Industrial Robot Programming

Saku Pöysäri
Tampere University

Tampere University

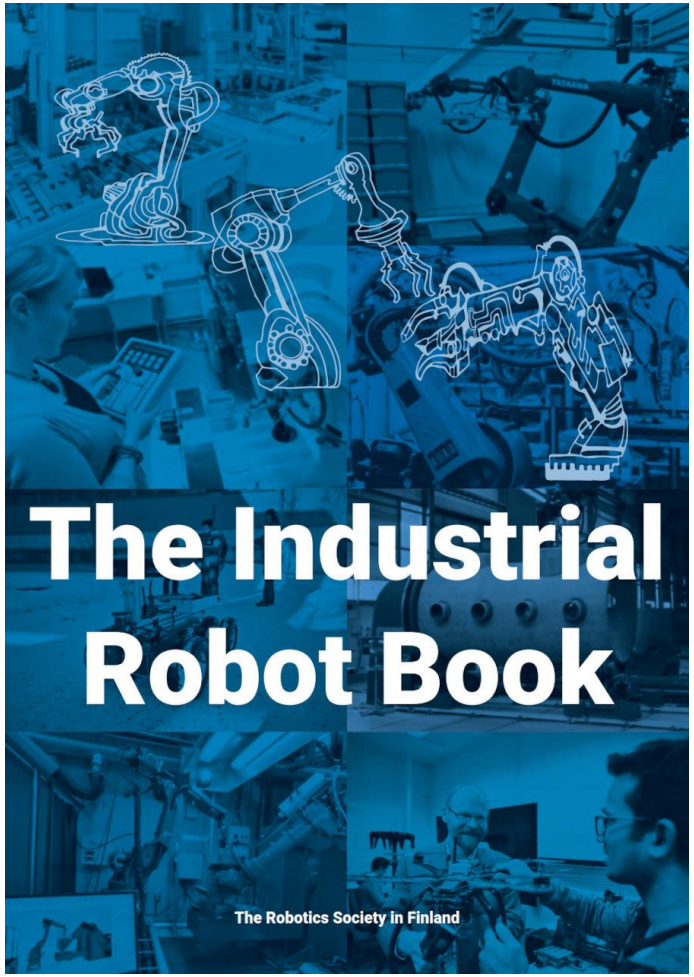
Funded by the European Union – NextGenerationEU

SECLE
The Service Centre for Continuous Learning and Employment

This material has been produced as part of the Komeetta training. The training has been funded by the European Union's Recovery and Resilience Facility (RRF), which is the key instrument of the EU's Recovery Facility (NextGenerationEU). The financing has been granted by the Service Centre for Continuous Learning and Employment. The Service Centre promotes the competence development of working-age people and the availability of skilled labour. The operations of the Service Centre are overseen by the Ministry of Education and Culture and the Ministry of Economic Affairs and Employment.

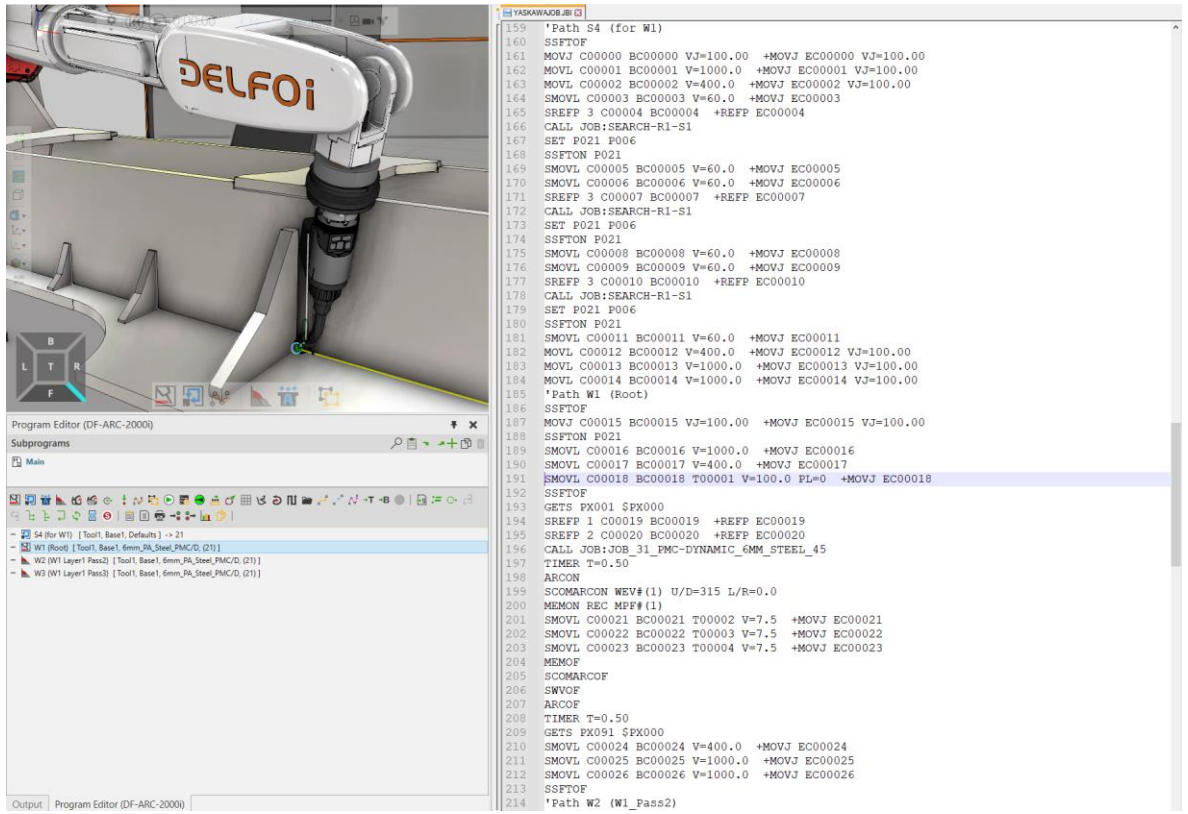
The Industrial Robot Book

- Lecture material is based on the content of the book
  - Chapter 5 – Industrial Robots
  - Chapter 11 – Robot programming
  - Chapter 12 – Simulation and offline programming
- You can buy the book from ellibs online shop or loan it from the university library
  - ISBN 978-952-65329-1-2 (EPUB)
  - ISBN 978-952-65329-2-9 (PDF)



Main tasks of robot programming

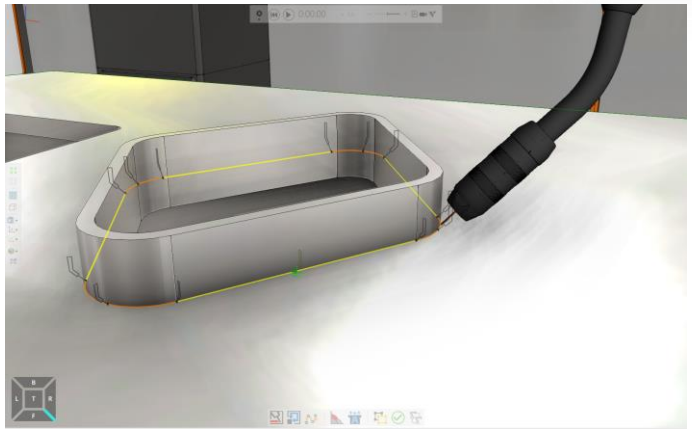
- How robot moves?
  - Sequence of operations
  - Motion paths
- How robot communicates?
  - Communication with external devices
- How robot operates in case of errors?
  - Interrupts and error recovery



