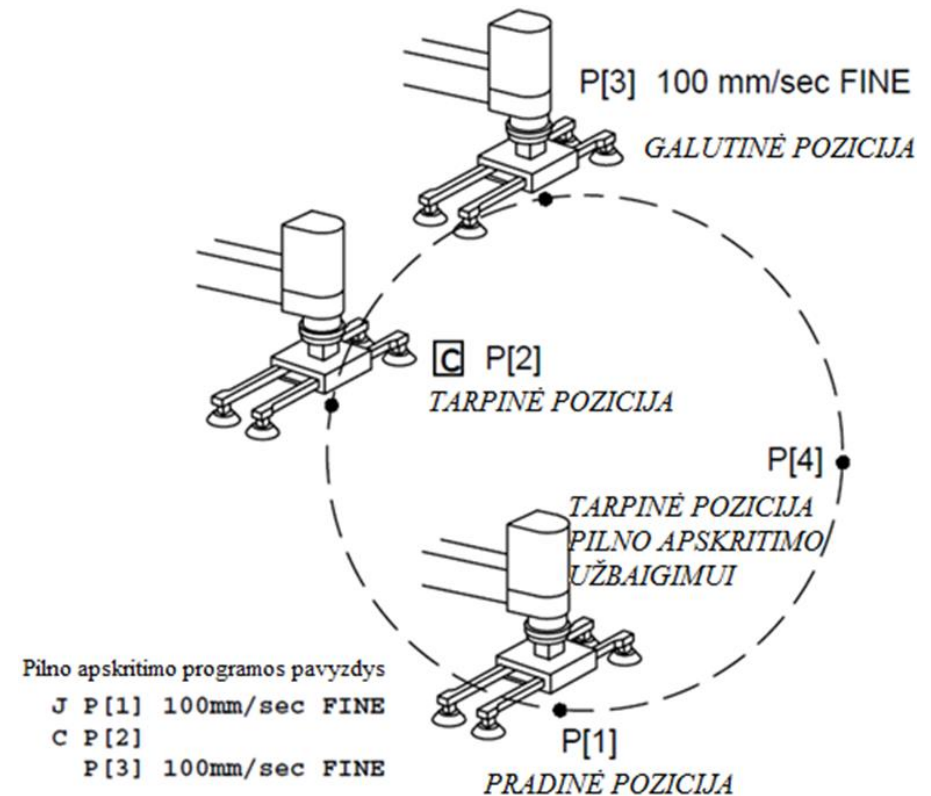




FANUC. Circular movement

TCP moves the arc trajectory from the starting point through two distinctly specified points. A complete circle is obtained by moving in two arcs.

J P[1] 100 mm/sec FINE
C P[2]
P[3] 100 mm/sec FINE
C P[4]
P[1] 100 mm/sec FINE

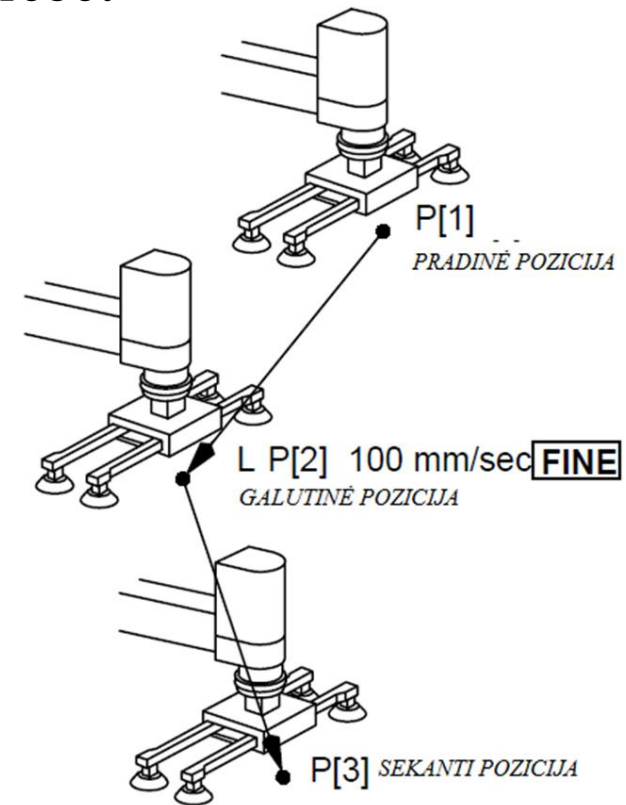


FANUC Motion type FINE

TCP motion can be of three types. During a **FINE** movement, the robot moves from point to point, stopping at each point.



J P[2] 100 mm/sec FINE

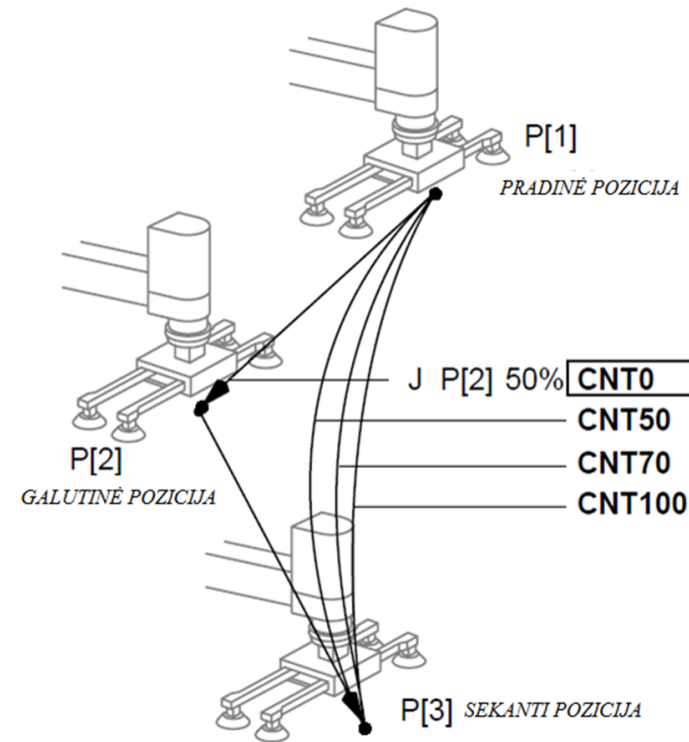


FANUC Motion type Continuous

TCP motion is smoothed using **Continuous** motion type. The **CNT** argument specifies the size of the smoothing - from 0 to 100.



