Lecture 10 – Friday April 5, 2013

Programming

Some slides of this lecture are based material from the following books:

• J. Rehg. *Introduction to Robotics in CIM Systems*. 2nd Ed., Prentice-Hall, 1992.
• P. McKerrow. *Introduction to Robotics*. Addison-Wesley, 1991.
• C. Ray Asfahl. *Robots and Manufacturing Automation*. 2nd Ed., Wiley, 1992.

Objectives

When you have finished this lecture you should be able to:

- Understand the main requirements and functions of robot programming.
- Identify different programming methods commonly used with industrial robots.
- Understand robot language development and different classes of programming languages.
- Familiarize with ABB RAPID® (Robotics Application Programming Interactive Dialogue) Programming Language.

Outline

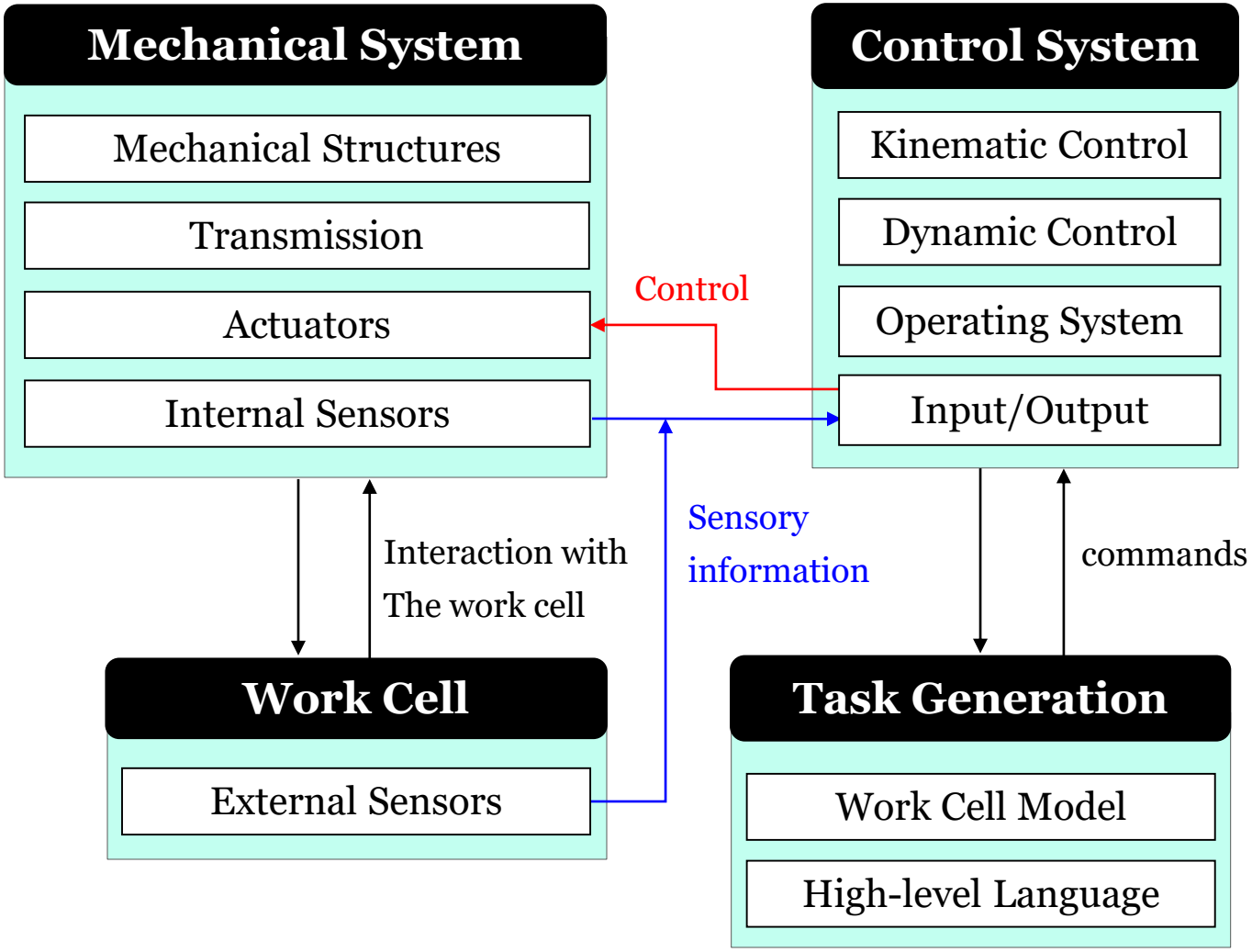
- Industrial Robot Programming
- Programming Methods
- Robot Language Development
- ABB RAPID
- Case Study: Inspection System
- Summary

Outline

- **Industrial Robot Programming**
- Programming Methods
- Robot Language Development
- ABB RAPID
- Case Study: Inspection System
- Summary

Industrial Robot Programming

- **Hardware and Software Structures**



Industrial Robot Programming

- **Software Characteristics**
  - ◇ Real-time System
  - ◇ Multi-axes System
  - ◇ Necessity for process synchronization
  - ◇ Grand Volume of Mathematical Calculations
  - ◇ Special Purpose Programming Language
  - ◇ High level of Security and Robustness

Industrial Robot Programming

- **Software Processes**
  - ◇ High Priority
    - – Safety Systems
    - – Emergency Stops
    - – Simultaneous Controllers of different axis (6 or more): 10-50 μsec per axis
  - ◇ Intermediate Priority
    - – Operating System
    - – Command Interpretation
    - – Path Calculation: 1-5 msec.
    - – I/O Communications
    - – Computer Link Communications.

Industrial Robot Programming

- **Software Processes**
  - ◇ Low Priority
    - – Interaction with Programming Unit
    - – Compiler
    - – Editor
    - – Communications with memory disks, diskette, etc.
    - – Variable Visualization
    - – Variable Simulation.

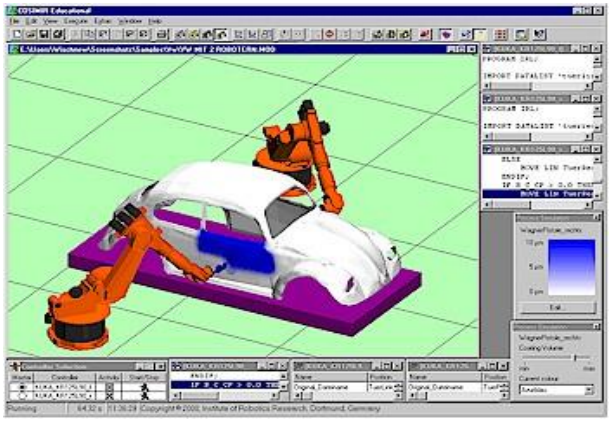
Industrial Robot Programming

- **Online Programming**

  Production **operation is stopped & programmer puts robot into programming mode** to teach the required position, motion & control sequences.

- **Offline Programming**

  All of the programming is performed **away from the robot & production area**; translation points are calculated by robot controller from translation point coordinate values entered in off-line mode; some touch-up may be required when testing the program in the operating robot.

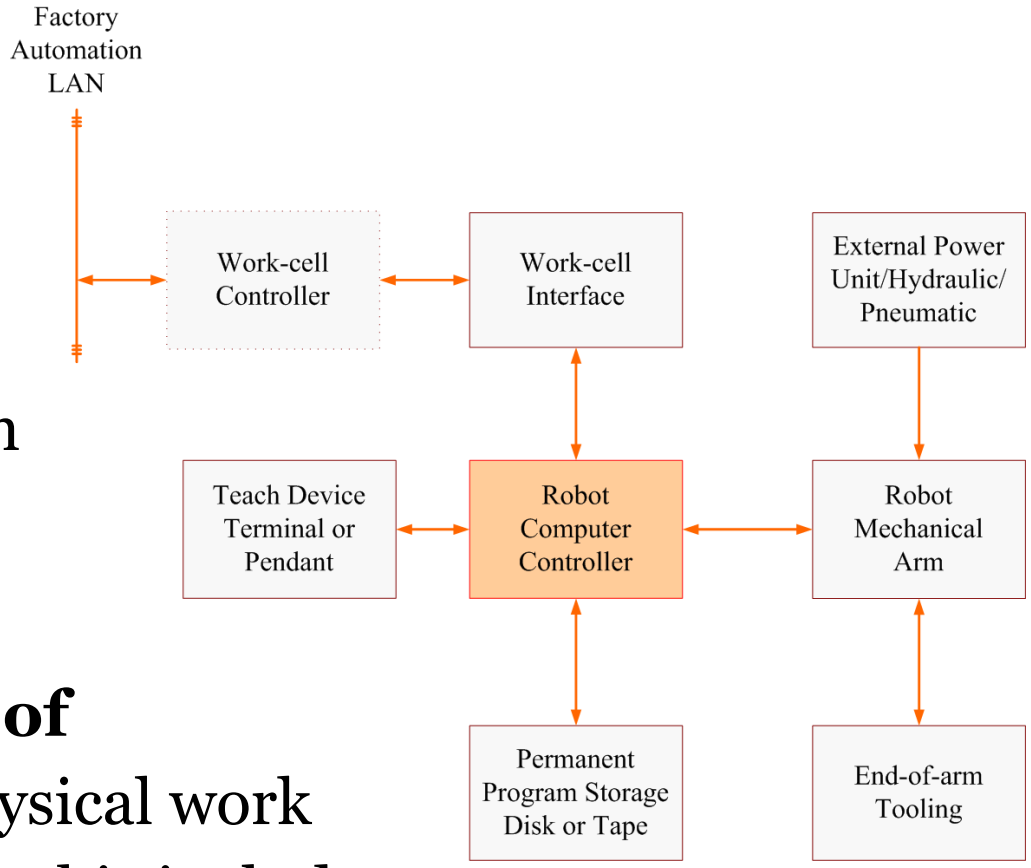


Industrial Robot Programming

- **Programming Main Functions**
  - ◇ Manipulation
  - ◇ Sensing
  - ◇ Intelligence
  - ◇ Data Processing