Funded by the European Union – NextGenerationEU

Programming methods

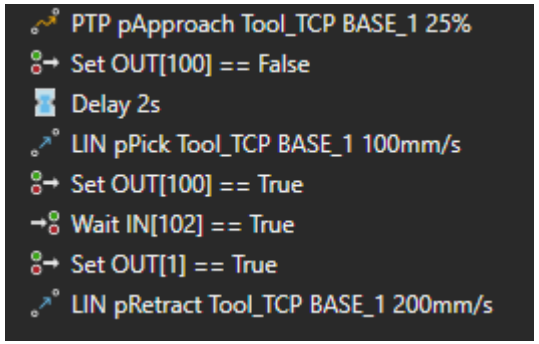
- Programming by guiding
  - Robot's arm and tool are manually moved to the desired position
  - This method has become prevalent with the introduction of collaborative robots and force control
- Teaching
  - Robot's arm and tool are positioned using a hand controller
  - This approach is suitable for programming “simple applications”
- Offline programming
  - Robots are programmed entirely using a computer outside of production
  - Programming can be text-based and/or model-based programming
  - This method is particularly suited for programming “complex applications”
  - It enables programming without interrupting production



Programming languages

- Each robot brand has its own programming language
- The structures of commands and naming conventions used in programming vary between robot brands, but the fundamental principles of programming remain the same
- Programming is based on traditional programming languages like Python and C++, which support mathematical and logical operations
- Programming can utilize either brand-specific software or generic software

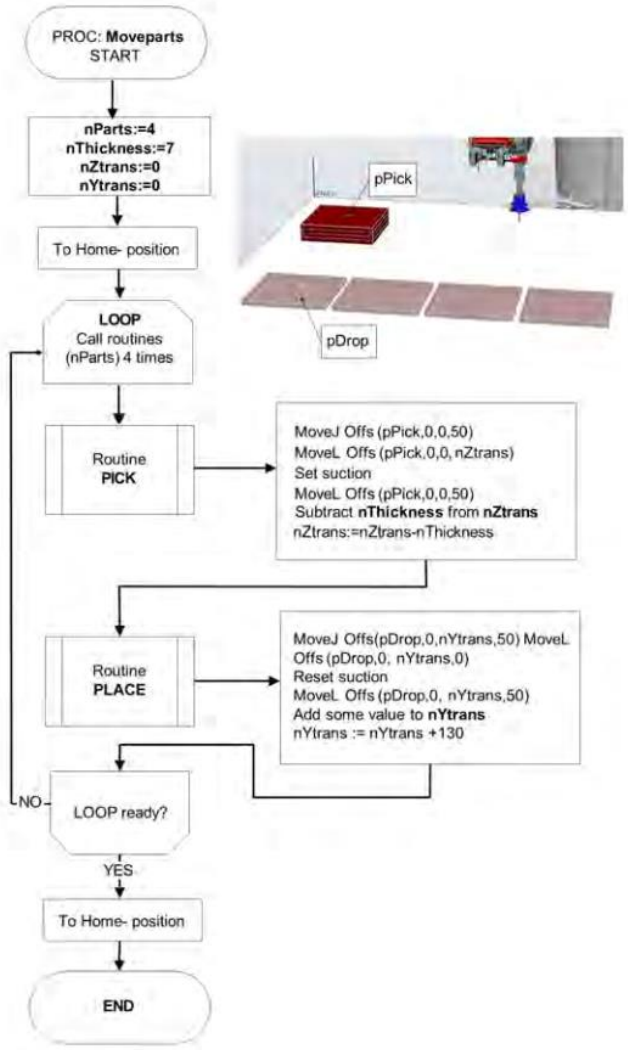
```
PROC main()
    MoveJ pHome,v1000,fine,Pen_TCP\WObj:=wobj0;
    TPErase;
    TPWrite "";
    WHILE TRUE DO
        TPReadFK answer, "Select the product to be manufactured:",
            "Cylinder", "Circle", "Quit", stEmpty, stEmpty;
        IF answer = 1 THEN
            size_determination answer;
        ELSEIF answer = 2 THEN
            size_determination answer;
        ELSE
            TPErase;
            quit;
        ENDIF
        MoveJ pHome,v1000,fine,Pen_TCP\WObj:=wobj0;
        TPErase;
        WaitTime 1;
    ENDWHILE
ENDPROC
```




```
   5:  !Check at HOME
   6:  IF DO[101:HOME Signal]=ON,
    :  JMP LBL[10]
  7:J  P[2:Start] 10% CNT100
   8:  LBL[1]
   9:  R[6:User Input]=0
  10:  CALL LISTMENU(2,6)
  11:  SELECT R[6:User Input]=1,
    :  JMP LBL[3]
  12:        =2,JMP LBL[5]
  13:        ELSE,JMP LBL[2]
```

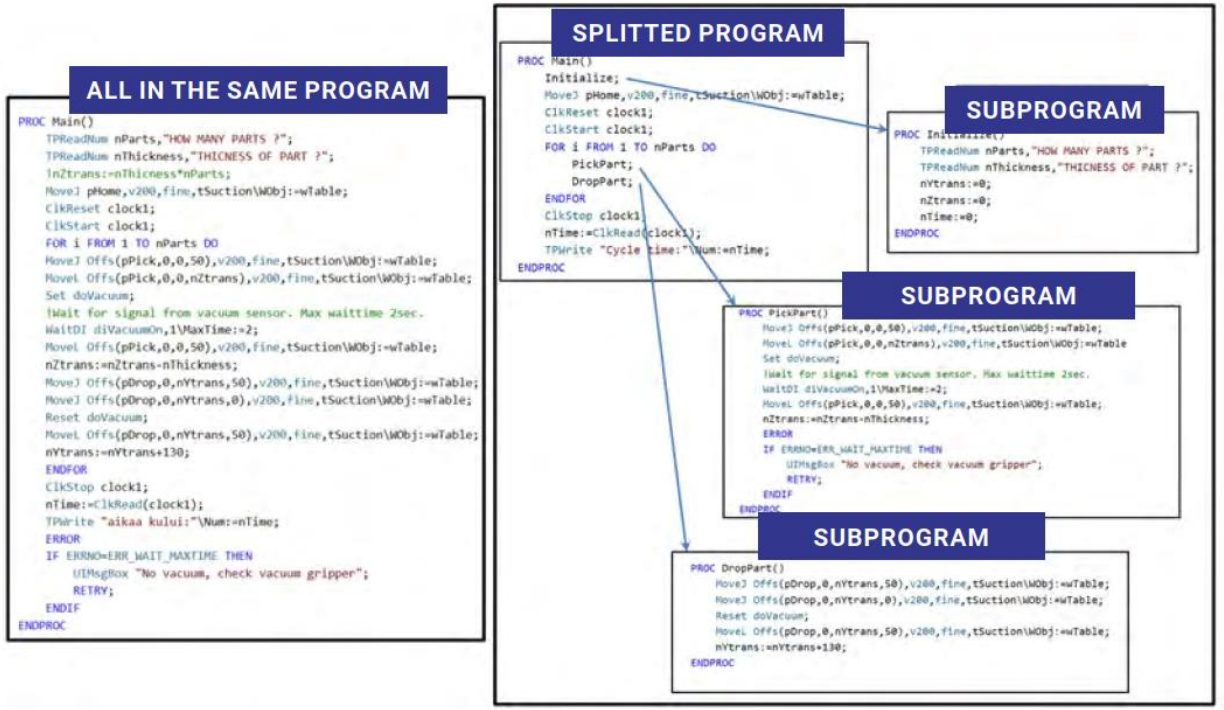
Stages of programming

- Start with process description
  - What should the program do? What are the requirements? What is the workflow?
- Programming the robot
  - Programming principles/guidelines
  - Preliminary reach and collision analysis
  - Definition of variables and other programmable components, such as tool center points, user coordinate systems, signals, etc.
  - Ensure the functionality of selected tool center points and user coordinate systems
  - Sequence of operations and program structure
  - Programming motion commands and other instructions
  - Define background tasks, error handling, and communication
  - Program testing, calibration, and commissioning
- Finally remember the documentation, including etc.
  - Program description
  - Author, date, version
  - Naming of points, variables, signals, TCPs, etc.
  - Commenting the program for better understanding by others
  - Detailed documentation of error handling
  - Listing and documenting peripheral devices and signals