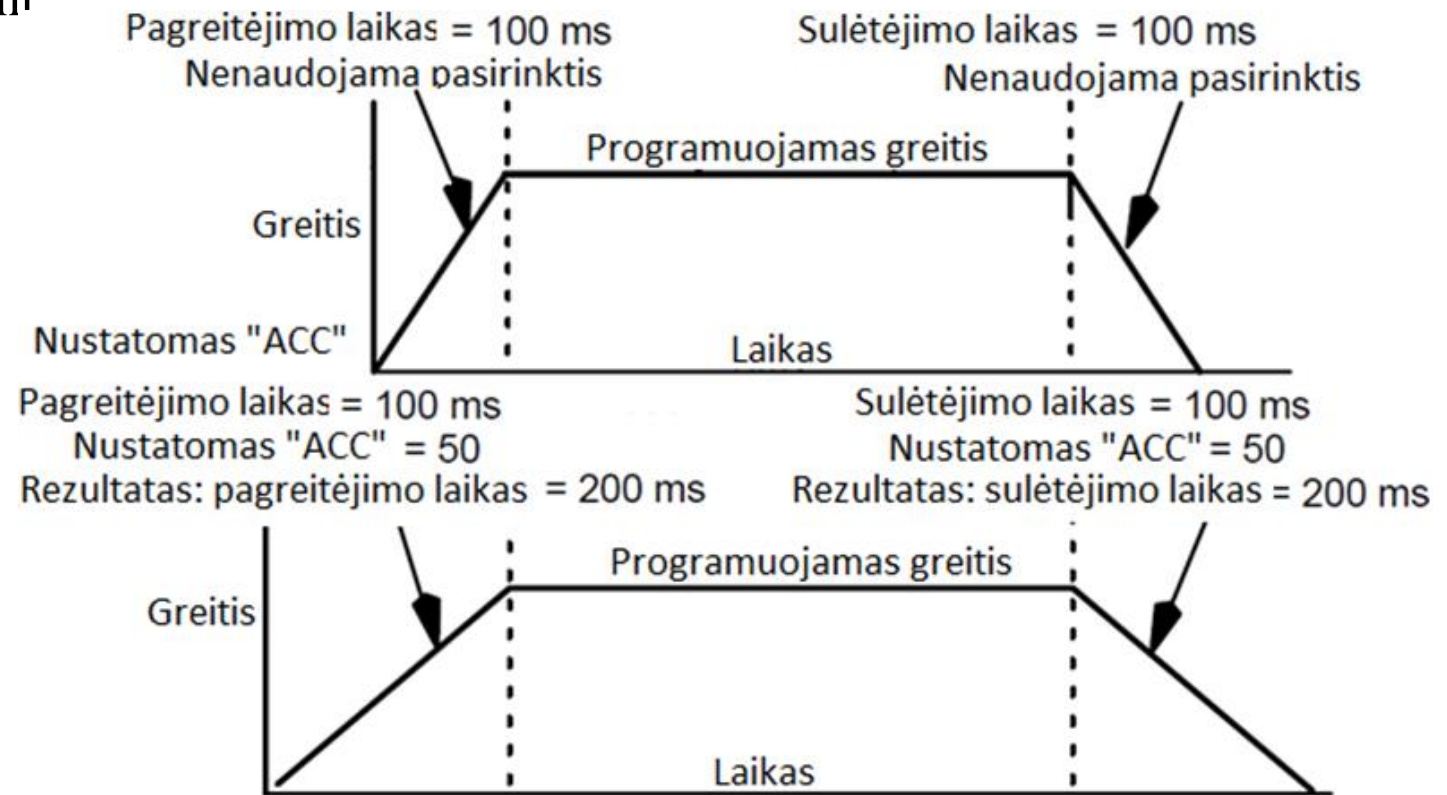
J P[2] 50% CNT70





FANUC motion acceleration, ACC

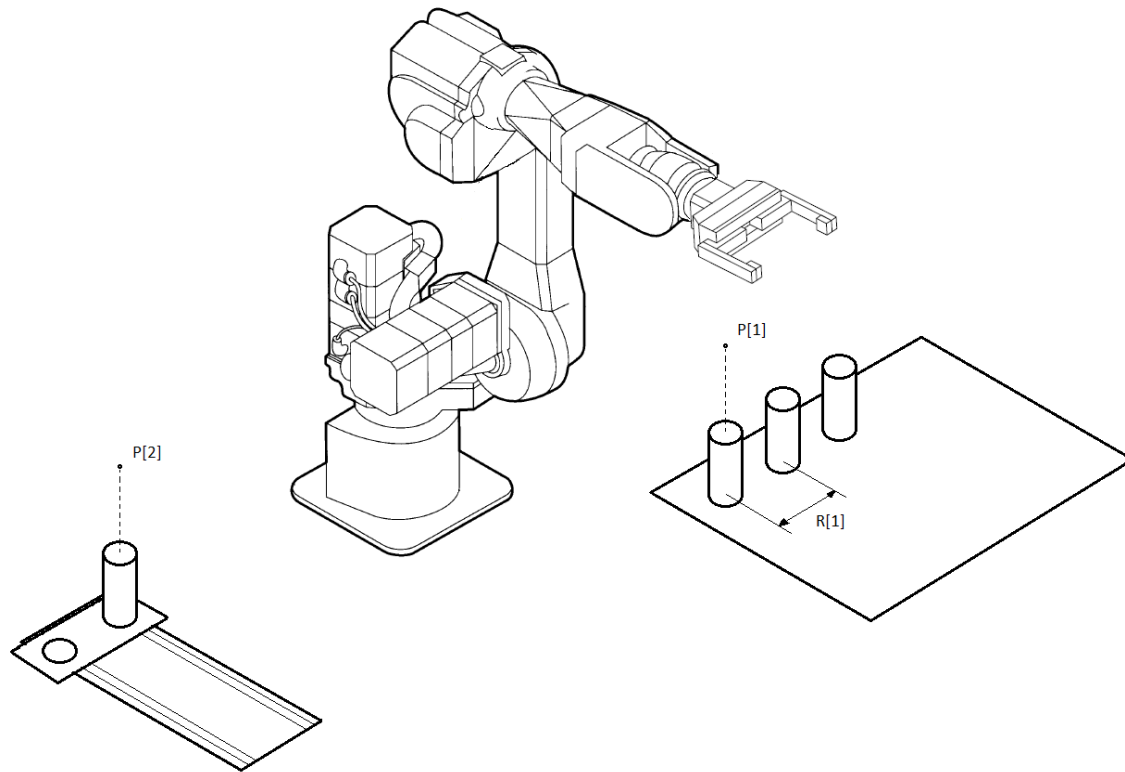
TCP motion acceleration may be limited by the **ACC** argument. The argument value can vary in range from 20 to 500%.



J P[2] 50% FINE ACC50



FANUC one dimensional palletizing



```

1:  PR[1,1]= 1075;
2:  R[1]=100;
3:  R[3]=0;
4:  R[6]=5;
3:  LBL[1];
4:  IF R[3]=R[6],JMP LBL[2];
6:  R[5]=R[1]*R[3];
7:  PR[1,2]=(-686.289)+R[5];
8:  R[3]=R[3]+1;
9:L  PR[1]  500mm/sec CNT0;
10: JMP LBL[1];
11: LBL[2]
    
```

MOTOMAN motion commands



MOVJ - all axes move in tune so that the movement is completed all at once. The TCP trajectory is unpredictable;

MOVL - from point to point TCP moves straight;

MOVC - TCP moves the arc from the starting point through two distinctly specified points. A full circle is obtained by moving in two arcs;

MOVS - TCP trajectory smoothed by spline;

IMOV - TCP moves straight the specified distance, not to a point.

MOTOMAN MOVJ command



The commands of the movement have a whole range of arguments. For example, **VJ** is the movement speed when the **MOVJ** command is performed:

MOVJ P000 VJ=Joint speed

PL=Position level

ACC=Acceleration adjustment ratio

DEC=Deceleration adjustment ratio

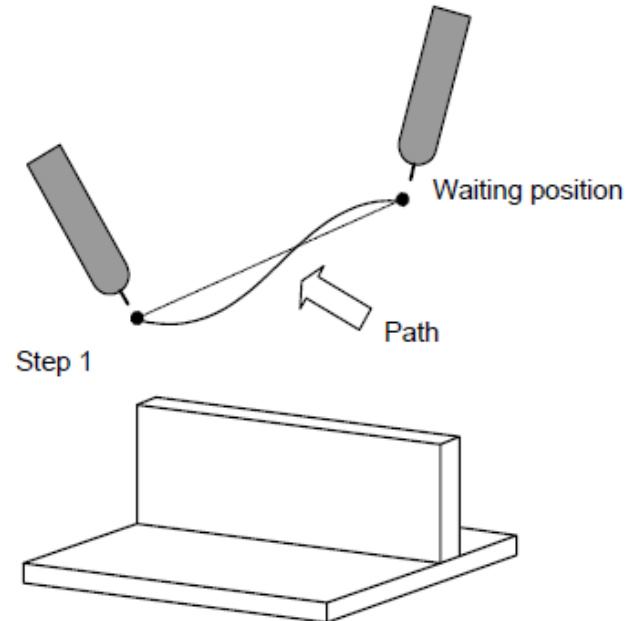
MOTOMAN MOVJ command

MOVJ P000 VJ=50.00

Move from the manipulator's waiting position to step 1. Move by joint interpolation at a speed of 50%.

The position in Step 1 is registered to the P variable no. 0.

The path during movement is not specified. Be careful of interference.



MOTOMAN MOVL command



Command **MOVL** has speed argument **V**:

MOVL P000 V=Joint speed

PL=Position level

ACC=Acceleration adjustment ratio

DEC=Deceleration adjustment ratio

MOTOMAN motion commands MOVJ, MOVL

NOP

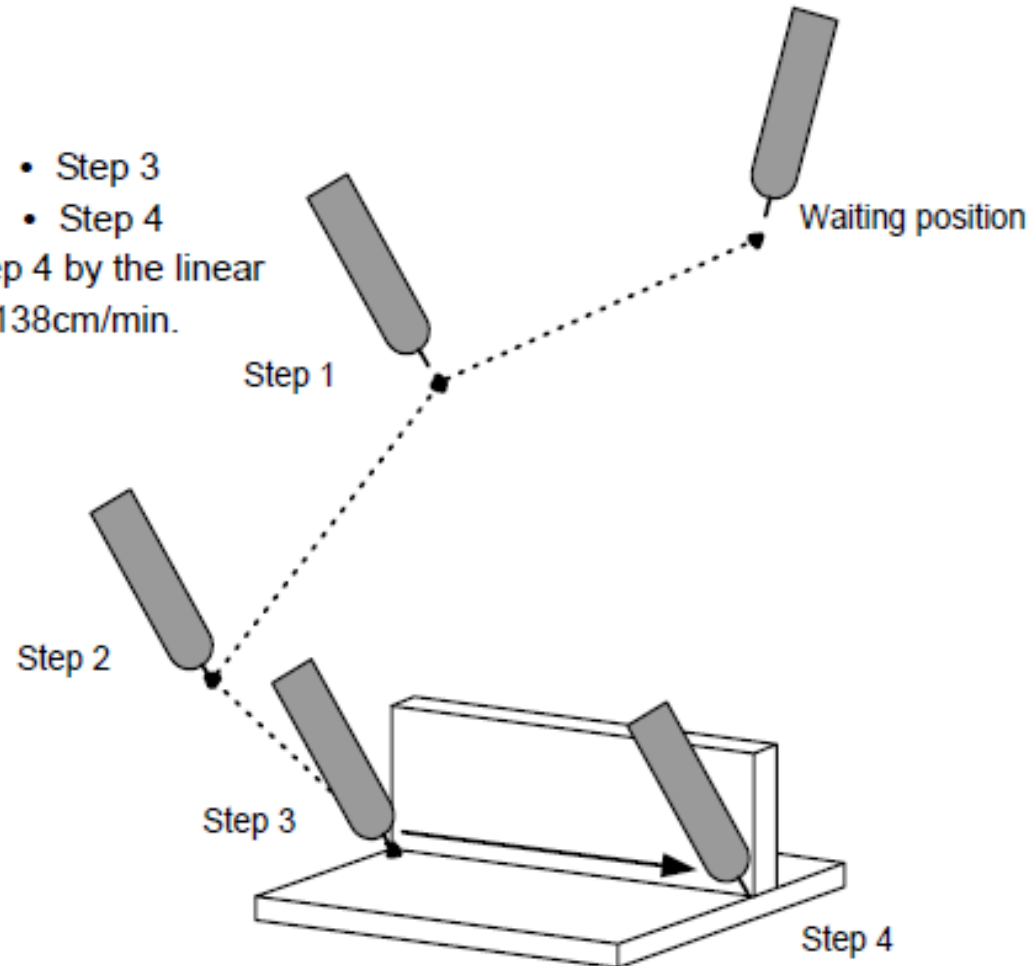
MOVJ VJ=50.00

MOVJ VJ=25.00

MOVJ VJ=12.50 . . . Step 3

MOVL V=138 . . . Step 4

Moves from Step 3 to Step 4 by the linear
interpolation at a rate of 138cm/min.



MOTOMAN MOVC motion command



```

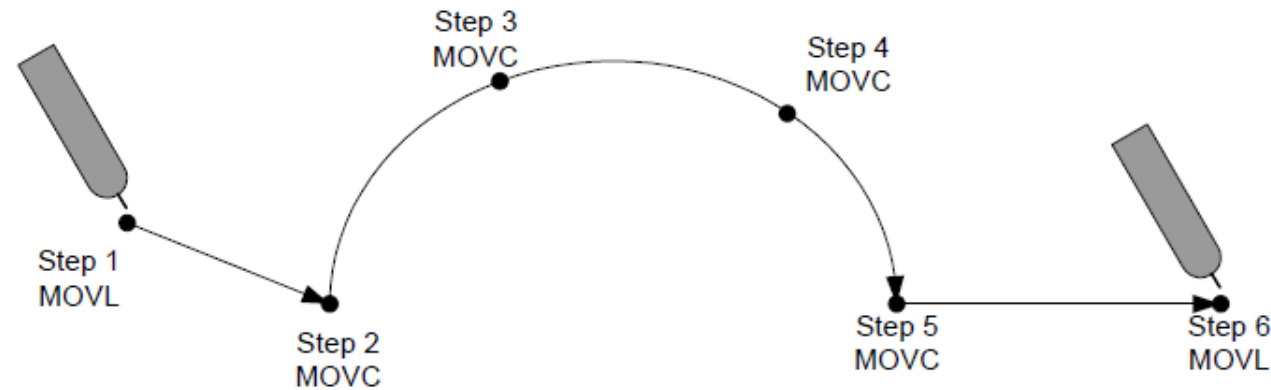
NOP
MOVL V=138
MOVC V=138 • • • Step 2
MOVC V=138 • • • Step 3
MOVC V=138 • • • Step 4
MOVC V=138 • • • Step 5
MOVL V=138
END
    
```

Moves from Step 2 to Step 5 by circular interpolation at a rate of 138 cm/min.