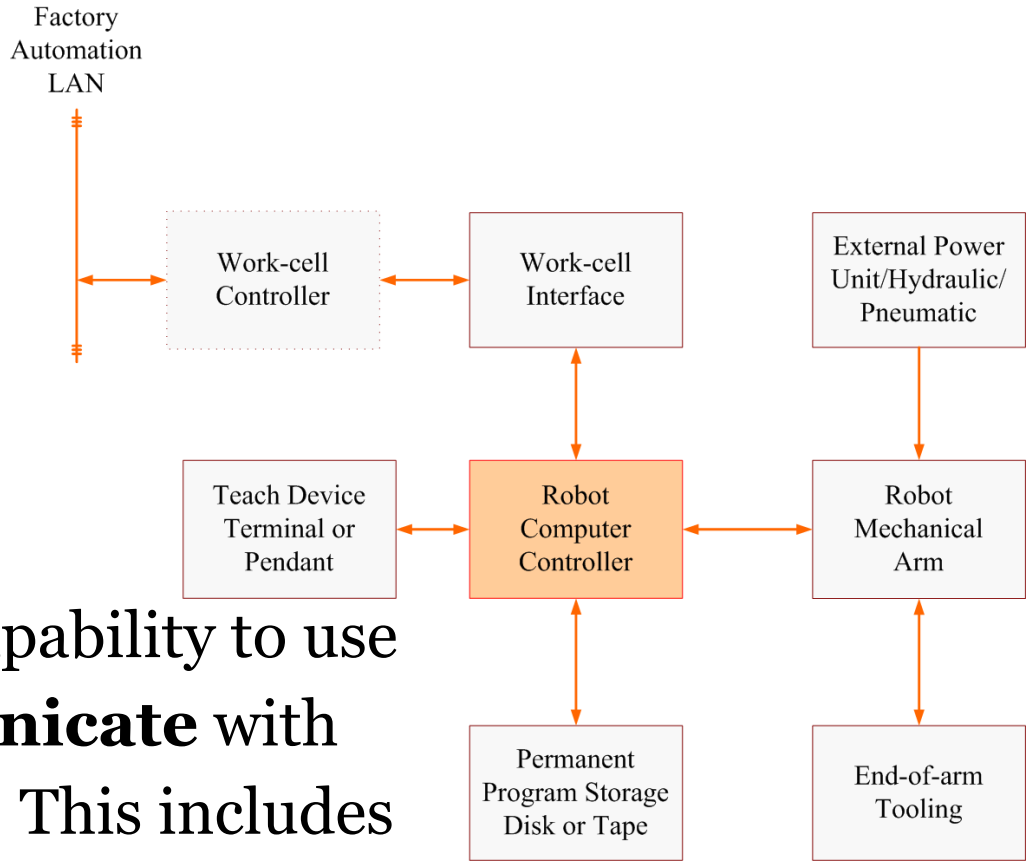
Industrial Robot Programming

- **Programming Main Functions**
  - ◇ **Manipulation:** The control of the **motion of all robot joints**. This includes position, velocity, and path control of the arm during all programmed motion.
  - ◇ **Sensing:** The **gathering of information** from the physical work cell surrounding the robot. This include the collection of sensory information and the control of peripheral equipment.



Industrial Robot Programming

- **Programming Main Functions**
  - ◇ **Intelligence:** The ability to **use gathered information** to modify system operation or to select various preprogrammed paths.



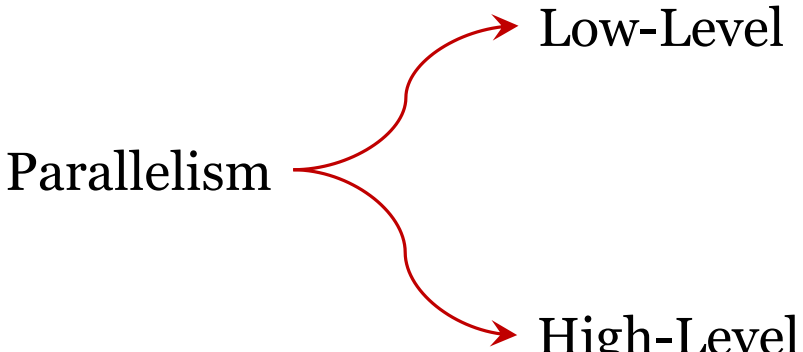
  - ◇ **Data Processing:** The capability to use data bases and to **communicate** with other intelligent machines. This includes the capability to **keep records, exchange programs, generate reports**, and **control activity in the work cell**.

Industrial Robot Programming

- **Software Requirements**
  - ◇ Motion Commands
  - ◇ Concurrency
  - ◇ Interprocess Communication
  - ◇ Event Synchronization
  - ◇ Polling or Interrupts
  - ◇ Sensor Variables
  - ◇ Initialization and Termination.

Industrial Robot Programming

- **Software Requirements**
  - ◇ **Motion Commands:** The control of motion requires the addition of **move commands** to the robot-level language, and the control of force requires the addition of **force commands**.
  - ◇ **Concurrency:**



At a low level, the programs must concurrently control the **joint motions** to achieve the desired Cartesian motion of the end-effector. These control programs must meet very tight timing constraints.

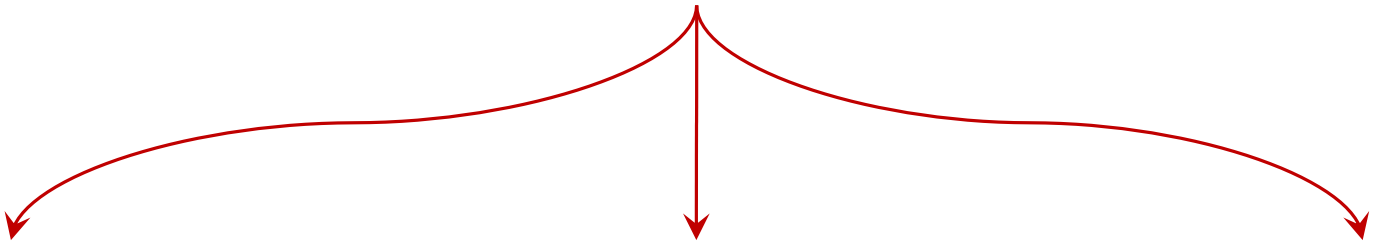
At a higher level, the robot operates in **parallel with other robots and the external sensors**. Data from these sensors is required by the robot-level programs for making decisions.

Concurrency can be achieved in three ways: **parallel processing**, **multitasking operating systems**, and **concurrent language constructs**.

Industrial Robot Programming

- **Software Requirements**
  - ◇ **Interprocess Communication:**

Interprocess or process-process Communication



Shared Memory

An area of memory is set aside for the storage of common data. This data must **be accessible** by several processes for both **read and write** operation. This is a **blackboard** technique.

Remote-procedure Call

Remote-procedure call (RPC) is a synchronous protocol that a process A (client) can use to request a service from process B (server) on a shared network. This synchronous operation requires the requesting process (client) to be suspended until the results of the remote procedure are returned.

Message Passing

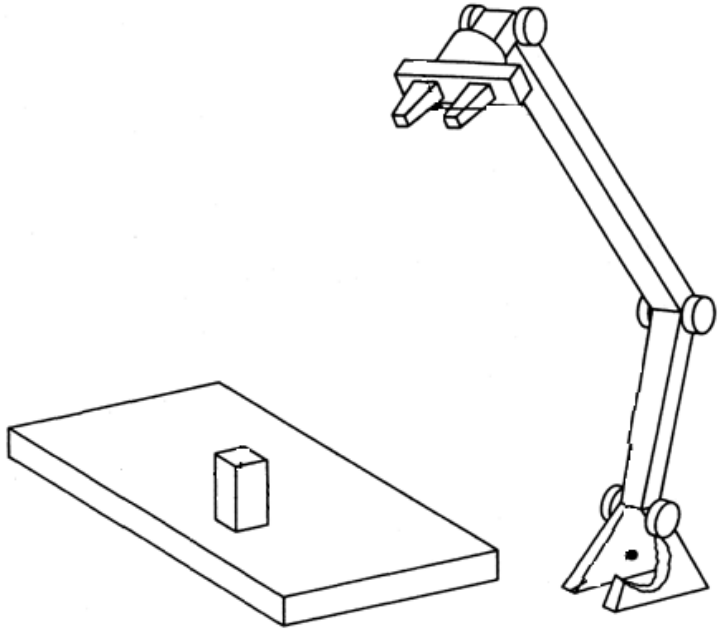
This is the simplest and safest model, where **processes send messages** to the other processes. Process **A** sends a message to process **B** and pauses until process **B** reads and acknowledge the message.

Industrial Robot Programming

- **Software Requirements**
  - ◇ **Event Synchronization:** because robots manipulate physical objects, the programs controlling the robot must be synchronized with events external to the computer.

    These events fall into three categories: **initiation events, termination events and error events**.

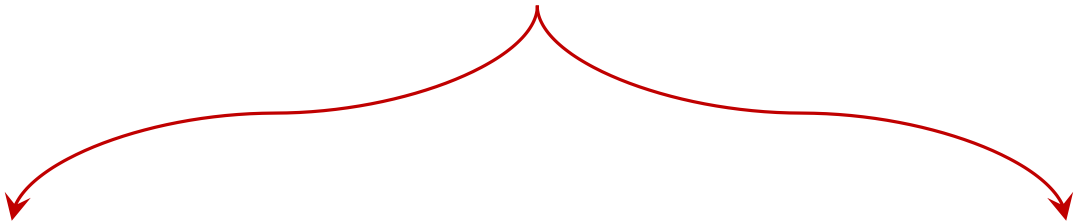
    For example, when command to grasp an object, the robot program should **check** that the object is grasped before proceeding. If the **grasp action failed**, an alternative course of action must be taken, such as calling an **error recovery routine**.



Industrial Robot Programming

- **Software Requirements**
  - ◇ **Polling or Interrupts:**

Event Detection Methods in Computing



Polling

Program **continually checks** the state of a sensor input. When certain predefined conditions occur, the polling program either sets a **flag** to indicate the occurrence of the event, or executes a procedure to **handle the event**

Interrupts

When an event occurs, an electrical pulse causes the **processor to halt** execution and vector to an interrupt-handling routine. As with polling, the interrupt routine can either set **a flag** or execute a procedure to handle the event.