Program structure

- Robot's work program should consist of a main program and subprograms
- Main program should be clear and simple, describe robot's primary tasks, and define the workflow
- Subprograms should contain the functionalities for the specific robot's work tasks
- Program structure should utilize modularity and parameterization whenever possible
  - Reusable subprograms support modularity and eliminate unnecessary reprogramming
    - E.g. standardized programs for tool change
  - Parameterized subprograms improve program reusability by allowing their parameters to be changed dynamically
    - E.g. producing similar but differently sized products using the same program



Robot commands

- A command (or instruction) defines a task that is executed when the command is carried out, such as
  - Moving the robot's arm
  - Turning the robot’s output on/off
  - Waiting for external sensor data
  - Modifying the value of a variable
  - Jumping within the program
- Commands consist of the command name and various definable arguments
  - The command name specifies the primary task, while the arguments provide specific characteristics
  - An argument can be either mandatory or optional, with optional arguments being disregarded if not needed

Basic commands for Industrial robots

- Motion commands
- Program flow commands
- Read and write commands for variables and registers
- Communication commands for external devices
- Error handling commands
- Application-specific commands

Funded by the European Union – NextGenerationEU

Motion commands

```
MoveJ pPoiminta,v100,z50,tTarrain32\WObj:=wTyopoyta1;
```

Motion Type | Target Point | Speed | Area | Tool (TCP) | (User) Coordinate System

- **Motion type**: Defines the type of movement between points
- **Point**: Defines the position, orientation and configuration of the target point
- **Motion speed**: Defines the movement speed of the tool center point (TCP)
- **Approach data**: Defines how close to a point the TCP moves when passing through it
- **Tool**: Defines the tool center point used for positioning
- **Coordinate system**: Defines the coordinate system in which the target point is placed

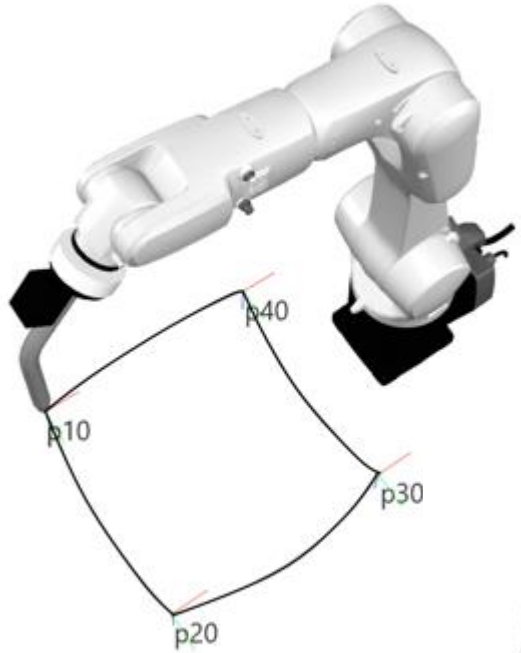
```
J  P[2:Start] 10% CNT100
```

```
LIN pRetract Tool_TCP BASE_1 200mm/s
```

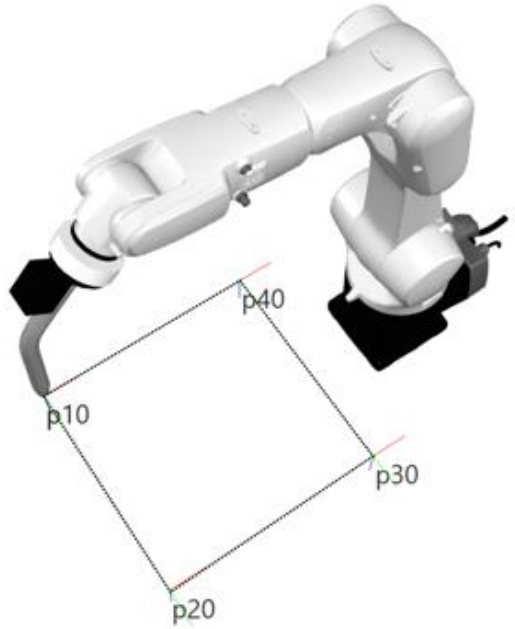

Funded by the European Union – NextGenerationEU

Motion types

- Joint motion
  - The "easiest" way for the robot to achieve positioning
  - Typically the "fastest" type of motion
  - "Undefined" path to the target point
  - Suitable for movements where precise trajectory tracking is not critical
- Linear motion
  - The "most laborious" way for the robot to achieve positioning
  - Typically the "most precise" type of motion
  - Moving along a linear path to the target point
  - Suitable for movements where precise trajectory tracking is critical



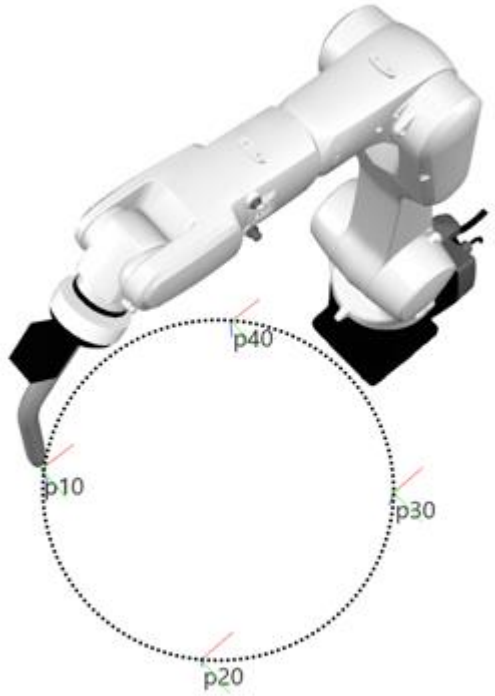
Joint motion



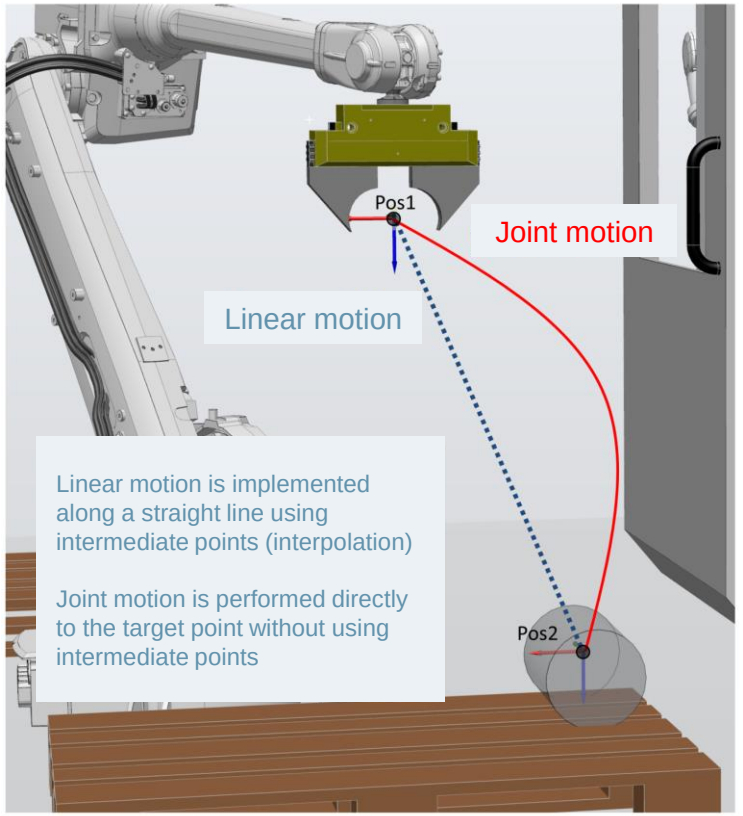
Linear motion

Motion types

- Circular motion
  - Arc movement through a via point to the endpoint
  - Implemented with two positioning points
    - Via point
    - End point
  - Typically, a complete circle motion consists of two circular motion commands
- Suitable motion type is chosen on a case-by-case basis