Moves to Step 3 in a circular arc formed with the teaching points in Steps 2, 3, and 4.

Moves to Step 4 in a circular arc formed with the teaching points in Steps 3, 4, and 5.

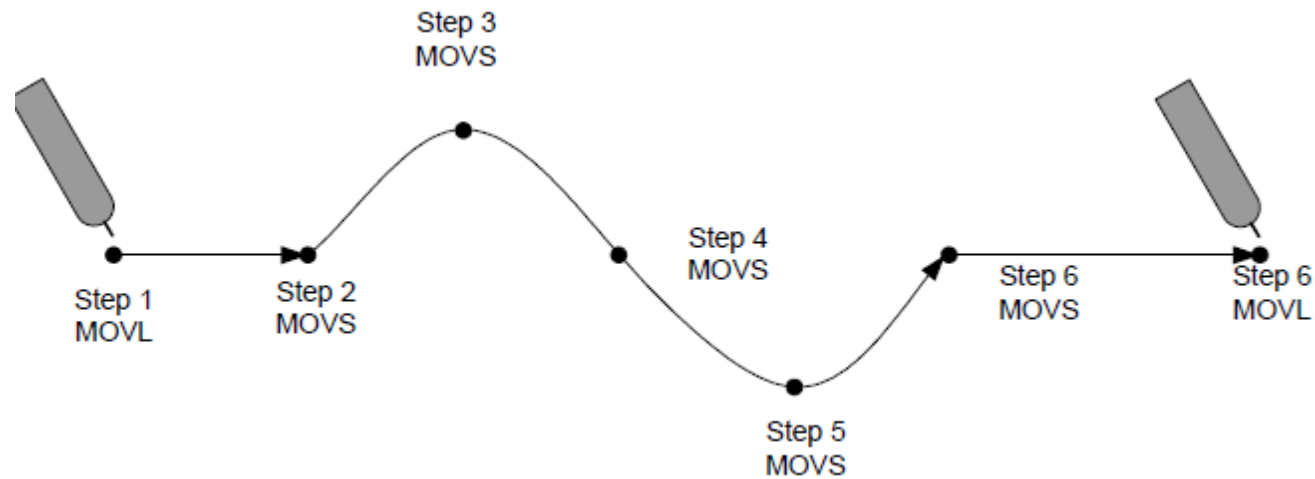
Moves to Step 5 in a circular arc formed with the teaching points in Steps 3, 4, and 5.



MOTOMAN MOVS motion command

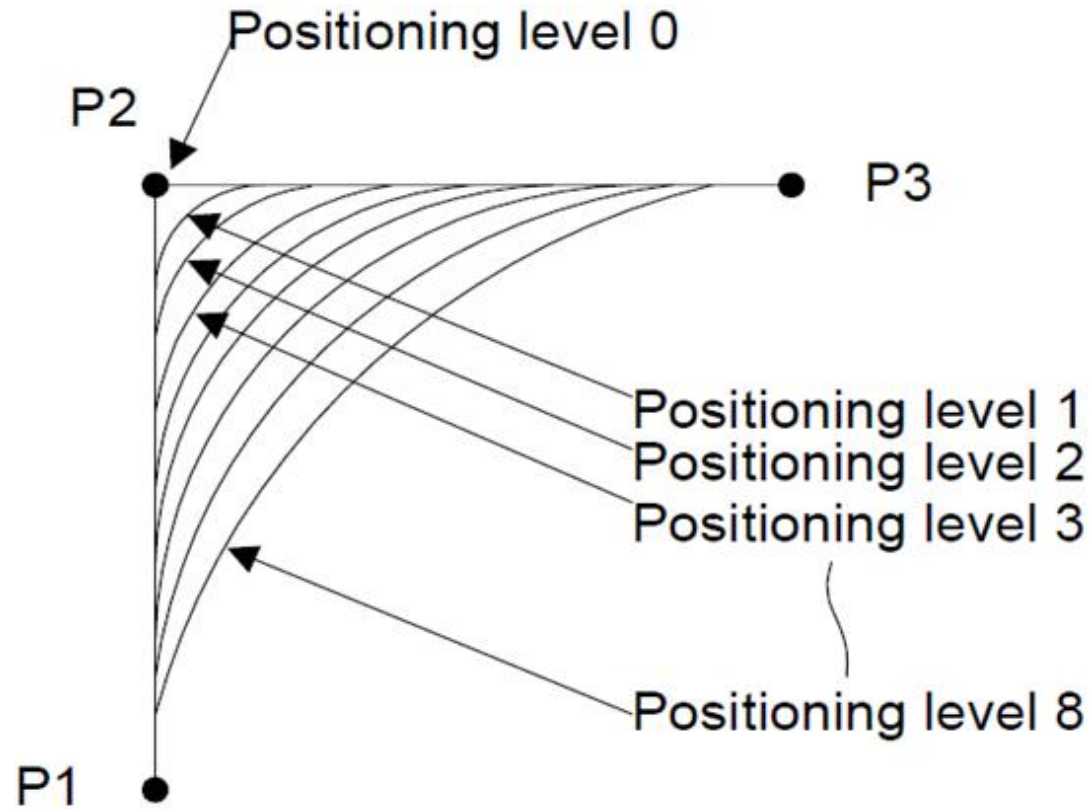
```

NOP
MOVL V=138
MOVS V=138 • • • Step 2
MOVS V=138 • • • Step 3
MOVS V=138 • • • Step 4
MOVS V=138 • • • Step 5
MOVS V=138 • • • Step 6
MOVL V=138
END
    
```



MOTOMAN motion smoothing, PL

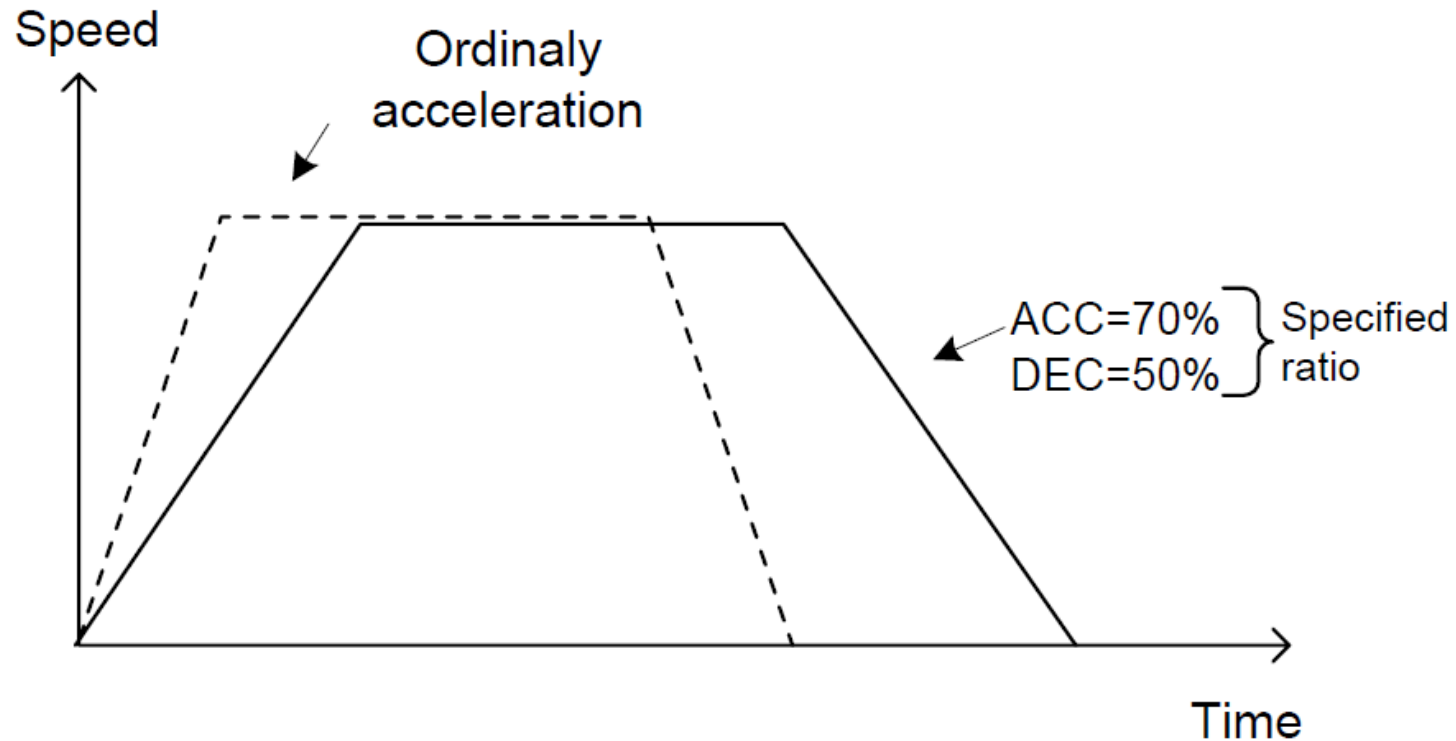
Smoothing argument PL has 8 values. When $PL=0$, smoothing is 0.



MOTOMAN motion acceleration, ACC and DEC



Acceleration are relative, from 20% to 100%.