Industrial Robot Programming

- **Software Requirements**
  - ◇ **Sensor Variable:** Sensor variables are used to store the values read from sensors. They are like ordinary variables in that they are used as loop guards in program constructs, for decision making, and in calculation.

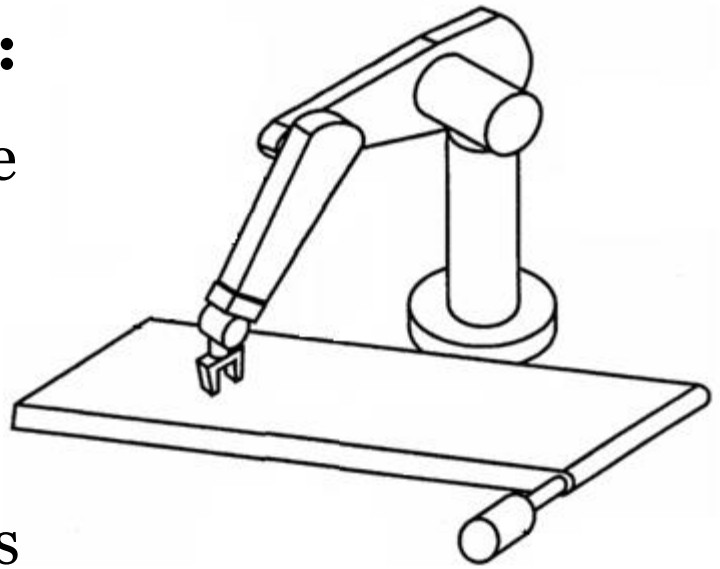
    However, they are sufficiently different from ordinary variables to require **special attention** in the language. Each sensor produces an electrical signal which may be a single bit, a binary number, or an array of binary numbers. Thus, the type of the **sensor variable** is determined by the **data structure required to store the sensed information**.

Industrial Robot Programming

- **Software Requirements**
  - ◇ **Initialization and Termination:** To reduce the problem of initializing a robot, we normally place it in a known home position.

When a **robot program terminates:**

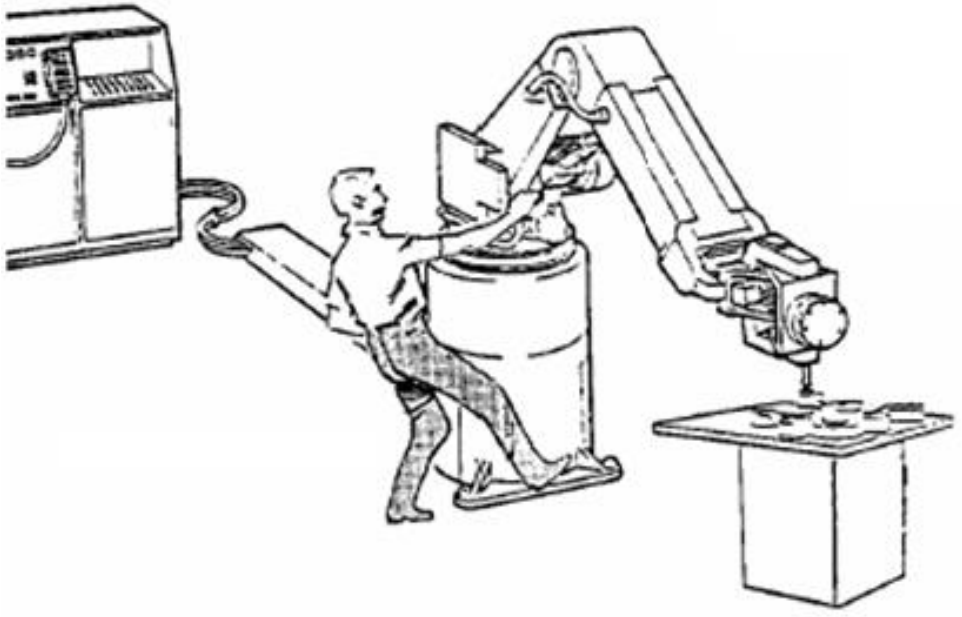
- It should move the robot to the home position,
- Apply the brakes,
- and zero all controller outputs by reducing the gain of the control loops to zero (**suiciding process**).



Industrial Robot Programming

- **Software Requirements**
  - ◇ **Initialization and Termination:**

    When a robot program **aborts**, a **potentially dangerous** situation occurs. **Applying the brakes** can cause an object to fly out of the gripper and **failure to apply the brakes** can allow the robot to crash into other objects.



Industrial Robot Programming

- **Software Requirements**
  - ◇ **Initialization and Termination:**

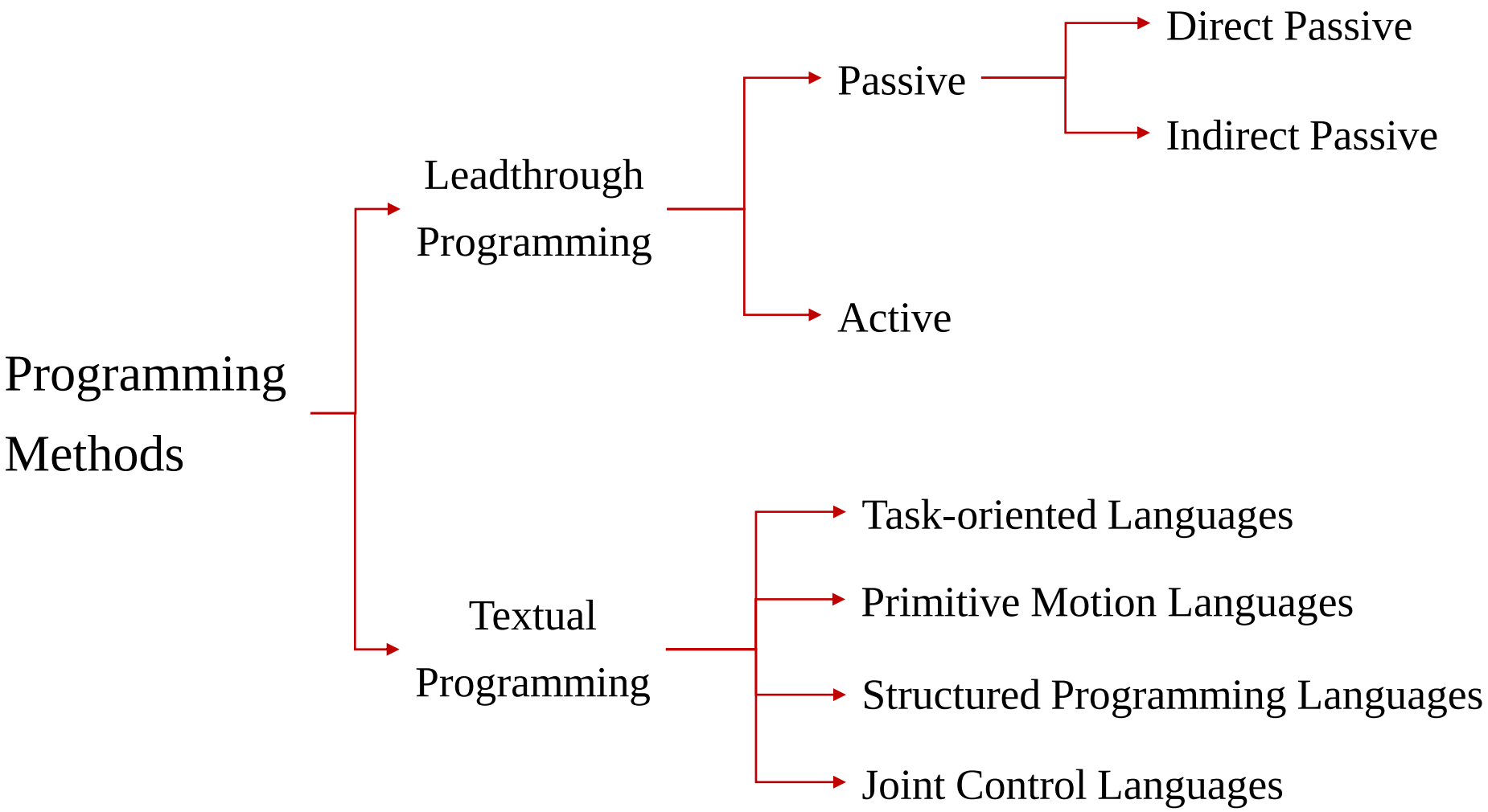
    Some commercial robots abort when passing near a **singularity** due to the **violence of the motion** about the singular point. If such a robot has **no brakes**, it will **collapse** in a heap on the objects it is manipulating.

    Watchdog timers to detect when operations have failed are required for the robot control system.