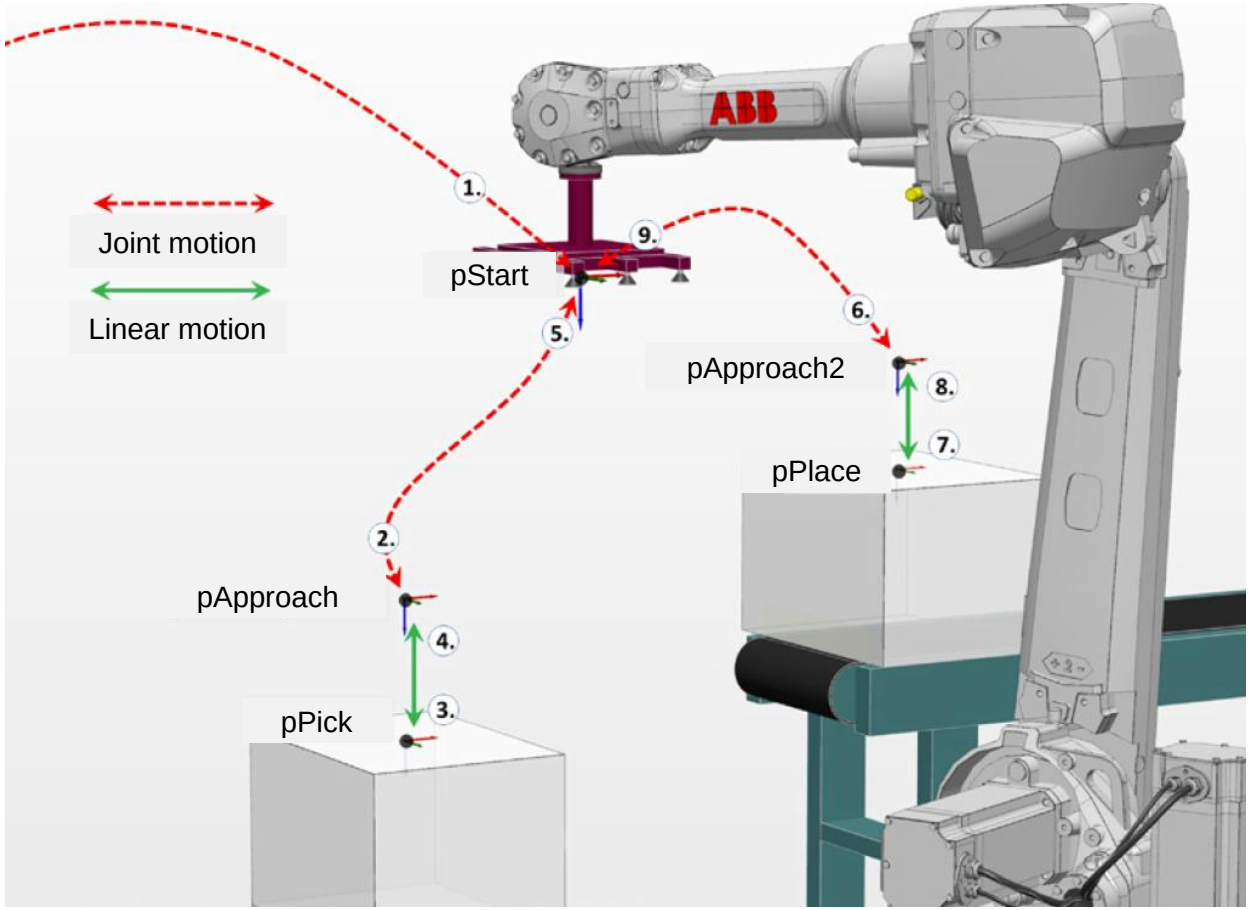


Circular motion



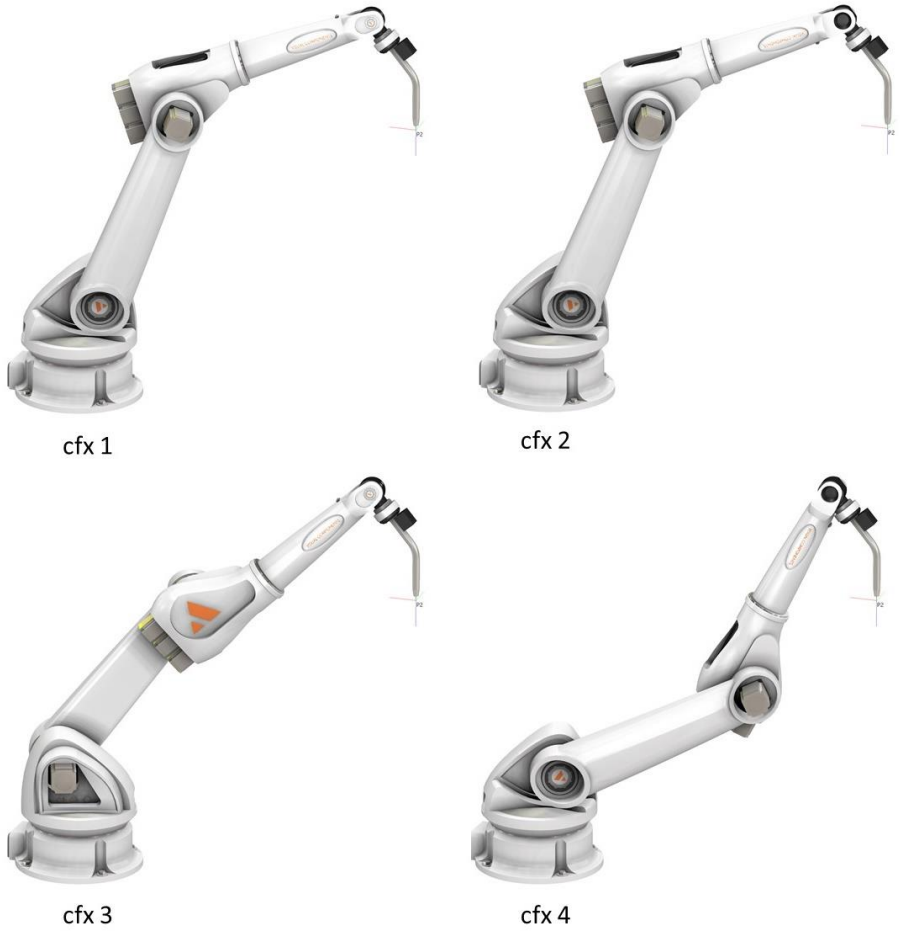
Points

- The location and orientation of the robot's tool center point is relative to the selected coordinate system, including information about the joint configuration
- Motion commands and target points along the robot's trajectory generally follow the same pattern regardless of the application, such as welding, painting, or assembly
  - 1. Start point
  - 2. Approach point
  - 3. Target point(s)
  - 4. Retract point
  - 5. Return to start point
- When defining target points, it is advisable to
  - Use a consistent naming convention
    - E.g., p10, p20, or pPick, pPlace
  - Define target points in the user coordinate system to make modifications easier later on
  - Manage the number of target points to reduce the number of points that need to be maintained and programmed
    - Utilize offset commands when defining approach and retract points