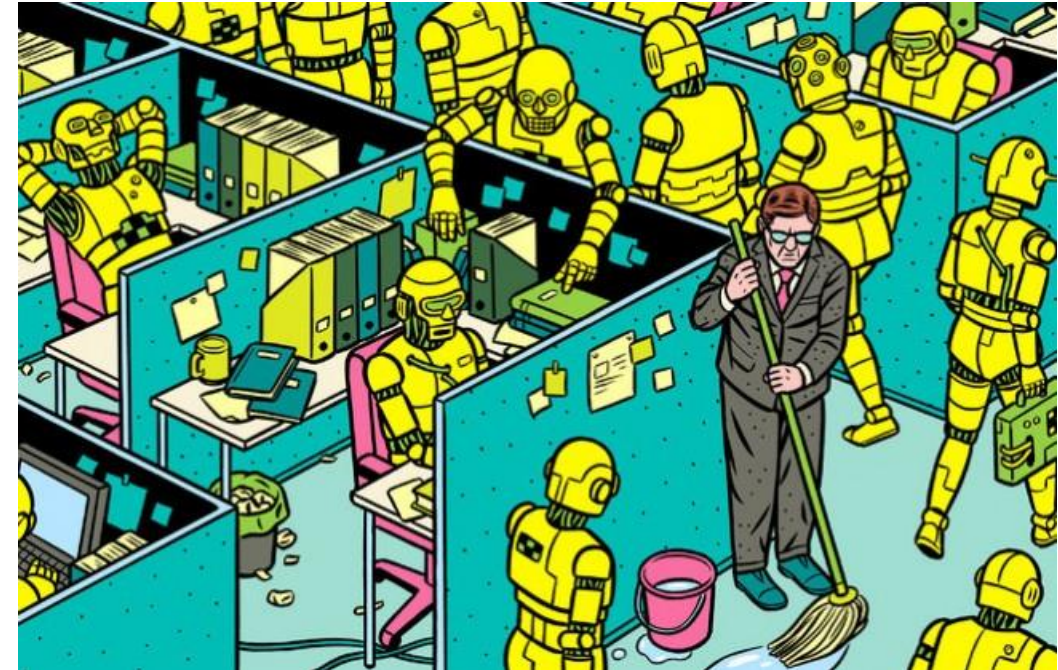


Programming methods

All robotic programming languages with each other are similar and have many identical or similar features. However, in each programming language, one can find a very strong property and at the same time, some kind of defect.

Robot programs can be entered in several ways:

- using Teach Pendant tools;
- using a special Walk-Through tool;
- using off-line programming tools;
- when writing the program code in a text editor





Teach pendants have a full menu system where you can select commands, arguments, input and output names. Syntax errors can not be made.

The teach pendants also have tools for controlling the robot's axis separately, globally, in tool and object coordinate systems, and thus specify the coordinates of the required point.



Walk-Through a programming method where the programmer has physical contact with the robot. The robot gripper has a sensor system that tracks human movements and forces the robot to move in the same way. If a robot is large, then a reduced copy is used to allow a person to perform all the movements. After that, the entire sequence of motions is stored in the robot memory, and the robot control program is generated from it.

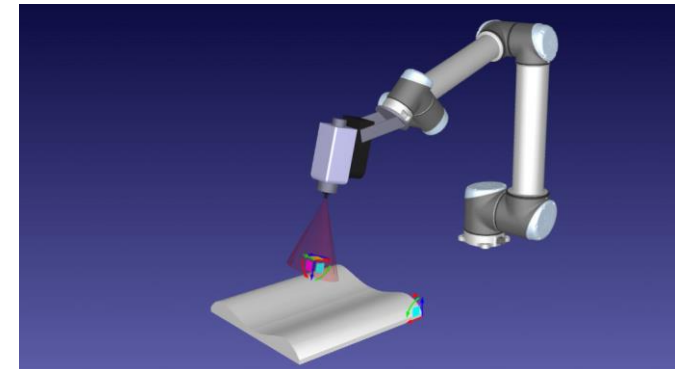


Offline Programming (OLP)

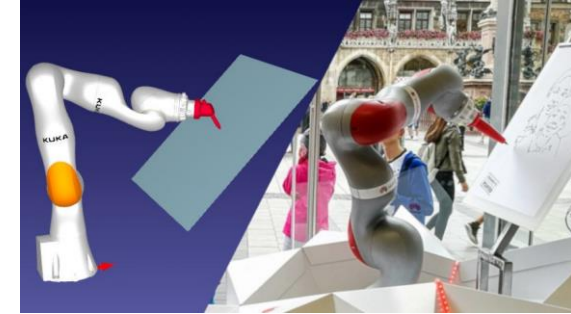
Offline programming refers to the practice of programming a machine (usually a robot or a CNC machine) without having the physical machine present. In other words, you first create the program on a computer and then later download it to the physical machine later.

In robotics, offline programming is used for a range of reasons, including:

- OLP saves time and helps you to improve your process's productivity;
- To access more advanced robot functionality by using specially designed software wizards and libraries. OLP is suited to a variety of different tasks, which it can often achieve more effectively than with conventional programming;
- To create a proof of concept before purchasing a robot, but in a way that allows you to use the same program when you do choose your robot;
- To streamline your software workflow.



Robot Simulation



A simulator is a piece of mechanical equipment or a software program which is designed to represent the conditions in a physical environment. Said another way, simulation involves imitating the real world.

The classic example of a simulator (from outside of robotics) is a flight simulator for training pilots. This machine includes both hardware and software elements to look and behave realistically like a real airplane.

In robotics, simulation is used for various purposes, including:

- To test the functionality of robot programs in a safe environment where the robot cannot harm itself or the environment.
- To test hundreds of different program permutations in a short amount of time to optimize the program.
- To run the program when no physical robot exists or one is not available.
- To create a proof of concept before you purchase a physical robot.

Many robot simulators involve a graphical representation of the robot. This is useful because it allows you to see what the simulation algorithms are actually doing underneath. Some of these simulators only have basic graphics (e.g. lines to represent the robot's links), whilst others allow you to model the entire workspace and use a realistic model of the robot.

Offline programming without simulation:

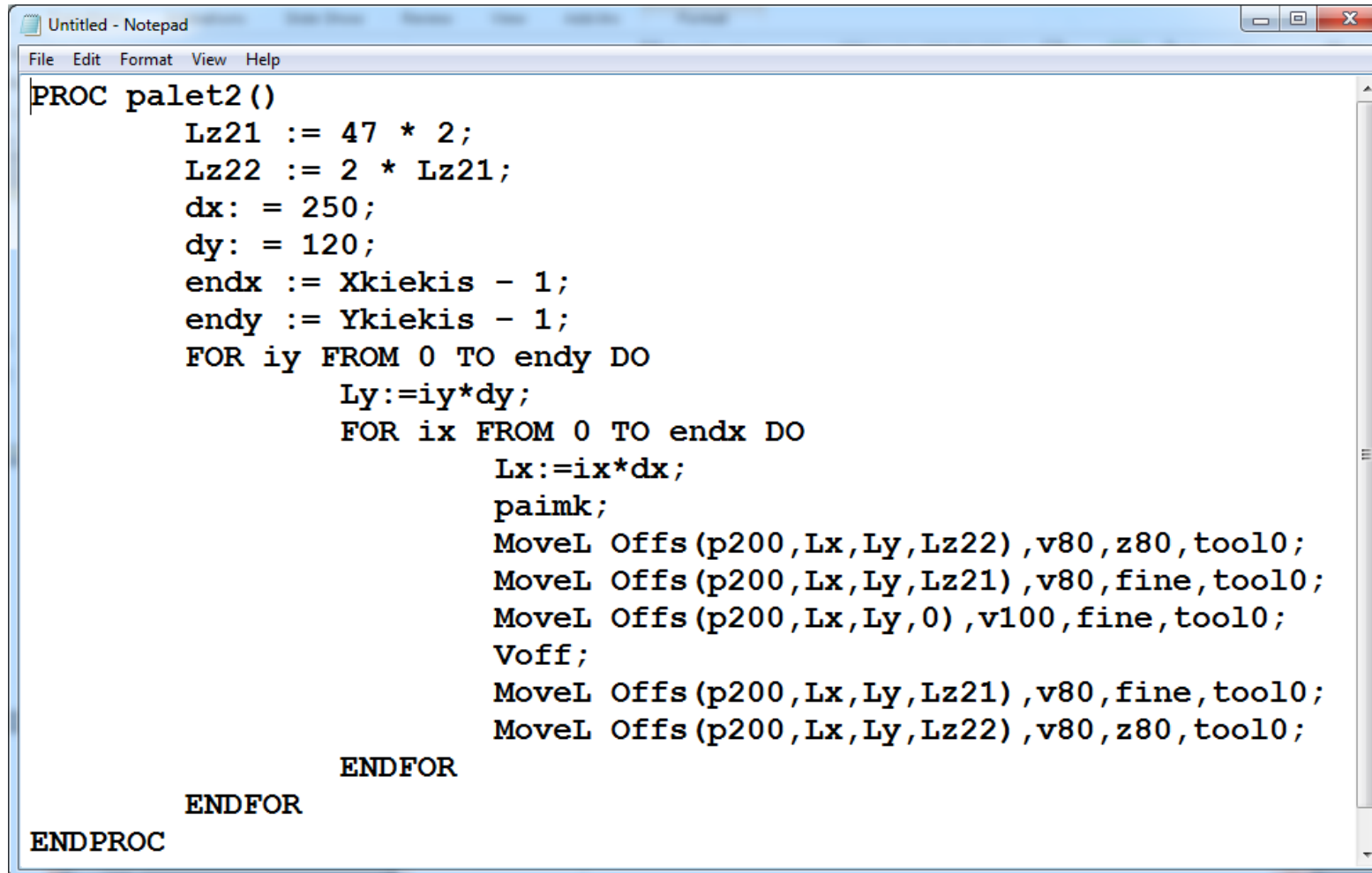
There is one situation where you would program a robot offline without using a simulator. This would be if you just programmed the robot in its native programming language using a text editor in your computer and then downloaded it directly to the physical robot once the entire program had been written.

Although this situation could technically be called “offline programming” — after all, you are still doing all the programming offline — it is not what we are usually referring to when we talk about OLP.

Usually, we mean programming a simulated robot.

Also, it is a terrible idea. If you have done any programming before, you’ll know that creating an entire program before doing any testing is a recipe for disaster. It’s much more effective to build up your program step-by-step and use a good simulator to see the effect of your instructions on a virtual robot.

Offline programming: Motion programming in a text editor environment



```

PROC palet2()
  Lz21 := 47 * 2;
  Lz22 := 2 * Lz21;
  dx: = 250;
  dy: = 120;
  endx := Xkiekis - 1;
  endy := Ykiekis - 1;
  FOR iy FROM 0 TO endy DO
    Ly:=iy*dy;
    FOR ix FROM 0 TO endx DO
      Lx:=ix*dx;
      paimk;
      MoveL Offs (p200,Lx,Ly,Lz22),v80,z80,tool0;
      MoveL Offs (p200,Lx,Ly,Lz21),v80,fine,tool0;
      MoveL Offs (p200,Lx,Ly,0),v100,fine,tool0;
      Voff;
      MoveL Offs (p200,Lx,Ly,Lz21),v80,fine,tool0;
      MoveL Offs (p200,Lx,Ly,Lz22),v80,z80,tool0;
    ENDFOR
  ENDFOR
ENDPROC
  
```


Offline programming: Motion programming in a specialized software

All of the robotic manufacturers sell software which enables programming robots without the robot itself. During the programming and program adjustment, the real robot can work on the production line.