Outline

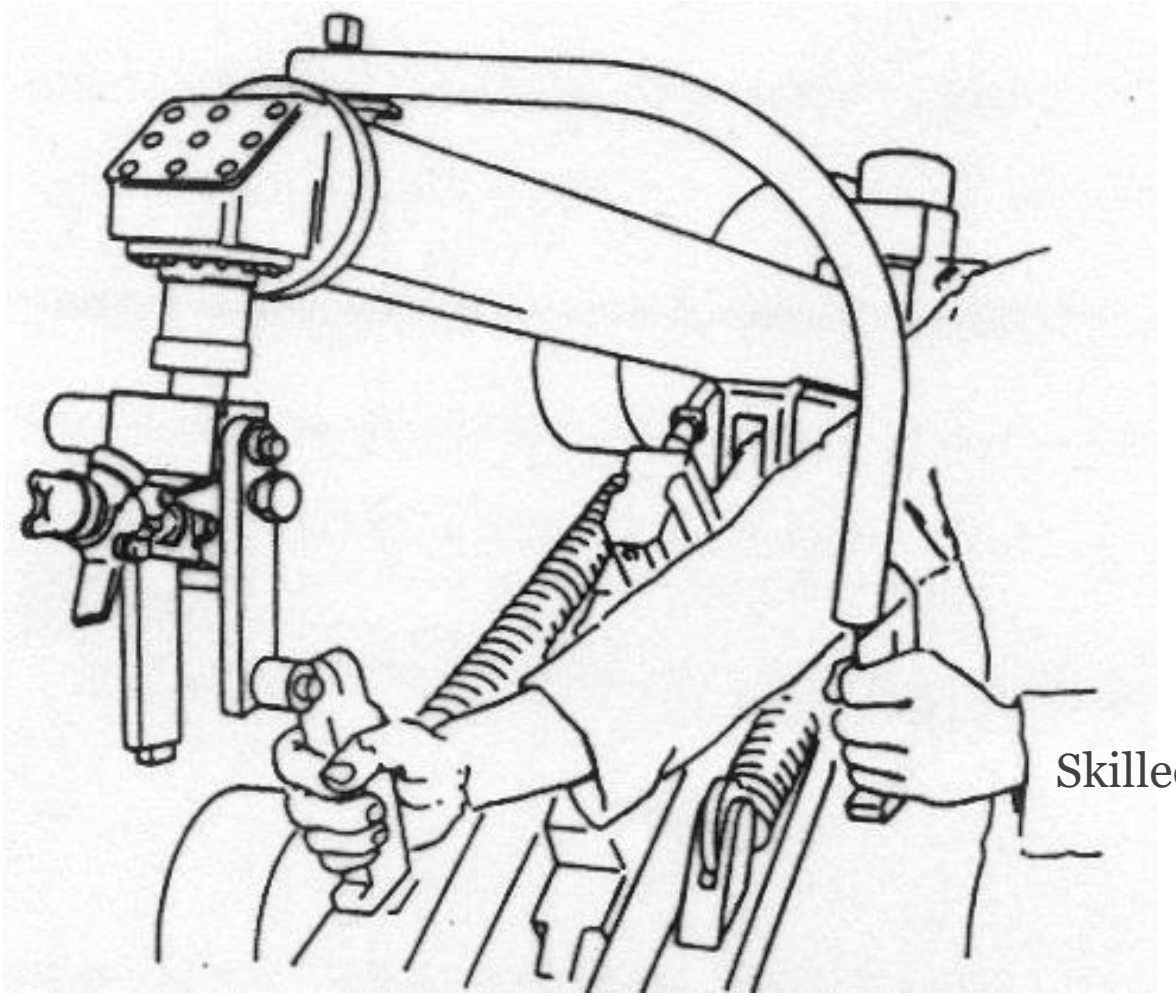
- Industrial Robot Programming
- **Programming Methods**
- Robot Language Development
- ABB RAPID
- Case Study: Inspection System
- Summary

Programming Methods



Programming Methods

- **Direct Passive Leadthrough Programming or Teaching**



Programming Methods

- **Direct Passive Leadthrough Programming or Teaching**

Robot is taught by manually seizing the end of the real robot arm and actually pushing it through the series of operations in a dry run of the real production process.

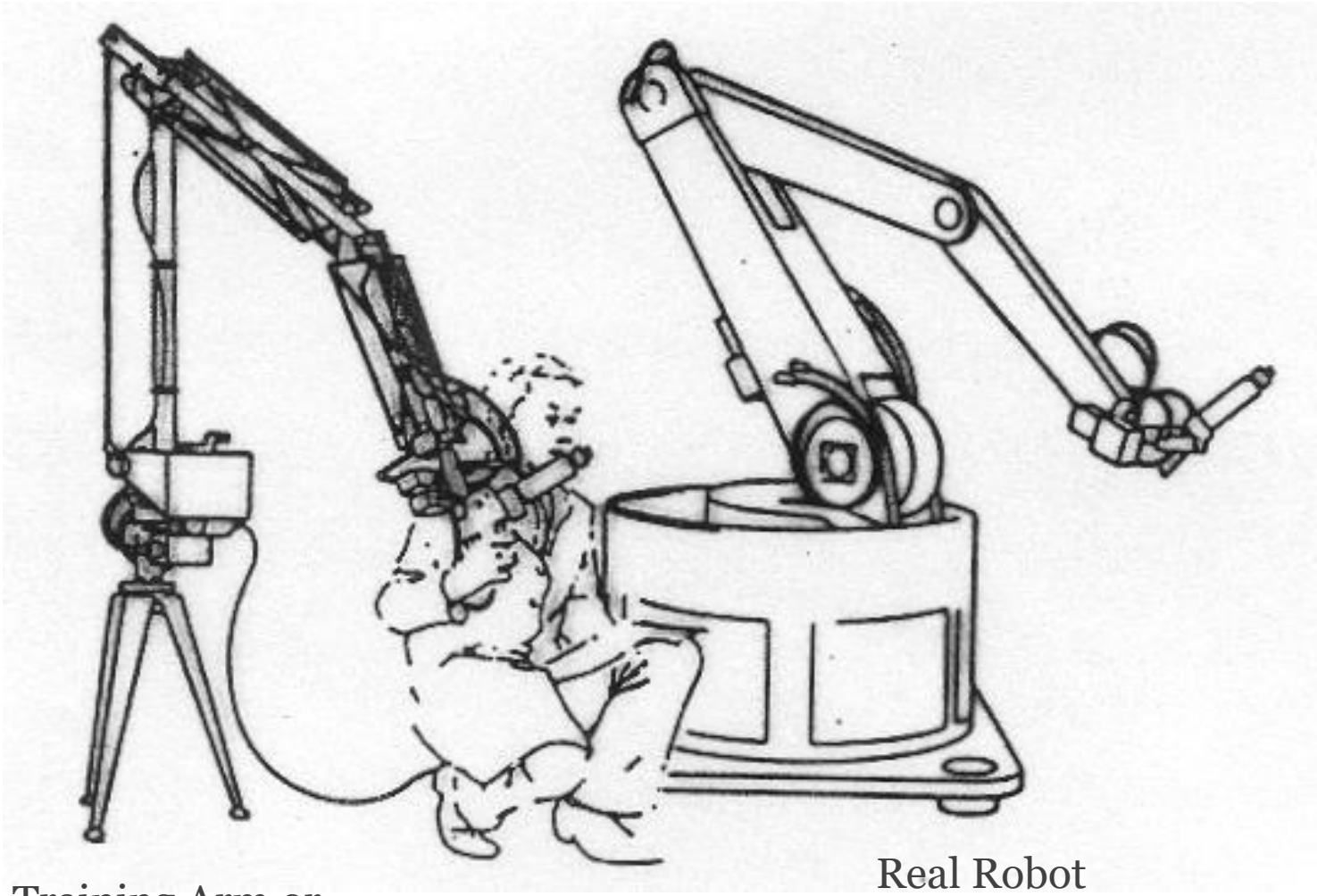
The robot can be commanded to repeat the performance indefinitely.

This type of programming is especially good for spray painting and welding applications. Operators skilled in conventional spray painting or welding can teach the robot their skills while simulating an actual manual performance of the job.

The robot then merely **mimics the actions of its teacher**.

Programming Methods

- **Indirect Passive Leadthrough Teaching**



Programming Methods

- **Indirect Passive Leadthrough Teaching**

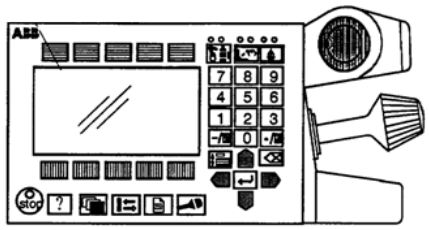
This method employs a mechanism that mechanically simulates the robot: a robot **training arm**, sometimes called a **dummy robot**. Compared with the real robot, the training arm is generally lighter and easier to manipulate by the skilled operator charged with the task of teaching the robot.

Spray painting is an ideal application for such mechanisms because the **skilled operator** must feel as though he or she is actually holding a paint gun while teaching the robot.

The comparatively light training arm can give that kind of feel to the operator. The training arm transmits its path to the control computer during the teach mode. In turn, the control computer drives the real robot through the same path of motion in the run mode.

Programming Methods

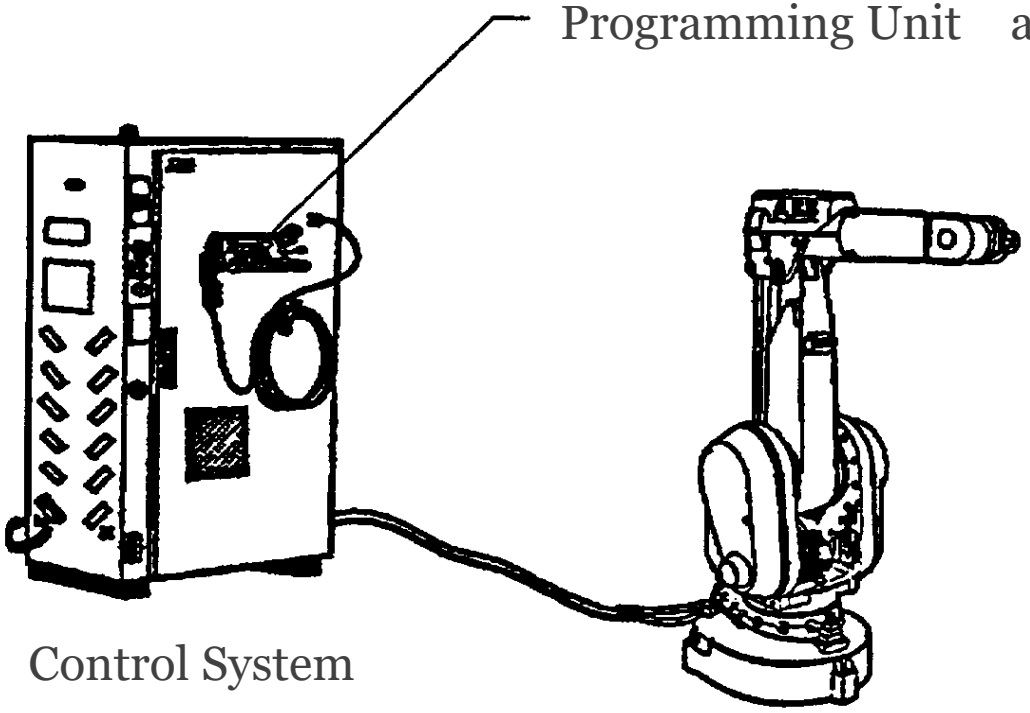
- **Active Leadthrough Programming (Teach-pendant Programming)**



Teach-Pendant or Teach Box

Teach-Pendants have controls for commanding the robot to memorize points along the path motion.