Joint configurations

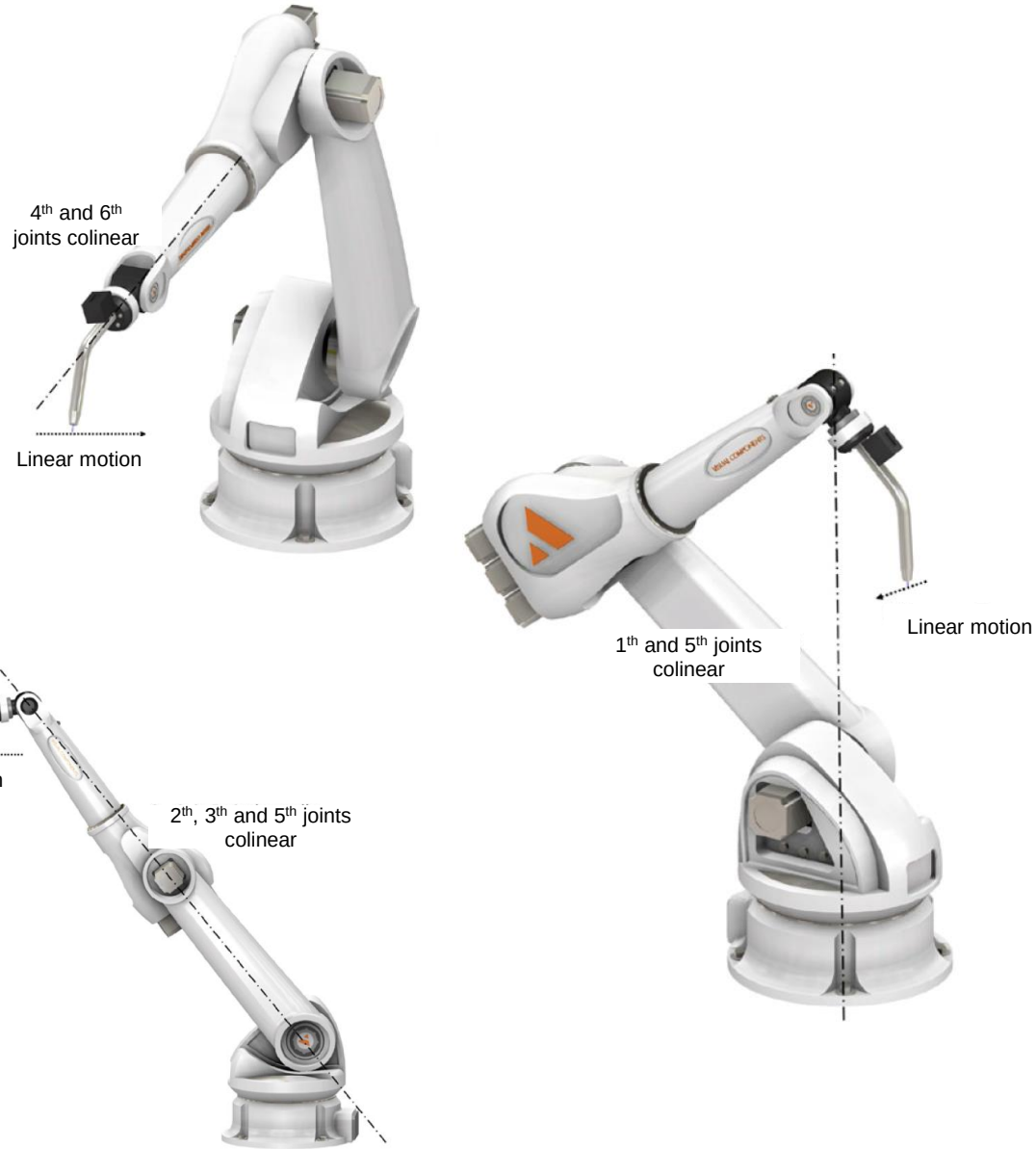
- Single point can be reached with multiple joint configurations
  - In practice, different poses of the robot's arm can achieve the same target point's location and orientation
  - Therefore, a joint configuration should be defined for each target point in addition to its location and orientation
- In cases of joint configuration errors, the robot may not reach the target point with the given joint configuration
  - This could be due to a significant change in joint configurations between separate target points
- To avoid joint configuration errors
  - Minimize joint configuration changes along the motion paths
  - Avoid large changes in joint configurations when switching between points
  - Avoid joint configurations where the robot's joints are at the extremes of their motion ranges



Funded by the European Union – NextGenerationEU

Singularity

- Singularity errors are related to situations where the robot attempts to maintain the tool point's orientation and trajectory throughout its entire movement
- In certain cases, the robot's ability to move the tool point at the selected speed and orientation can be lost, leading to uncontrolled motions and speeds
- These situations occur when
  - Two of the robot's joints align in the same direction (colinear)
    - Causing the joint angle velocities to reach infinite values
  - Two of the robot's joints are concentric
    - Resulting in a loss of degrees of freedom in the motion
- These scenarios are referred to as singularity points, which can be categorized into three types
  - Wrist singularity
  - Elbow singularity
  - Shoulder singularity



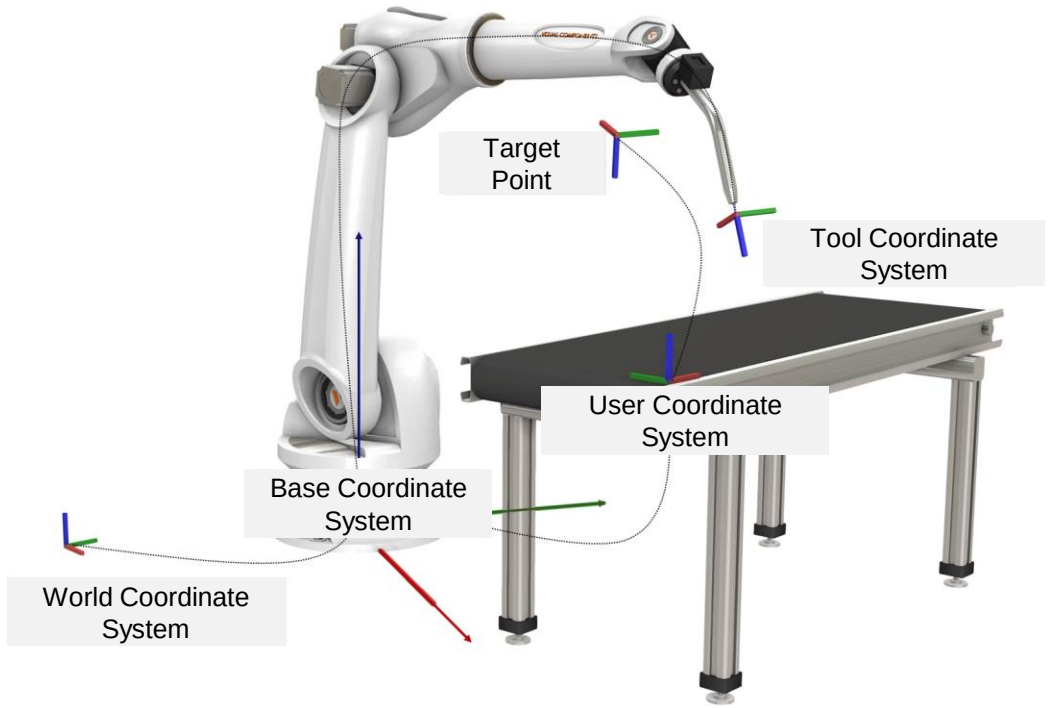
Singularity

- Typically, robot stops at the point of a singularity
- However, near singularity points, it is possible to allow the robot to deviate from its path
  - Bypassing a singularity point
- Joint movements are "singularity-free," as the joints move independently toward their target positions rather than along a predetermined path
- Avoiding singularity errors can be achieved through careful programming and design
  - Avoid programming target points close to singularity points
  - Use intermediate points if target points are far apart
  - Utilize joint movements whenever possible to ensure controlled transitions into and out of the manufacturing process
  - In tool design, avoid setting the fifth axis at a 0° position → The wrist should be slightly tilted
- Some robot brands have special commands to handle singularities and joint configuration errors
  - Special commands define the robot's behavior in such situations