This software has the tools to create a mechatronic robot environment - a robotic cell. It can contain various sensors, actuators, conveyors, objects, etc.

The most popular are:

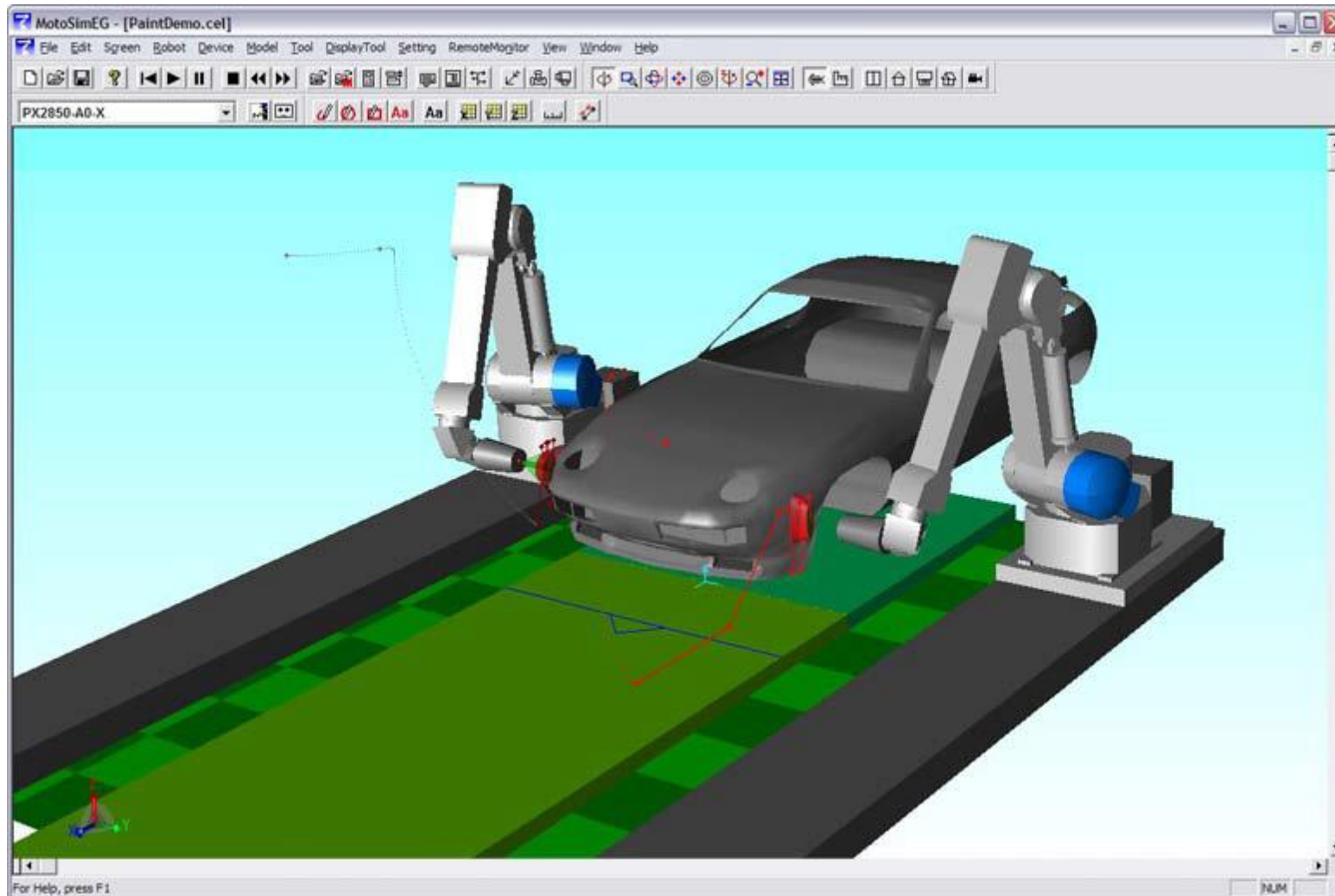
Robot Studio - ABB;

MOTOSIM EG - Motoman;

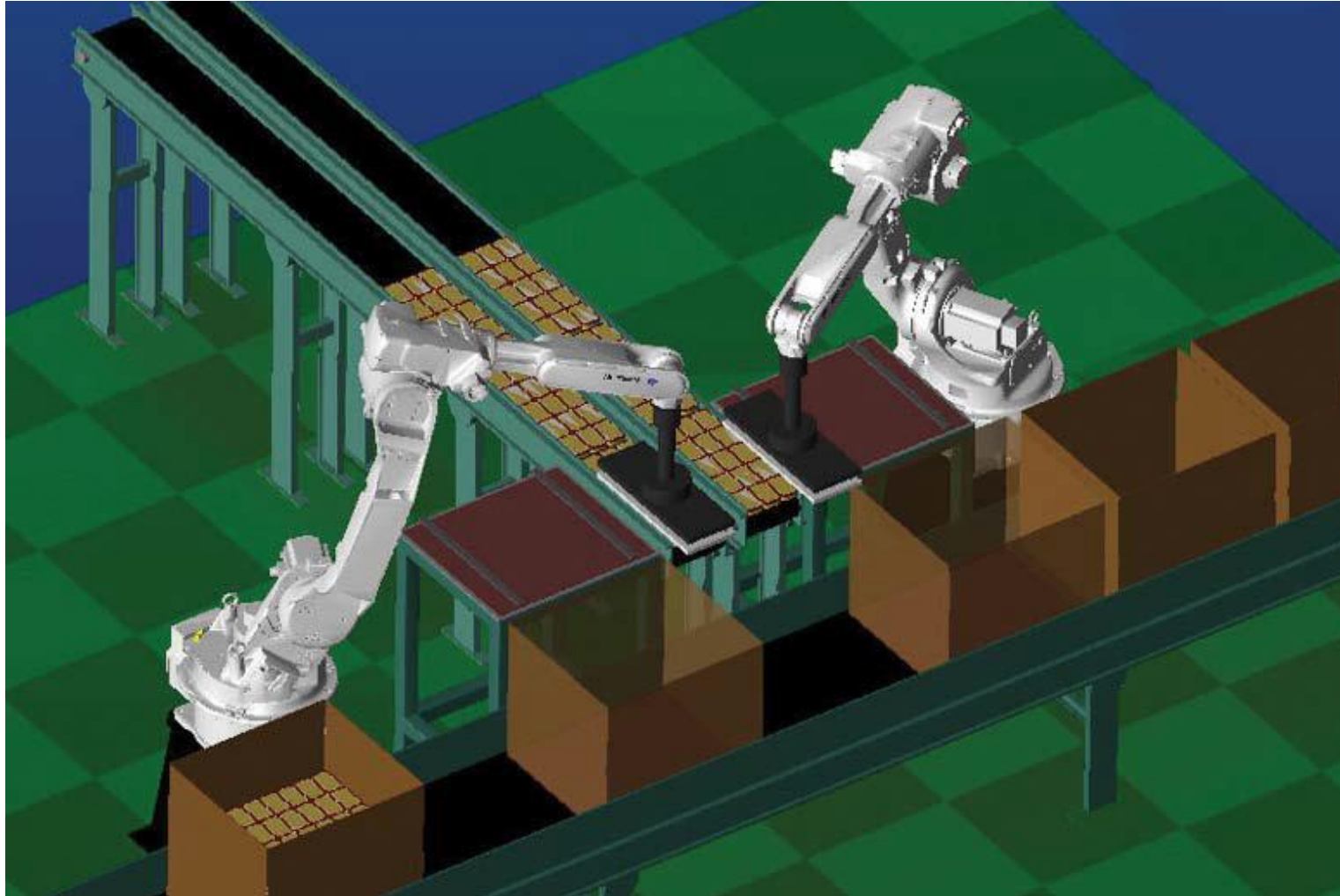
ROBOGUIDE - Fanuc;

etc.

Motion programming in the MOTOSIM EG environment. Synchronization of two robots.



Motion programming in the MOTOSIM EG medium. Two MOTOMAN robots and three conveyors.



Advantages of offline programming combined with simulation

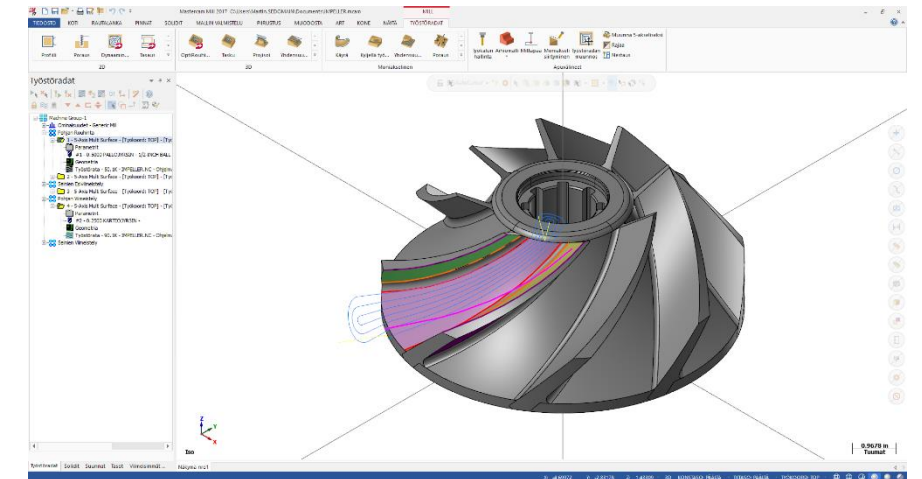
- Without the ability to test certain features in reality, we can at least create a summation for learning purposes;
- Also, everyone can try out the development of the program individually;
- The robot can be programmed even before it is set up (calibration);
- Design and engineering flaws can be identified early on. Changes can be made on the computer if necessary;
- Extensive changes to robot applications are often much easier to implement through offline programming than through direct modification of the robot;
- In the 3D computer environment, every part of the robot environment can be viewed from all sides. In real life, certain perspectives are often concealed or hard to access;
- The use of offline programming can boost the profitability of robots, particularly in machining processes, as it can reduce the time required to put them into service and minimise the need for subsequent modifications. Layout diagrams, additional axes and multi-robot solutions can be designed more comfortably and transparently, too.