Programming Unit

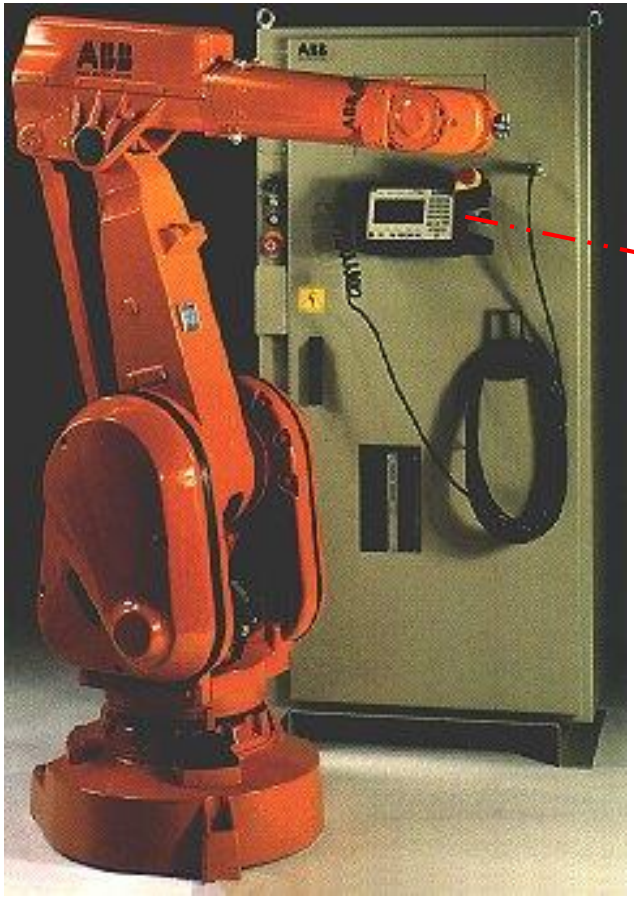


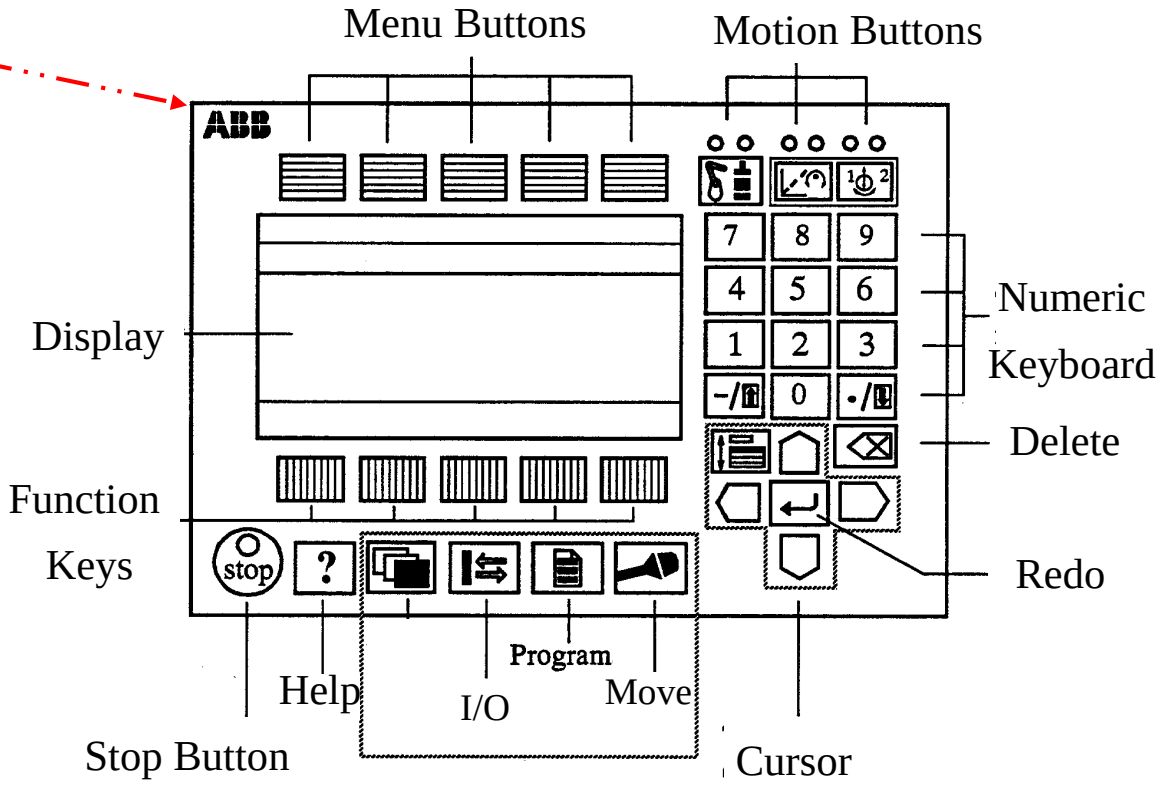
Control System

Manipulator

Programming Methods

- **Active Leadthrough Programming (Teach-pendant Programming)**





Programming Methods

- **Active Leadthrough Programming**

Mitsubishi RV-2AJ

Controller

Emergency button

Teach pendant

GUC-Festo Lab

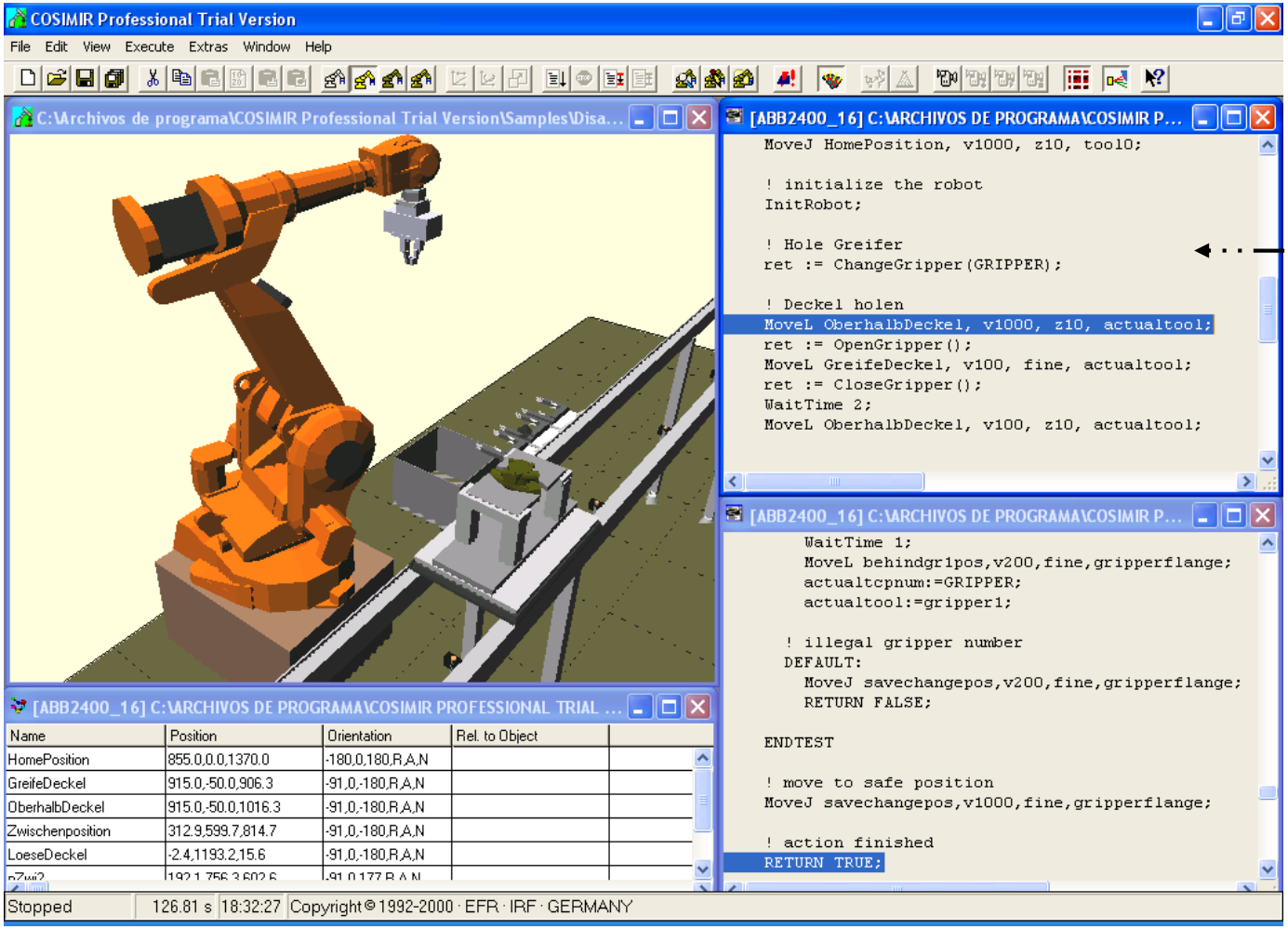
Programming Methods

- **Active Leadthrough Programming (Teach-pendant Programming)**

Fig. 2
El panel de control KUKA (KCP) con ratón 6-D integrado

Programming Methods

- **Textual Programming**



MELFA-BASIC Programming language

Outline

- Industrial Robot Programming
- Programming Methods
- **Robot Language Development**
- ABB RAPID
- Case Study: Inspection System
- Summary

Robot Language Development

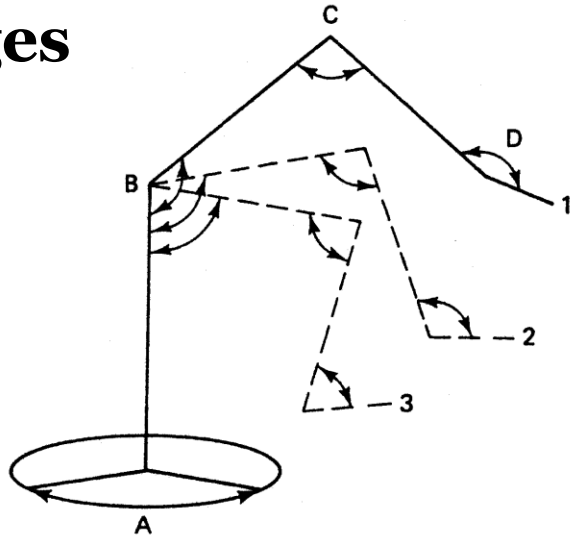
Level 4	Task-oriented Languages	Place object 1 on object2	AUTOPASS

Level 3	Structured Languages	If qmonitor eq 0 then
		begin
		.
		.
		else
			V, V+, HELP, KARL, AML, AML/E, MCL, DARL II, VAL II, RAPID, Melfa-Basic

Level 2	Primitive Motion Languages	Approach hole, 100 Move hole	RAIL, T3, RoboTalk, RPL, VAL
Level 1	Joint Control Languages	BASE CW 20	PLCs-controlled machines

Robot Language Development

- **Joint-Control Languages**
  - ◊ Languages at this level concentrate on the **physical control of robot motion** in terms of joints or axes.