Funded by the European Union – NextGenerationEU

Coordinate systems

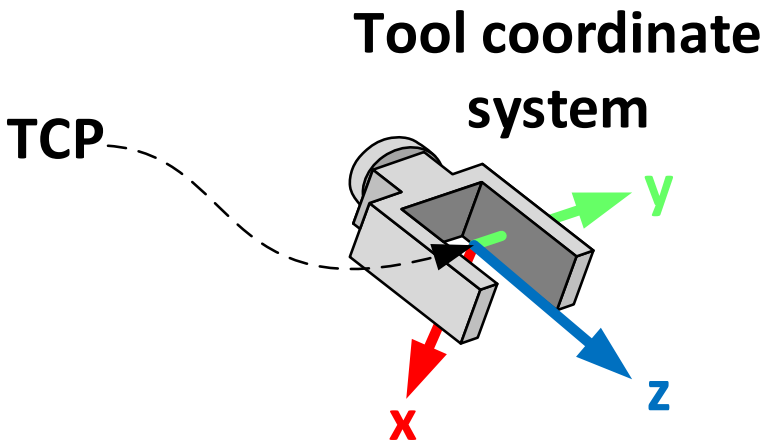
- Coordinate systems are used to represent the locations and orientations of points in a three-dimensional environment
- Various coordinate systems of robots are interconnected
- Tool and user coordinate systems are linked to the base coordinate system, which determines the placements and movements of the robot
- Target points are always stored in the selected coordinate system
  - If a user coordinate system has not been defined, it is replaced by the base coordinate system
- User coordinate system defines a coordinate system specifically intended for the robot's working area
  - It indicates both the location of the origin (0, 0, 0) and the angular offsets around the X, Y, and Z axes in the base coordinate system
  - It is possible to define multiple user coordinate systems for a robot, so it is essential to choose which one to use
  - Defining positioning points in the correct user coordinate system facilitates the modifications and transfers of programs later on

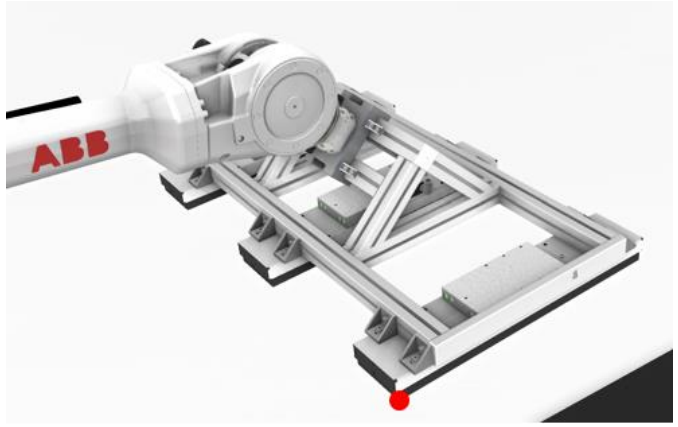


Tool Center Point

- Robot's positioning and movements are linked to the tool center point and the tool's orientation

  → Tool coordinate system
- Tool Center Point (TCP) is the point to which all robot positioning is related
- It is possible to define multiple tool coordinate systems for a robot, so it is essential to choose which one to use
  - If a user coordinate system has not been defined, it is replaced by the default tool coordinate system





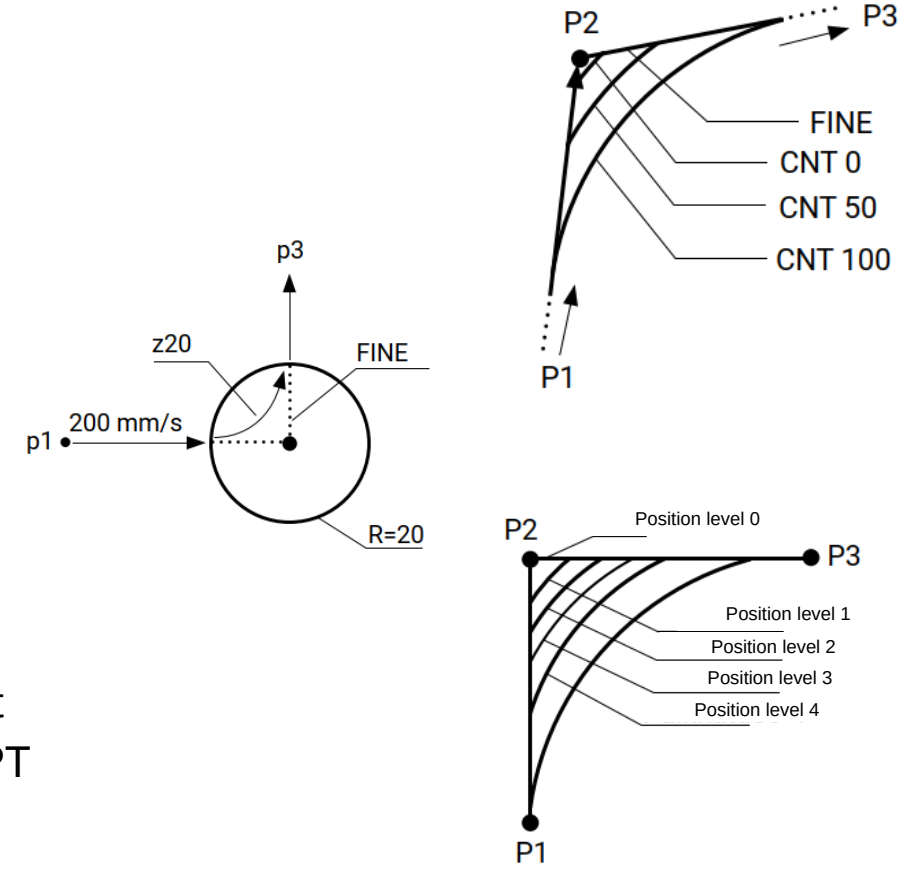
Speed

- Speed is specified based on the type of movement
  - **Joint movements:** Percentage of the maximum rotational speed of the robot's axes
  - **Linear and circular movements:** Distance traveled by the tool point in a specific time (e.g., mm/s)
- Speed is set on a case-by-case basis, for example
  - Lower speeds for pick and place points
  - Higher speeds for approach and departure points
- Move command’s speed setting can be overridden, for example
  - With a separate command
  - Using the robot’s hand controller

SECLE The Service Centre for Continuous Learning and Employment
Funded by the European Union – NextGenerationEU

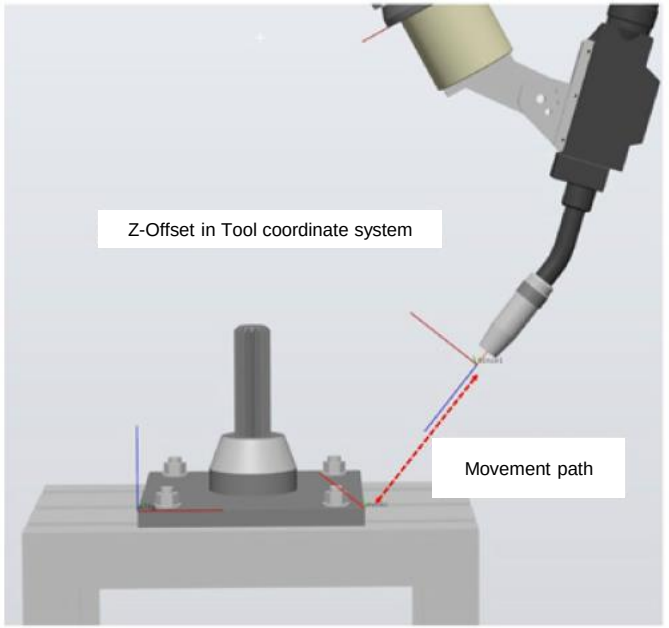
Approach data

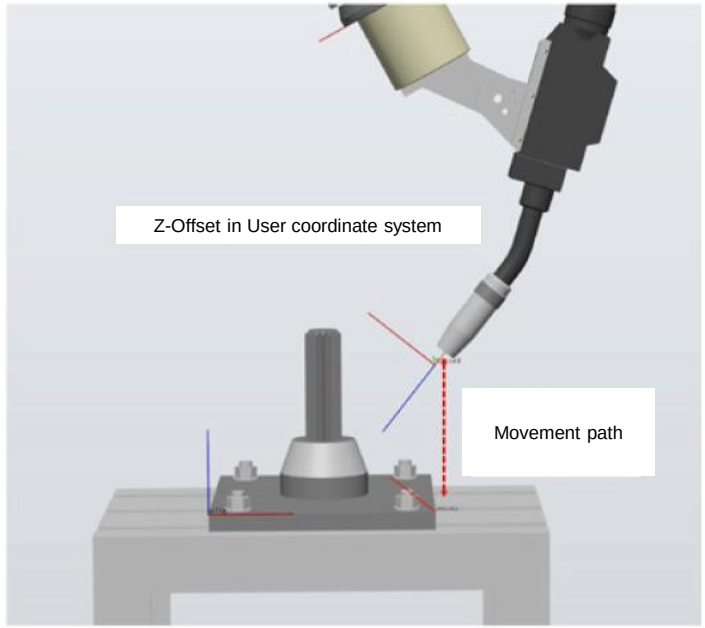
- Zone accuracy defines how accurately target point is reached
  - How much deviation is allowed in the robot's motion path
  - Useful for optimizing programs
- Zone accuracy is defined differently by each robot brand
- The Fine value is used to achieve precise positioning
  - Target point is reached accurately
  - Robot stops at the target point before moving to the next point
  - Fine values are used, e.g. at pick and place points
- Brand-specific values are used for path smoothing
  - Target point is not reached accurately
  - Robot does not stop at the target point before moving to the next point
  - For example: ABB’s Zone value, Fanuc's CNT value, and Yaskawa's PT value
  - Path smoothing can be used, e.g. at approach and departure points