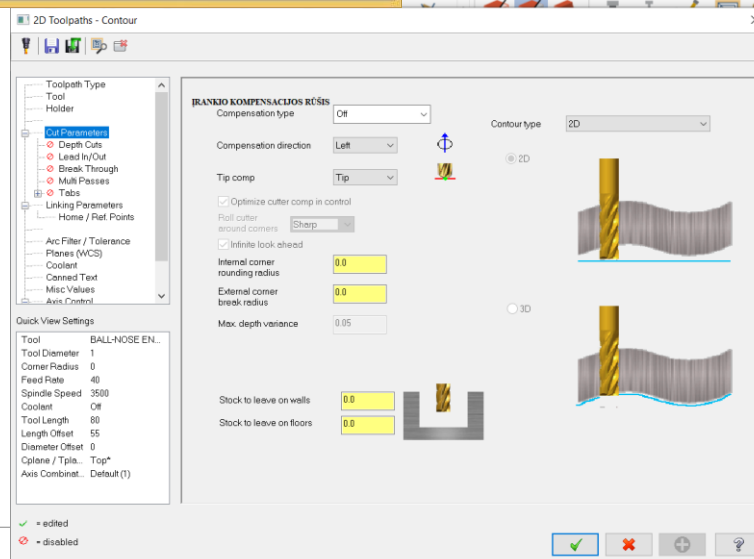
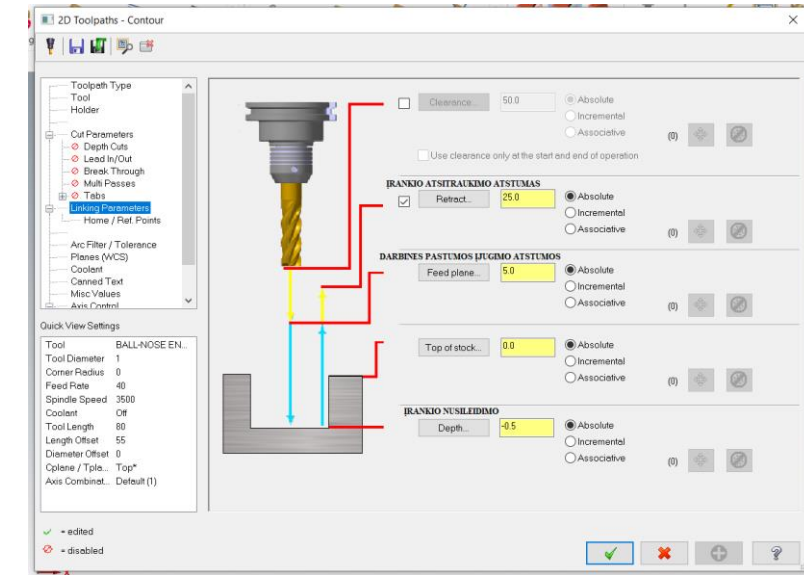
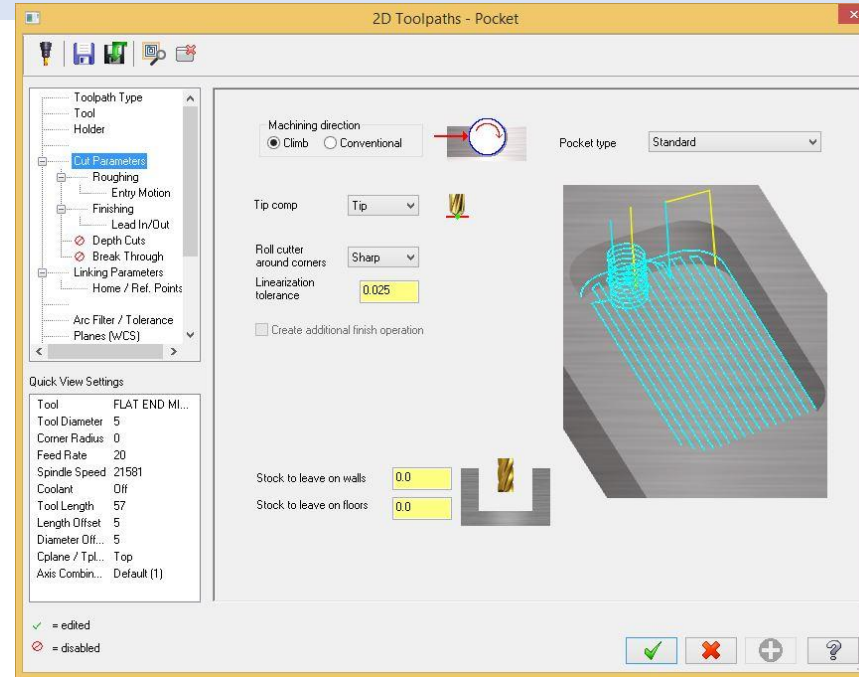
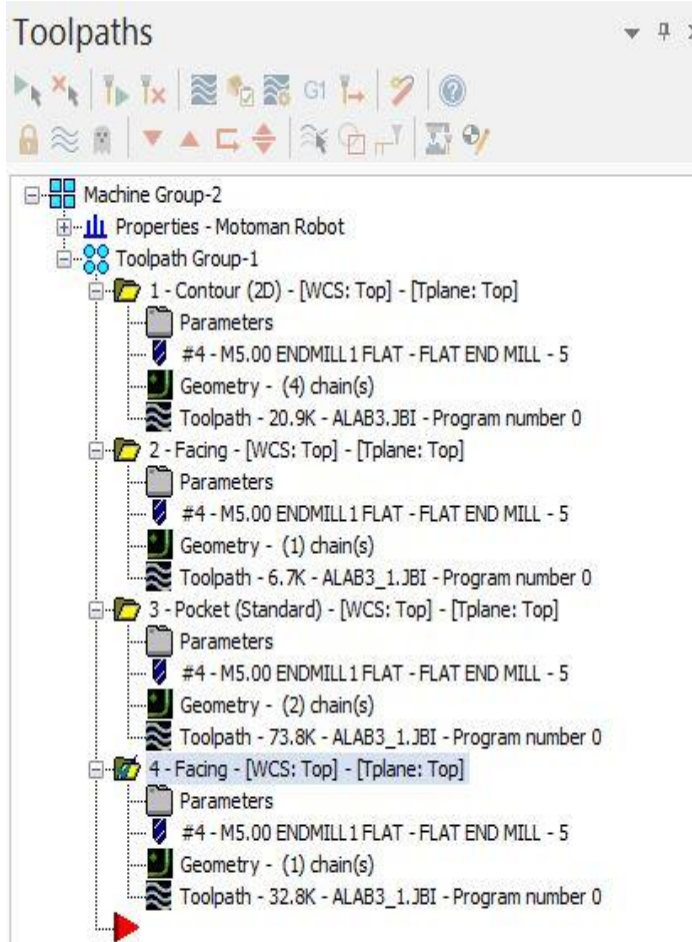
Offline programming using CAD/CAM/CAE features