Position	Joint angles			
	A	B	C	D
1	120°	130°	100°	160°
2	120°	100°	100°	110°
3	120°	80°	85°	70°

  - ◊ This level language requires the user to program in **joint space**. The term joint space means that all the programmed points in the robot's work envelope are expressed as a series of joint values for all the joints of the arm.

  - ◊ This level is used on some less sophisticated **point-to-point servo** machines and on all **stop-to-stop** pneumatic robot **controlled with PLCs**.

Robot Language Development

- **Primitive Motion Languages**

Level 2	Primitive Motion Languages	Approach hole, 100 Move hole	RAIL, T3, RoboTalk, RPL, VAL

◇ A program point is generated by **moving the robot** to a desired point and **depressing a program switch**. A sequence of points is saved in this manner, producing a complete program.

◇ Program **editing capability** is provided.

◇ **Teaching motion** of the robot is controlled by either a **teach pendant, terminal, or joystick**.

◇ The programmed and teaching motion can occur in the **Cartesian, cylindrical, or hand coordinate modes**.

Robot Language Development

- **Primitive Motion Languages**

Level 2

Primitive Motion Languages

Approach hole, 100
Move hole

- ◇ Interfacing with **work-cell** equipment is possible.
- ◇ The language permits **simple subroutines and branching**.
- ◇ **VAL** (**V**ictor **A**ssembly **L**anguage) is an example for this level.

VAL

```
PROGRAM DEMO
10 OPENI
20 SPEED 100 MMPS ALWAYS
30 MOVE #A
40 WAIT SIG(1001)
50 SPEED 80 MMPS
60 APPRO B,50
70 MOVES #B
80 BREAK
90 CLOSE
100 SPEED 80 MMPS
110 DEPARTS B,50
120 MOVE D
130 SPEED 80 MMPS
140 APPRO C,50
150 MOVES #C
160 OPENI
170 DEPARTS 50
END
```

Robot Language Development

- **Structured Programming Languages**

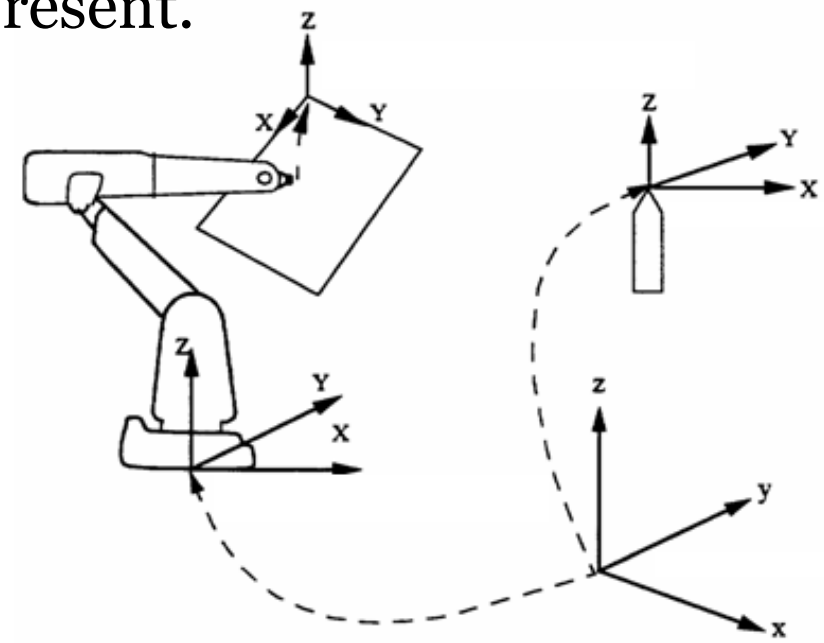
Level 3

Structured Languages

```
If qmonitor eq 0 then
begin
.
.
else
```

V, V+, HELP, KARL, AML, AML/E, MCL, DARL II, VAL II, RAPID, Melfa-Basic

- ◇ A **structured control** format is present.
- ◇ **Extensive use of coordinate transformations** and reference frames is permitted.
- ◇ **Complex data structures** are supported.



Robot Language Development

- **Structured Programming Languages**

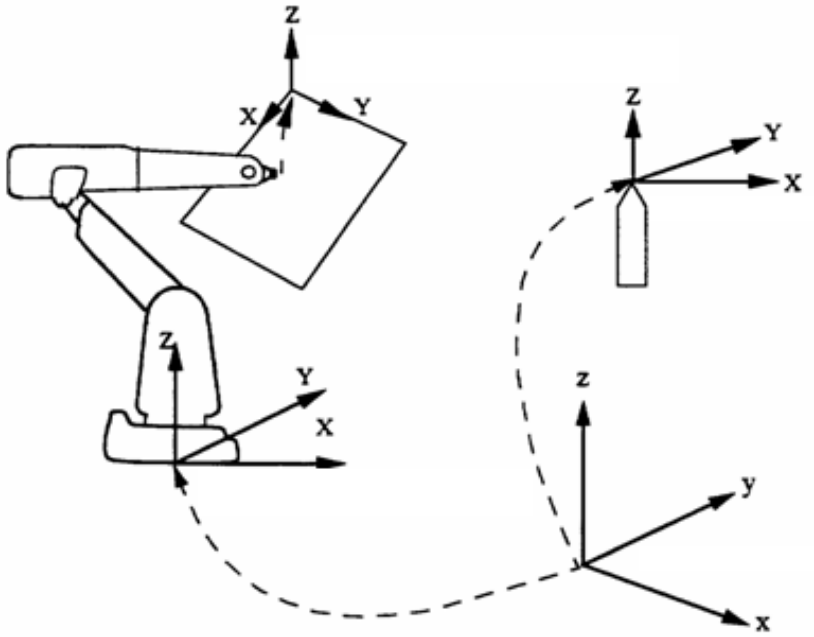
Level 3

Structured Languages

```
If qmonitor eq 0 then
begin
.
.
else
```

V, V+, HELP, KARL, AML, AML/E, MCL, DARL II, VAL II, RAPID, Melfa-Basic

- ◇ The format encourages **extensive use of branching and subroutines** defined by the user.
- ◇ **Off-line programming** is provided.



Robot Language Development

- **Task-oriented Languages**

Level 4 | Task-oriented Languages | Place object 1 on object2

AUTOPASS (Automatic Programming System for Mechanical Assembly)

◇ Programming in **natural language** is permitted. A natural language command might be "**Put bracket A on top of bracket B**".

◇ A plan generation feature allows replanning of robot motion to avoid undesirable situations.

◇ A world modeling system permits the robot to keep track of objects.

Robot Language Development

- **Task-oriented Languages**

Level 4 | Task-oriented Languages | Place object 1 on object2 | AUTOPASS (Automatic Programming System for Mechanical Assembly)

◇ The inclusion of **collision avoidance** permits accident-free motion.

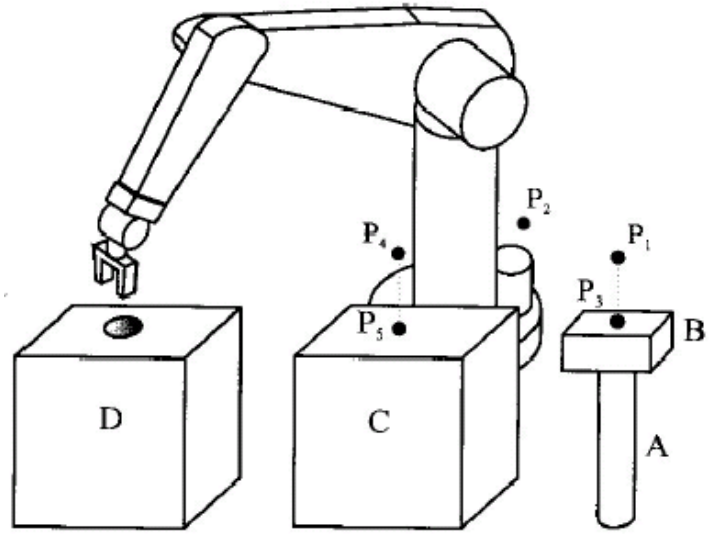
◇ **Teaching** can be accomplished by showing the robot an example solution.