Other arguments for motion commands

- Vary by robot brand
- Offset functions, i.e., position data translations
  - The robot's target point is shifted relative to the original point
  - Adds specified values to the original target point for location (X, Y, Z) and/or orientation (Rx, Ry, Rz)
  - Implemented in two ways:
    - Offset along tool coordinate system
    - Offset along user coordinate system





Other arguments for motion commands

- Acceleration/Deceleration
  - Adjusts the rate of change of the speed specified in the motion command when departing from or reaching target points
  - Control of acceleration/deceleration is necessary, e.g. when handling a workpiece
    - High acceleration/deceleration values may cause the workpiece to detach from the gripper

- Payload
  - Specifies the mass, center of mass, and any moments of the robot tool and workpiece
    - Important for motion control and accuracy
  - Different payloads apply when the gripper is empty versus when handling a workpiece
    - Incorrect payload settings may cause positioning inaccuracies

SECLE The Service Centre for Continuous Learning and Employment
Funded by the European Union – NextGenerationEU

Variables

- Variables in programming refer to a named, symbolic data storage that allows information to be retrieved (read) and stored (written)
- A variable has a symbolic name (identifier), a value, and a scope
  - Symbolic name is referenced during programming
  - Value can be changed during program execution
    - **Exception:** Constant variables, whose values cannot be changed
  - Scope defines the area within the program where the variable can be used
- Different types of variables exist depending on the robot brand, typically including:
  - Strings
  - Numeric variables (integer, float)
  - Boolean (true/false)
- Variables enables dynamic robot programming
  - Parametric programming
- Registers enable storing and reading values of variables and other programming components in lists/arrays

```
8   PROC Main()
9       init;
10      MoveAbsJ jHome,v1000,z50,tool0\WObj:=wobj0;
11      !Asking product lenght from operator
12      TPReadNum nProductLenght,"Enter product lenght (mm)";
13      !calculating  x-coordinate of pickpoint = products lenght/2
14      !This value is used in move instructions offset function.
15      nPick_X:=nProductLenght/2;
16
17      WHILE TRUE DO
18          WaitDI diPArtOnPos,1;
19          MoveJ Offs(pPick,nPick_X,0,200),v1000,z50,tSuction\WObj:=wConveyor;
20          MoveL Offs(pPick,nPick_X,0,200),v100,z50,tSuction\WObj:=wConveyor;
21          Set doVacuum;
22          MoveL Offs(pPick,nPick_X,0,200),v300,z50,tSuction\WObj:=wConveyor;
23          MoveAbsJ jHome,v1000,z50,tool0\WObj:=wobj0;
24          DropPart;
25      ENDWHILE
26  ENDPROC
```

The offset value of the picking point depends on the length of the product

Program flow commands

- Programs are rarely executed from start to finish in the written order of commands
  - Various program flow commands are used to control program execution
- **Conditional Statements**
  - Executing a specific part of a program until predefined conditions are met
  - Typically implemented with IF structures
    - IF
    - IF – ELSE
    - IF – ELSE IF – ELSE
- **Loops and Iterative Structures**
  - Repeating a specific part of a program
  - Section of the program can be repeated a set number of times, until a certain condition is met, or indefinitely
  - Various implementation options:
    - **FOR Loop**
      - The number of repetitions can be set as an absolute value, e.g., repeat a part of the program 10 times
    - **WHILE Structure**
      - The number of repetitions is tied to a condition, e.g., repeat until a certain variable changes from false to true
      - It is also possible to define an infinite loop for an indefinite period by setting the condition to "True"

```
PROC main()
    MoveJ pHome,v1000,fine,Pen_TCP\WObj:=wobj0;
    TPErase;
    TPWrite "";
    WHILE TRUE DO
        TPReadFK answer, "Select the product to be manufactured:",
            "Cylinder", "Circle", "Quit", stEmpty, stEmpty;
        IF answer = 1 THEN
            size_determination answer;
        ELSEIF answer = 2 THEN
            size_determination answer;
        ELSE
            TPErase;
            quit;
        ENDIF
        MoveJ pHome,v1000,fine,Pen_TCP\WObj:=wobj0;
        TPErase;
        WaitTime 1;
    ENDWHILE
ENDPROC
```


Funded by the European Union – NextGenerationEU

Program flow commands

- Program/Function Calls
  - Calling subprograms to execute the functionalities they contain
  - Programs can be called from main programs or from other programs
    - Various parameters can be passed to the subprogram in the function call
  - After the subprogram has been executed, program execution returns to the main program
    - The subprogram can also return values to the main program
- Jump Commands (Jump/Label)
  - Jump commands transfer program execution to a specific location (branch) within the program
    - A jump can be conditional, e.g., jump to line X if the value of a variable is true

SECLE
The Service Centre for Continuous Learning and Employment
Funded by the European Union – NextGenerationEU

Program flow commands

- Wait Commands
  - Suspend program execution temporarily
  - The duration of the pause can be predetermined or condition-based
    - Predetermined time, e.g., 1 second
    - Condition-based, e.g., waiting until a specific signal state changes to true
  - Delays are often needed in programming, for example, to accommodate pneumatic actuators
- Stop/Pause Commands (Halt/Break/Pause)
  - Designed to stop program execution and allow the program to resume from where it left off when restarted
- Exit/Abort Commands
  - Intended to stop program execution and terminate it without the option to resume from the point of interruption

Funded by the European Union – NextGenerationEU

Peripheral control commands

- Robots are always part of a production system, making communication with other devices, such as grippers, conveyors, programmable logic controllers (PLCs), and other robot cells, essential
  - Control of production cells can be centralized, for example, using PLCs
  - Various types of bus computing solutions and traditional I/O controls are available to facilitate communication
- Communication between the robot and peripheral devices typically uses I/O control
  - The **Set** command sets a selected output signal to a value of 1
  - The **Reset** command sets a selected output signal to a value of 0
  - The **WaitDI** command waits for a selected input signal to change to either 0 or 1
- Traditional I/O control is the simplest option for controlling the robot's peripheral devices
- If external axes are attached to the robot, e.g. if the robot is mounted on a linear track, their control is integrated into the robot's control system
  - In the case of a linear track, the track's servomotor acts as the robot's "7th axis"