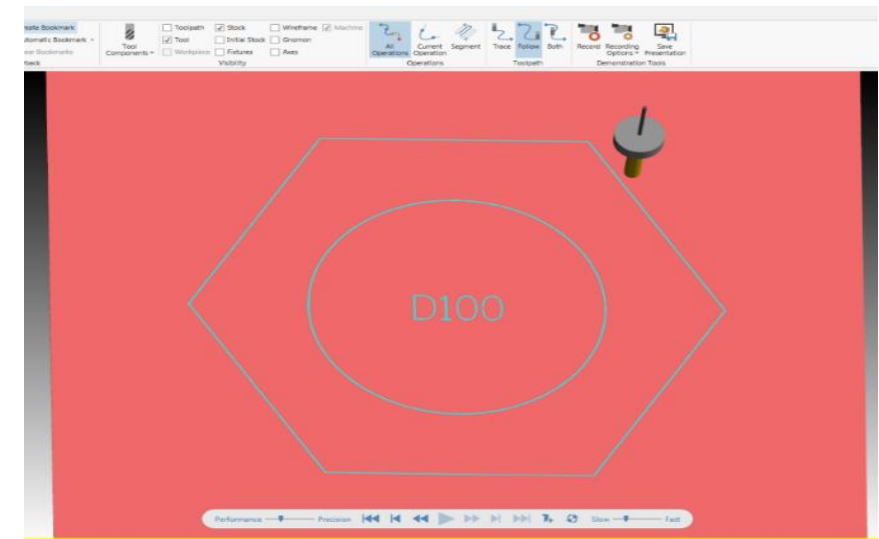
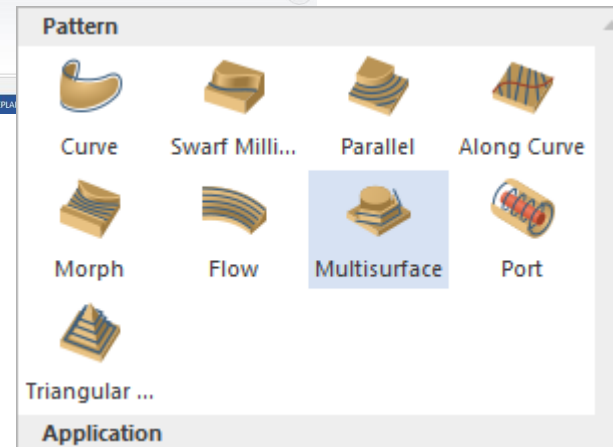
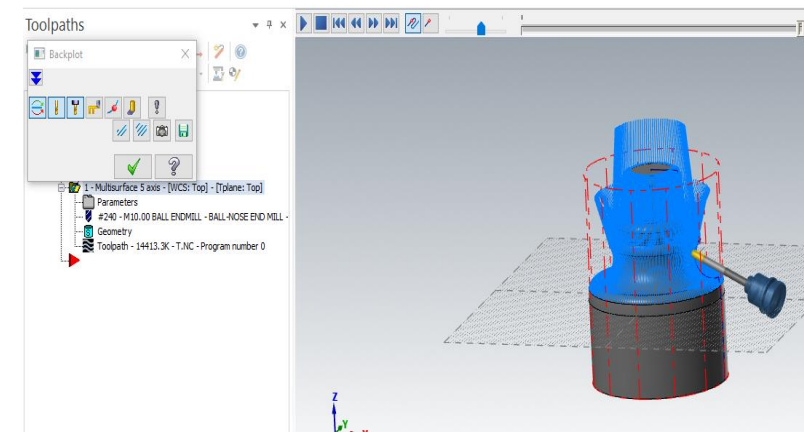
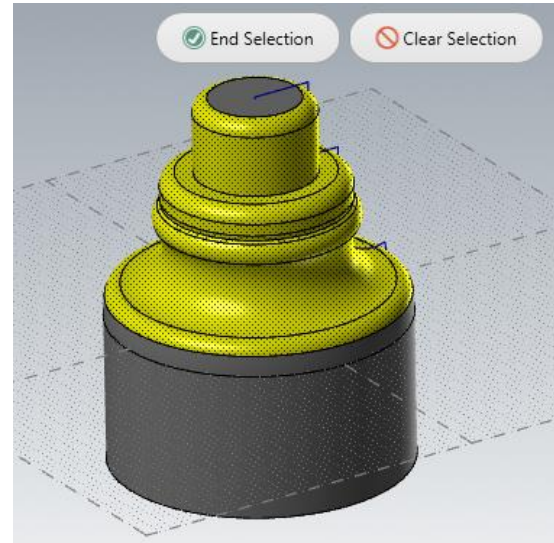
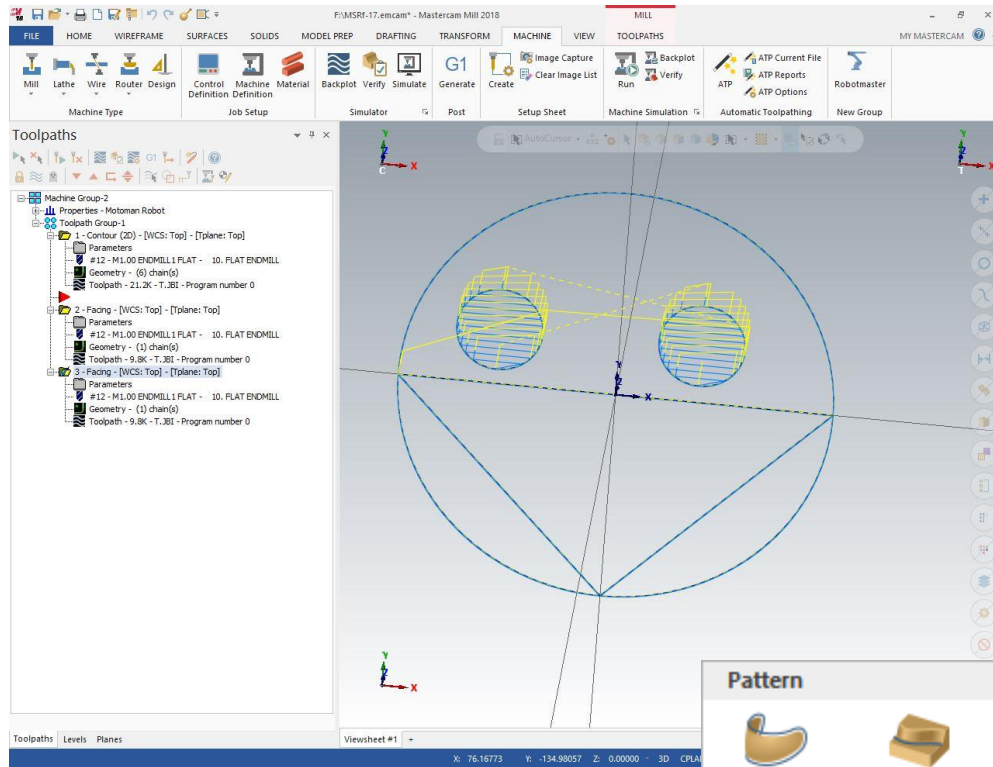
Mastercam (also MasterCAM) is a suite of Computer-Aided Manufacturing (CAM) and CAD/CAM software applications. Founded in MA in 1983, CNC Software, Inc. is one of the oldest developers of PC-based computer-aided design / computer-aided manufacturing (CAD/CAM) software. They are one of the first to introduce CAD/CAM software designed for both machinists and engineers. Mastercam, CNC Software's main product, started as a 2D CAM system with CAD tools that let machinists design virtual parts on a computer screen and also guided computer numerical controlled (CNC) machine tools in the manufacture of parts. Since then, Mastercam has grown into the most widely used CAD/CAM package in the world.

Mastercam users can create and cut parts using one of many supplied machine and control definitions, or they can use Mastercam's tools to create their own customized definitions;

Mastercam's name is a double entendre: it implies mastery of CAM (computer-aided manufacturing), which involves today's latest machine tool control technology; and it simultaneously pays homage to yesterday's machine tool control technology by echoing the older term master cam, which referred to the main cam or model that a tracer followed in order to control the movements of a mechanically automated machine tool.

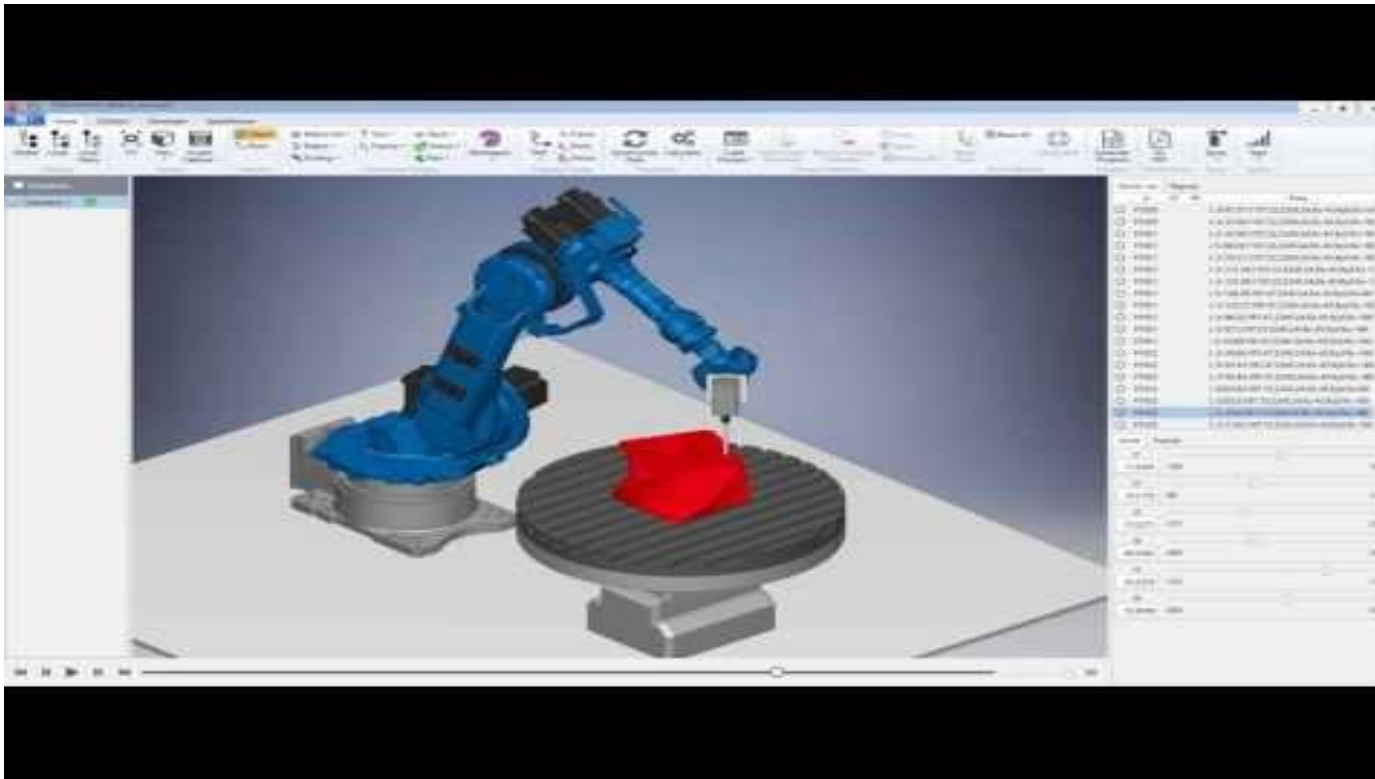






Robotmaster

Robotmaster uses integrated CAD/CAM functionality to make robotic programming easy and intuitive for everyone. It is used by a wide range of industries to program robots for tasks that include surfacing, 3D milling, additive manufacturing, welding, painting, and more.



Robotmaster is very powerful for path planning and programming, but we don't cater to the markets of pick and place or palletization.



Cutting

Robot work cells are flexible cost effective solutions for cutting complex shapes on 2 or 3D surfaces using laser, plasma, water-jet, gas cutting, or compliant knife tooling, with high speed and precision, without additional finishing steps.

[Learn More](#)



Machining

Robotic milling offers versatility to create simple or complex parts of any size with lower equipment and operating costs.

[Learn More](#)



Trimming

Robot work cells are flexible cost effective solutions for cutting complex shapes on 2 or 3D surfaces using for cutting, routing, trimming or compliant knife tooling, with high speed and precision, without additional finishing steps.

[Learn More](#)



De-burring

Robotmaster can be used to generate a robot program for deburring rough edges of machined parts or gears, or for any other process that requires smoothing of edges or surfaces.



Paint/Spray Coating

Robots can precisely control spray angle, speed and flow intensity for complete and consistent coverage over complex surfaces.

[Learn More](#)



Surface Treatment

Robots produce consistent, high-quality surface finishing and hardening with greater throughput in the face of a worldwide shortage of skilled surface finishing craftsmen.



Welding

Robotic welding increases shop productivity and delivers high weld quality in the face of a worldwide shortage of skilled welders.

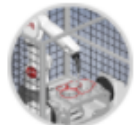
[Learn More](#)



Polishing/Sanding

Robots produce consistent, high-quality finishing with greater throughput while reducing the exposure of workers to the contaminants, noise and monotony of grinding, polishing and sanding processes.

[Learn More](#)



Dispensing

Robotmaster can be used for programming the robotic dispensing of adhesives and seals.

[Learn More](#)



Cooperative robotics

Robots can simultaneously work together to thermal spray a turbine blade

[Learn More](#)



Additive Manufacturing

Automated programming for robotic additive manufacturing applications can be generated from CAD models and subsequently modified with minimal effort to adjust for all process based parameters

[Learn More](#)



Deflashing

Robotmaster can be used for robotics programming of deflashing and compliant knife trimming applications.