Robot Language Development

Origin	Level 2	Level 3	Level 4
ABB		RAPID	
GMFanuc		KARL	
Kuka		KRL	
Cincinnati Milacron	T3		
Mitsubishi		MELFA-BASIC	
General Electric		HELP	
Automatix	RAIL		
IBM		AML, AML/E	AUTOPASS
Adept		V, V+	
McDonnel Douglas		MCL	
Rhino	RoboTalk		
Seiko		DARL II	
Westinhouse	RPL		
Unimation	VAL	VAL II	

Robot Language Development

- **Other Classification:**



Task Level	Assembly A with D	
Object Level	Situate B over C Situate A inside D . . .	MIT LAMA, IBM AUTOPASS, RAPT
Robot Level	Move P1 Open_gripper Wait 0.5 Set Input1 . . .	ABB RAPID, V+, LM, IBM AML, AL, VAL II

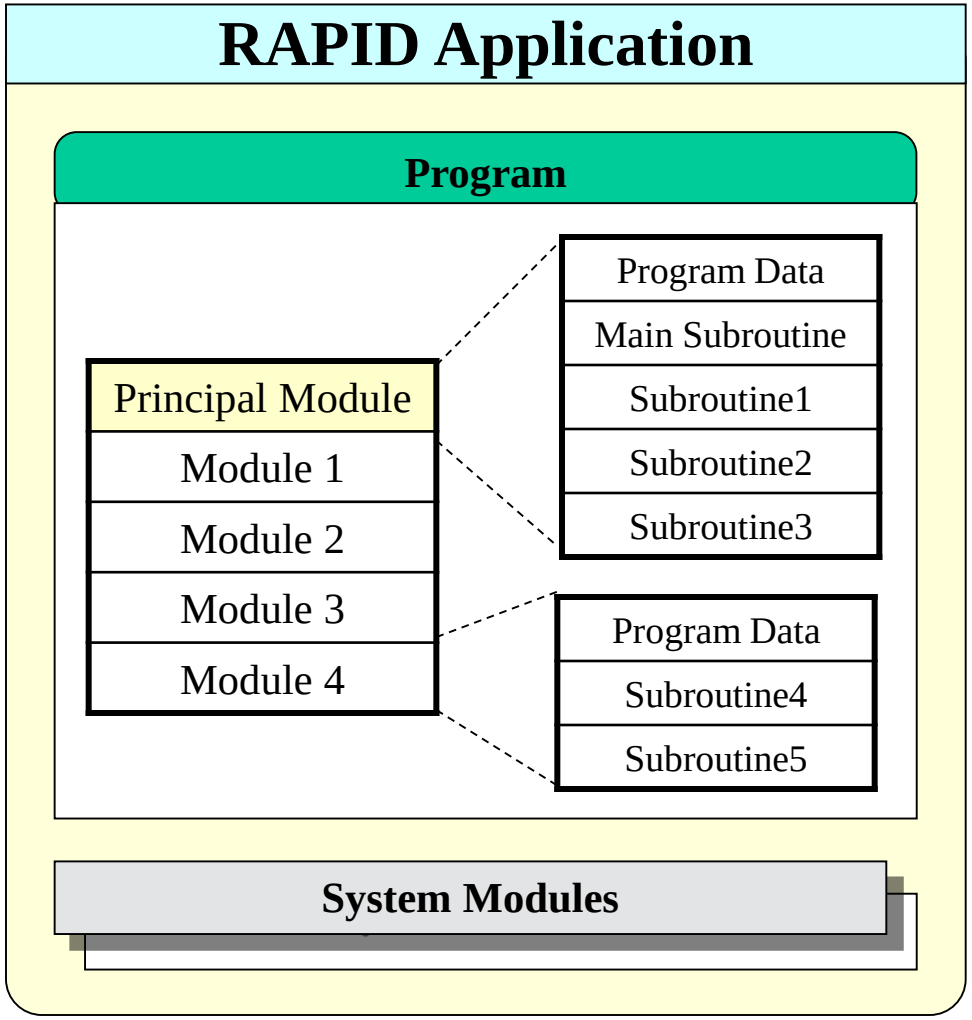
Outline

- Industrial Robot Programming
- Programming Methods
- Robot Language Development
- **ABB RAPID** For Reading
- Case Study: Inspection System
- Summary

ABB RAPID

- RAPID (**R**obotics **A**pplication **P**rogramming **I**nteractive **D**ialogue) is a level-3 textual programming language developed by ABB.

- A RAPID application consists of a program and a series of system modules.



ASEA Brown Boveri

ABB RAPID

- **Program**

```
%%%
VERSION:1
LANGUAGE:ENGLISH
%%%
MODULE Module1

  declarations

  PROC proc1()
     . . .
  ENDPROC

  PROC main()
    . . .
    . . .
  ENDPROC

ENDMODULE
```

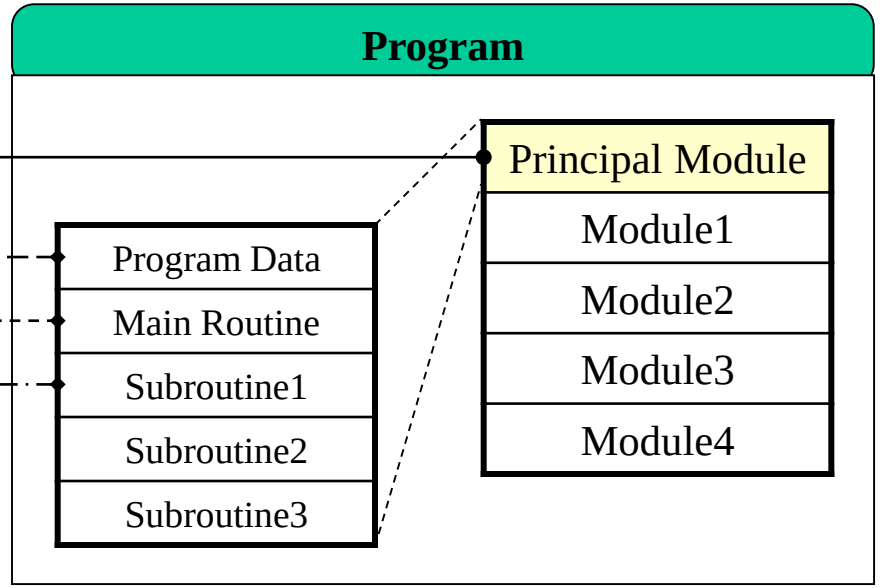


ABB RAPID

- **Program**

```
%%%
VERSION:1
LANGUAGE:ENGLISH
%%%
MODULE Example
  CONST robtarget A:=[[0,0,0],[0,0,0,0],[0,-1,0,0], [9E+09,...]];     !Load

  CONST tooldata gripper:= [TRUE, [[0,0,0],[1,0,0,0]],
                                  [0,[0,0,0],[1,0,0,0],0,0,0]];
  PROC close_gripper()
    Set sgripper;
  ENDPROC

  PROC pick_piece()
    MoveJ B1,v100,z5,gripper;
    MoveL B,v80,fine,gripper;
    close_gripper;
  ENDPROC
```

ABB RAPID

- **Program**

```
  PROC main()
    CONST dionum ready:=1;
    open_gripper;
    WHILE TRUE DO
      MoveJ A,v100,fine,gripper;
      WaitDI econtrol,ready;
      pick_piece;
      MoveL B1,v80,z5,gripper;
      MoveJ D,v100,z100,gripper;
      MoveJ C1,v100,z5,gripper;
      MoveL C,v80,fine,gripper;
      open_gripper;
      MoveL C1,v80,z5,gripper;
    ENDWHILE
  ENDPROC
ENDMODULE
```

ABB RAPID

- **Basic Elements**
  - ◇ **Identifiers:**

Example:

```
MODULE module_name
PROC routine_name()
VAR pos      variable_name:
```

- The first character must be a letter.
- Maximum length is 16.
- Case sensitive.
- Reserved Words:

AND	BACKWARD	CASE	CONNECT	CONST	DEFAULT	DIV
DO	ELSE	ELSEIF	ENDFOR	ENDFUNC	ENDIF	ENDMODULE
ENDPROC	ENDTEST	ENDTRAP	ENDWHILE	ERROR	EXIT	FALSE
FOR	FROM	FUNC	GOTO	IF	INOUT	LOCAL
MOD	MODULE	NOSTEPIN	NOT	OR	PERS	PROC
RAISE	READONLY	RETRY	RETURN	STEP	TEST	THEN
TO	SYSMODULE TRAP	TRUE	VAR	VIEWONLY	WHILE	
WITH	XOR					

ABB RAPID

- **Basic Elements**
  - ◇ **Data Types:**

Constants: (CONS)

Variables: (VAR)

Persistent: (PERS) can only be declared at module level, not inside a routine, and must be given an initial value. The initialization value must be a single value (without data or operands), or a single aggregate with members which, in turn, are single values or single aggregates.

String:

```
"This is a string"
```

Comments:

```
! This is a comment
```

ABB RAPID

- **Basic Elements**
  - ◇ **Data Types:**

Numeric value: num

Example:

```
VAR num flow := 0;
flow := 2.34;
```

Valid values: 5 0.37 0.1E-5 -12.34

Boolean value (True/False): bool

```
VAR bool <identifier>:= <valor>
                <value>: TRUE / FALSE
        <logic expression>
```

Example:

```
VAR bool open:=TRUE;
open:=FALSE;
open:= reg1 > 1;
```

String: string

Example:

```
VAR string text;
text:= "Start moving";
```

ABB RAPID

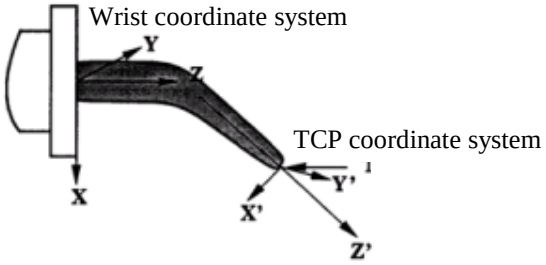
- **Basic Elements**
  - ◇ **Data Types:**

`pos`**:** a register represents only the position X, Y y Z in mm.

x is a numeric variable.

y is a numeric variable.

z is a numeric variable.