Funded by the European Union – NextGenerationEU

Error Handling

- Preparing for errors and recovering from errors is one of the most challenging tasks in robot programming
  - In error situations, the program operates differently than usual → Causes challenges
- There are different methods for error handling
  - Safely suspending program execution and/or
  - Recovery actions to recover from the error
- There are various types of errors, such as missing sensor data
- When a predefined error is detected, an error handler is activated
- Different commands can be used in error handling, for example
  - The **WaitDI \ Maxtime** command can identify if a predefined signal does not respond within a specified time, triggering error handling in the program
  - **Error** program blocks can be used to transfer program execution to error handling
  - **Retry**, **Trynext**, and **Return** commands allow retrying the program section after error correction, moving to the next command after error handling, or returning to program execution after error handling
- Not all error situations can be automatically recovered
  - For example, emergency stops and collisions require the operator to perform recovery actions

```
PROC PickPart()
    !Routine for picking part from pile.
    MoveJ Offs(pPick,0,0,50),v200,fine,tSuction\WObj:=wTable;
    MoveL Offs(pPick,0,0,nZtrans),v200,fine,tSuction\WObj:=wTable;
    Set doVacuum;
    !Wait for signal from vacuum sensor. Max waittime 2sec.
    WaitDI diVacuumOn,1\MaxTime:=2;
    MoveL Offs(pPick,0,0,50),v200,fine,tSuction\WObj:=wTable;
    nZtrans:=nZtrans-nThickness;

    ERROR
    IF ERRNO=ERR_WAIT_MAXTIME THEN
        UIMsgBox "No vacuum, check vacuum gripper";
        RETRY;
    ENDIF
ENDPROC
```

If signal is not **1** after 2sec. Then execution jump to ERROR-handler

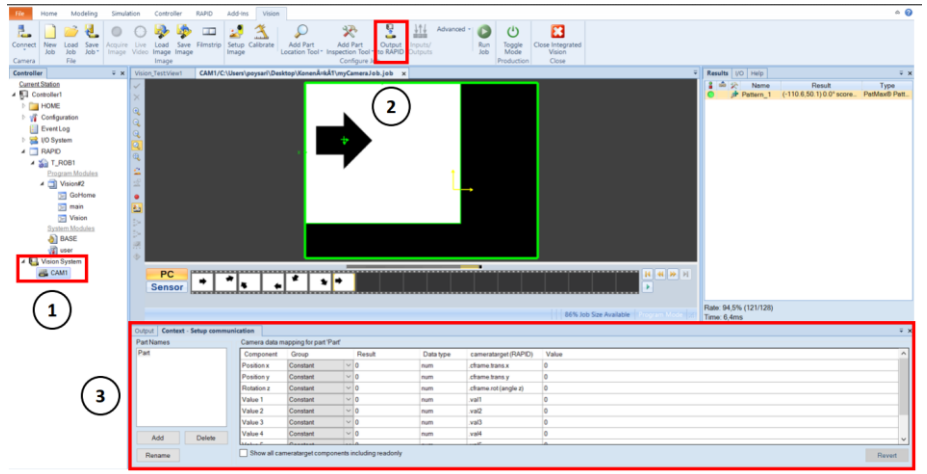
Program execution stops, info text appears to handling unit.

Operator approve error and program jump back and trying again.

Application-specific commands

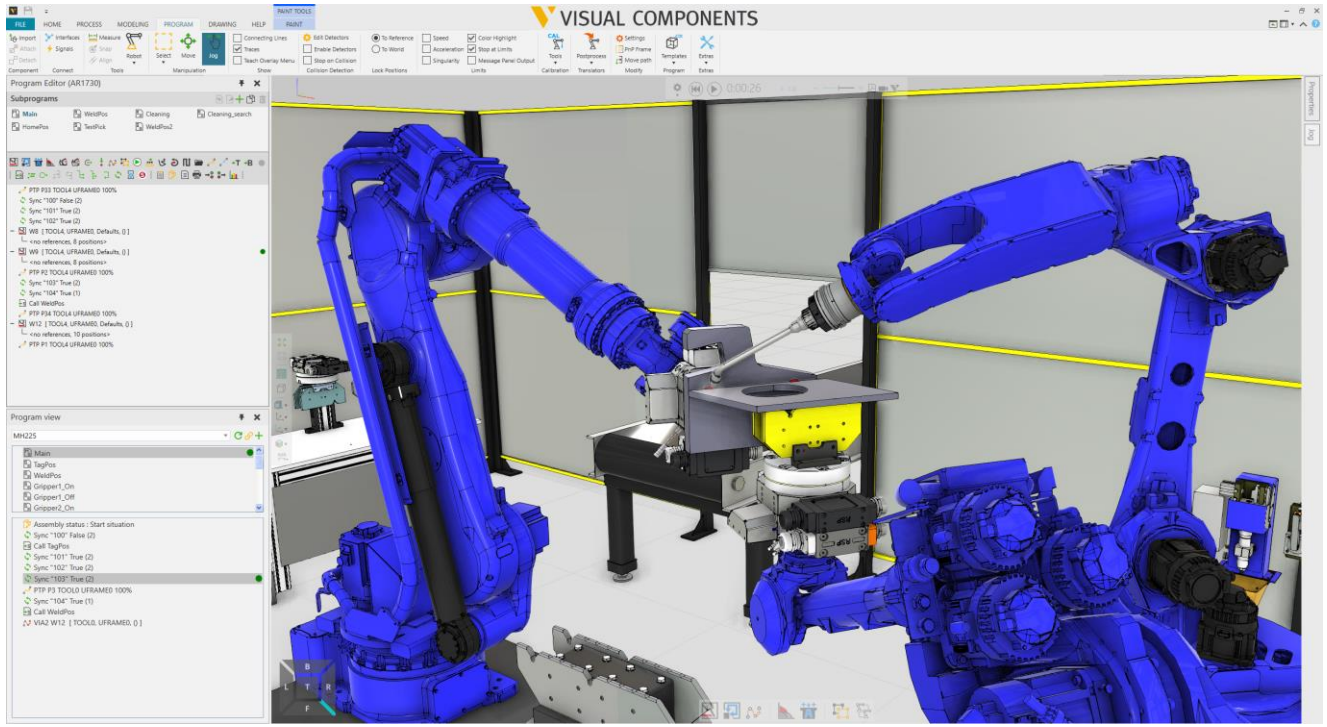
- Welding
- Painting
- 3D printing
- Palletizing
- Polishing
- Deburring
- Machine Vision
- Safety
- Etc.

```
PROC Vision()
  !Laitetaan kamera ohjelmatilaan. Vaihda tarvittaessa kameran nimi.
  CamSetProgramMode CAM1;
  !Lataa tehty konenäköohjelma.Vaihda tarvittaessa kameran nimi.
  CamLoadJob CAM1, myjob;
  !Aseta kamera päälle. Vaihda tarvittaessa kameran nimi.
  CamSetRunMode CAM1;

  !Siirretään robotti joko kuvauspisteeseen (jos kamera kiinnitettynä robottiin) tai sivuun kameran kuvausalueelta.
  MoveJ pCamera,v1000,fine,tool0\WObj:=wobj0;

  !Asetetaan alkuun, että kappaletta ei löytynyt.
  bPartFound:=FALSE;

  !While-silmukalla etsitään kuvaa.
  WHILE bPartFound = FALSE DO
      !Otetaan kameralla kuva. Vaihda tarvittaessa kameran nimi.
      CamReqImage CAM1;
      !Asetetaan muuttuja alustavasti todeksi, jos kuva saadaan otettua.
      bPartFound:=TRUE;
      !Jos ei saada kuvaa 2 sekunnin kuluessa siirrytään
      !virheenkäsittelyyn ja yritetään ottaa kuva uudestaan.
      CamGetResult CAM1, mycameratarget \MaxTime:=2;
  ENDWHILE

  !Siirretään otetusta kuvasta löydetyn kappaleen sijaintitiedot robotin työkoordinaatistoon.
  wCameraSystem2.oframe := mycameratarget.cframe;
```



Programming checklist

- Before Programming
  - Create process description
  - Define/review programming guidelines
  - Conduct preliminary reach and collision analysis
  - Define the workflow and program structure → Modularity and parametrization
  - Define variables and other programmable components, e.g. user and tool coordinate systems
  - Define preliminary communication with peripheral devices and error handling
- During Programming
  - Use correct and tested user and tool coordinate systems
  - Select appropriate motion types, speeds, and zone accuracies
  - Define points carefully
  - Check joint configurations and singularities
  - Add communication with peripheral devices and error handling
  - Program the entire workflow
  - Comment the program
- After Programming
  - Test and optimize the program (simulate if possible)
  - Calibration of robot system (for offline programming)
  - Commissioning
  - Documentation



Funded by the European Union – NextGenerationEU