[Learn More](#)



Part to Tool

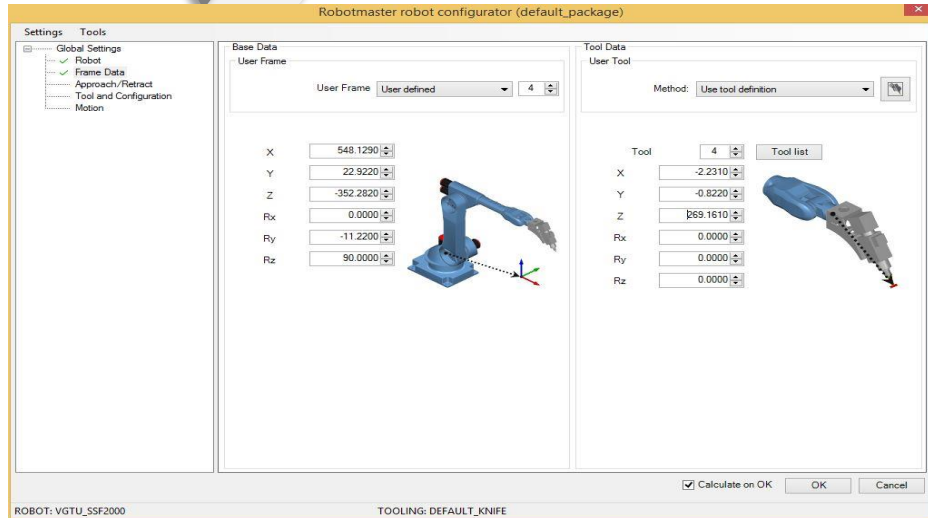
Robotmaster can be used for programming part to tool applications.

[Learn More](#)



Rail/RotaryPolishing/Sanding

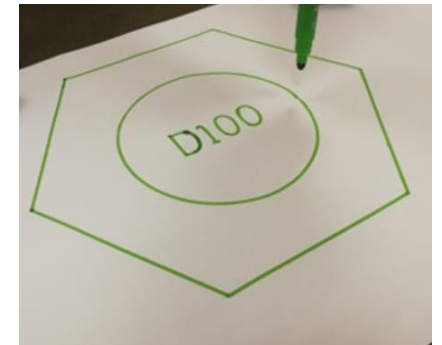
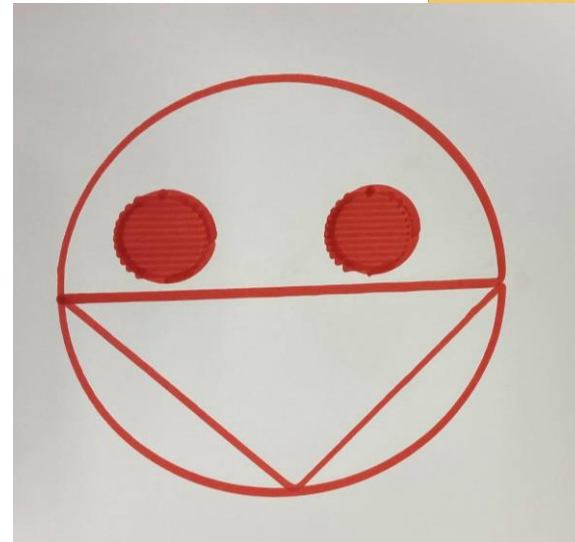
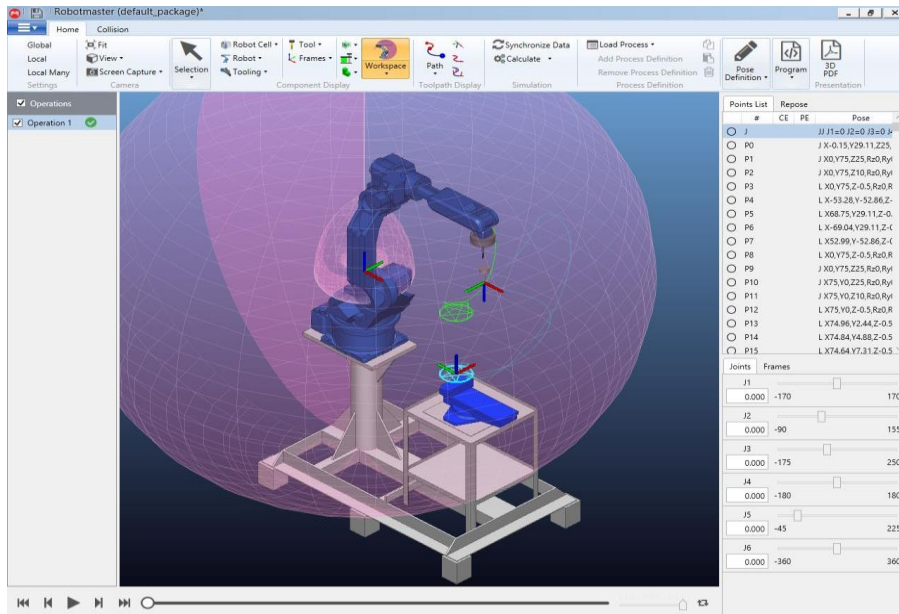
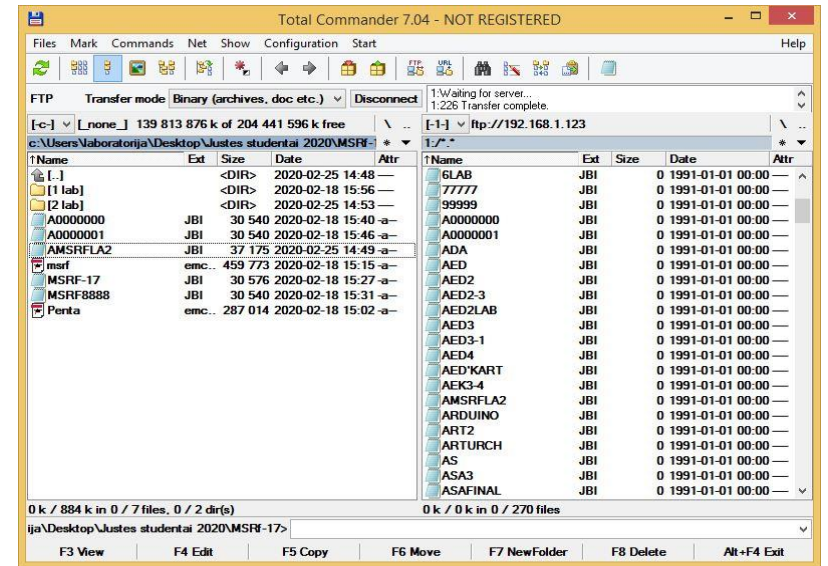
Robotmaster provides an extensive suite of robotic programming and simulation functionality for rails and rotaries to easily program robots mounted on linear rails or parts mounted on rotary tables. The rails and rotaries can be used as indexers or in full simultaneous motion.



*TMRINO42 – Užrašinė

Failas Redagavimas Formatavimas Rodymas Žinynas

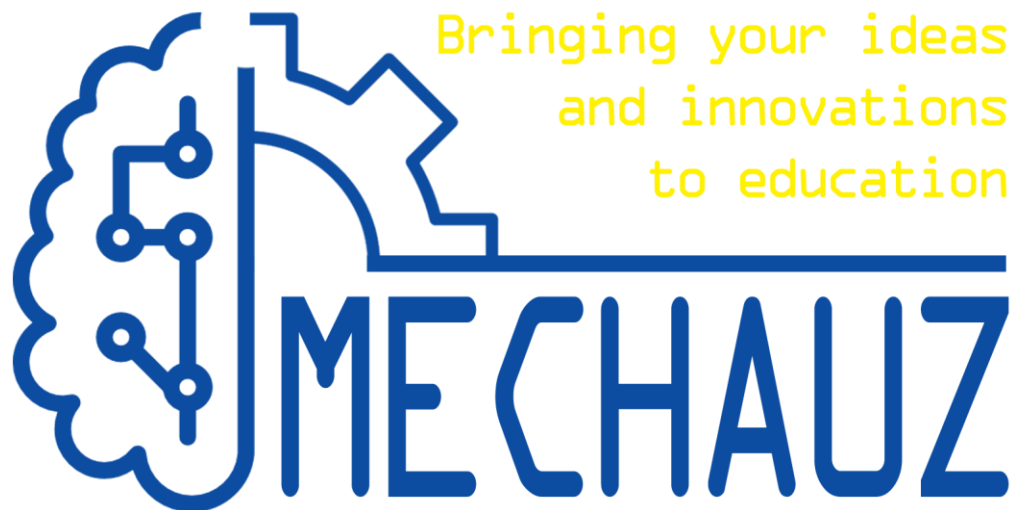
```
//JOB
//NAME TMRINO01|
//POS
//NPOS 375,0,0,0,0
//USER 4
//TOOL 4
//POSTYPE USER
//RECTAN
//RCONF 1,0,0,1,0,0,0,0
C0000=57.735,200.000,25.000,180.000,0.000,0.000
C0001=57.735,200.000,5.000,180.000,0.000,0.000
C0002=57.735,200.000,-0.500,180.000,0.000,0.000
C0003=0.000,100.000,-0.500,180.000,0.000,0.000
C0004=57.735,0.000,-0.500,180.000,0.000,0.000
C0005=173.205,0.000,-0.500,180.000,0.000,0.000
C0006=230.940,100.000,-0.500,180.000,0.000,0.000
C0007=173.205,200.000,-0.500,180.000,0.000,0.000
C0008=57.735,200.000,-0.500,180.000,0.000,0.000
C0009=57.735,200.000,25.000,180.000,0.000,0.000
C0010=88.624,187.250,25.000,180.000,0.000,0.000
C0011=88.624,187.250,5.000,180.000,0.000,0.000
C0012=88.624,187.250,-0.500,180.000,0.000,0.000
C0013=88.624,167.250,-0.500,180.000,0.000,0.000
C0014=94.203,167.250,-0.500,180.000,0.000,0.000
```



Questions to Ask before choosing a Robot Simulation Software:

- What robot brands and models does the software support?
- What are the options for training and support?
- Is this the best software solution for my process application?
- How long does it take to create a typical program?
- How long does it take to solve robotic errors and collisions?
- How challenging will it be to translate my knowledge of a given CAD or simulation software to this new software?
- Will it be possible or convenient to expand from my current process, such as welding, to another process such as palletizing or painting?
- How is data handled and shared?
- Will the robot show all the process details I need, not just the robot? (spray deposition, part positions, etc.)
- How accurate is the software? Is an external measuring device required or is just a robot used for measuring?

<https://www.engineering.com/story/the-what-why-and-how-of-industrial-robot-simulation-software-for-offline-programming-olp>



www.mechauz.uz

THANK YOU

FOR YOUR ATTENTION