Example:

```
VAR pos position1;
position1 := [500, 0, 940];
position1.x := position1.x + 50;
```

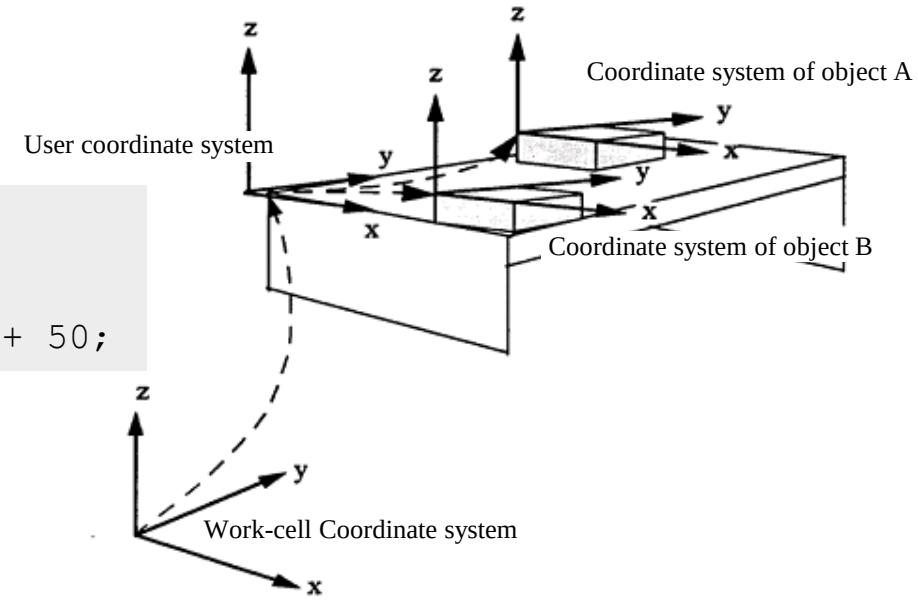
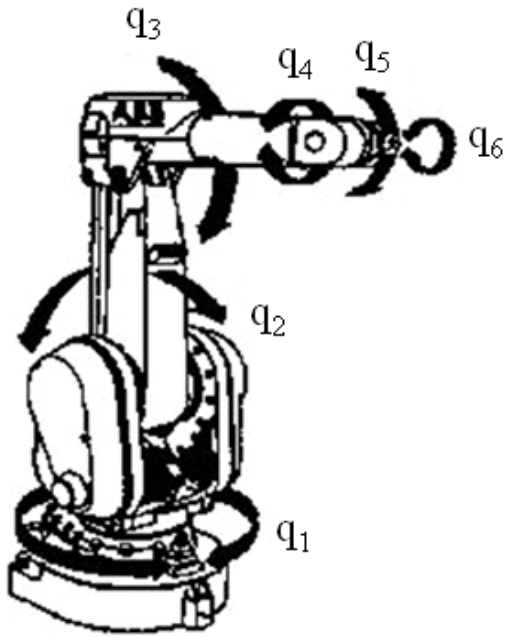


ABB RAPID

- **Basic Elements**
  - ◇ **Data Types:**

`orient`**:** a register to save the orientation (q1, q2, q3, q4,q5,q6).

`pose`**:** a register for both position and orientation.



Example:

```
VAR pose pos1;
pos1 := [[500, 100, 800],[1,0,0,0]];
pos1.trans := [650, -230, 1230];
pos1.trans.y := -23.54;
```

ABB RAPID

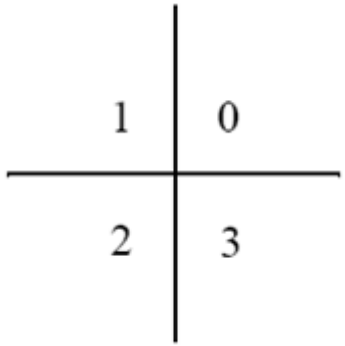
- **Basic Elements**
  - ◇ **Data Types:**

`confdata`**:** is used to define the axis configurations of the robot

cf1: the current quadrant of axis 1,

cf4: the current quadrant of axis 4,

cf6: the current quadrant of axis 6,



Example:

```
VAR confdata conf10:=[1,-1,0]
```

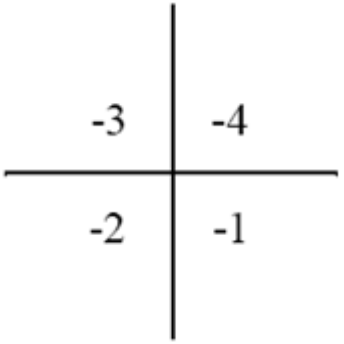


ABB RAPID

- **Basic Elements**
  - ◇ **Data Types:**

`loaddata`**:** is used to describe loads attached to the mechanical interface of the robot (the robot's mounting flange).

`mass`**:** weight in kg.

`cog`**:** centre of gravity of the load.

`aom`**:** The orientation of the axes of moment of the load at the centre of gravity.

`ix,iy,iz`**:** The moment of inertia of the load around x, y, z in kgm^2.

Example:

```
VAR loaddata piece:=[5,[50,0,50],[1,0,0,0],0,0,0];
```

ABB RAPID

- **Basic Elements**
  - ◇ **Data Types:**

`tooldata`**:** is used to describe the characteristics of a tool, e.g. a welding gun or a gripper.

`robhold` **:** bool to show of the tool is fixed or not.

`tframe` **:** Coordinate system of the tool

TCP Position (x,y,z)

Orientation. (q1,q2,q3,q4)

`tload`**:** load of the tool

Example:

```
PERS tooldata gripper:=[TRUE,[[97,0,220],
[0.924,0,0.383,0]],5,[-23,0,75],[1,0,0,0],0,0,0]]
```

ABB RAPID

- **Basic Elements**
  - ◇ **Modules:**

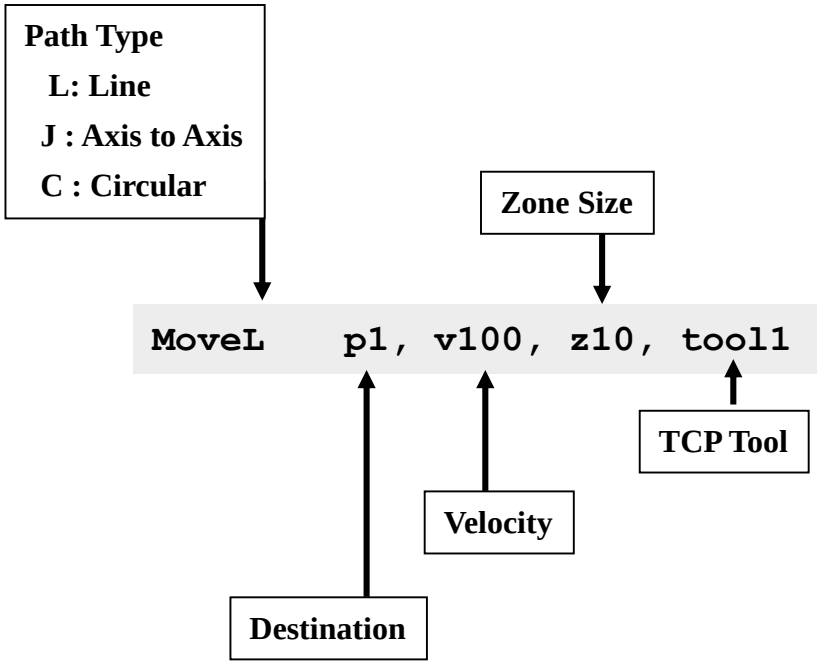
Declaration:

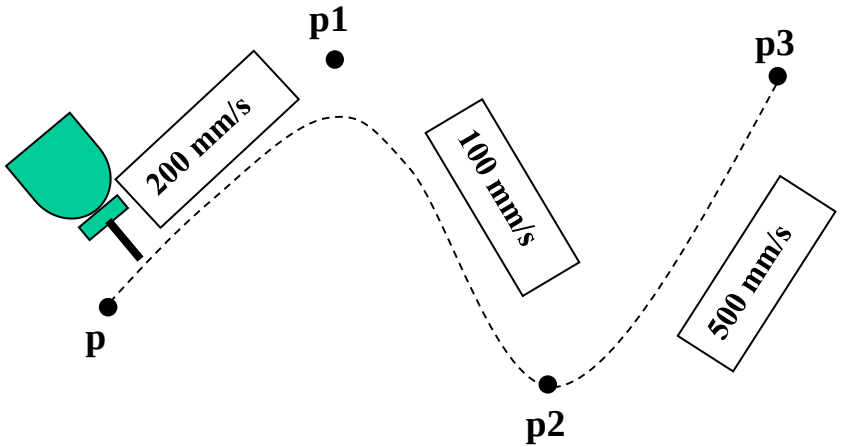
```
MODULE          <module_name> [<List of attributes>]
                <Declaration>
                <Routine Declaration>
ENDMODULE
```

```
[<List of attributes>]:
    SYSMODULE
    NOSTEPIN
    VIEWONLY
    READONLY
```

ABB RAPID

- **Basic Elements**
  - ◇ **Motion Commands:**

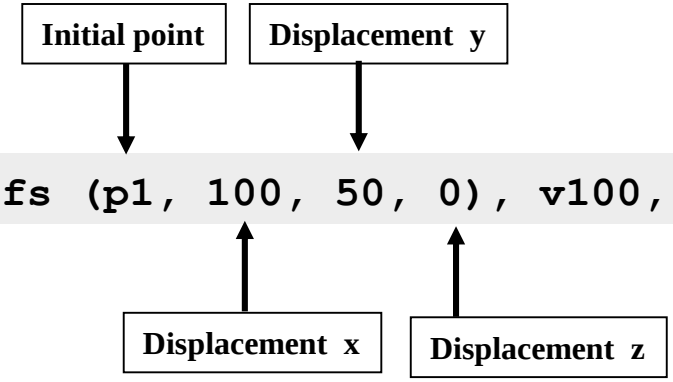


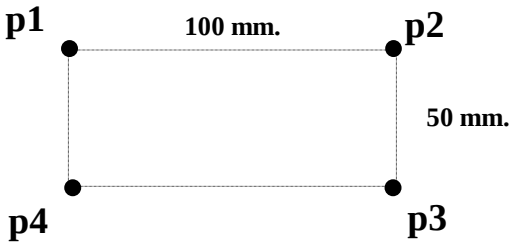



```
MoveL  p1, v200, z10, tool1
MoveL  p2, v100, fine, tool1
MoveJ  p3, v500, fine, tool1
```

ABB RAPID

- **Basic Elements**
  - ◇ **Motion Commands:**





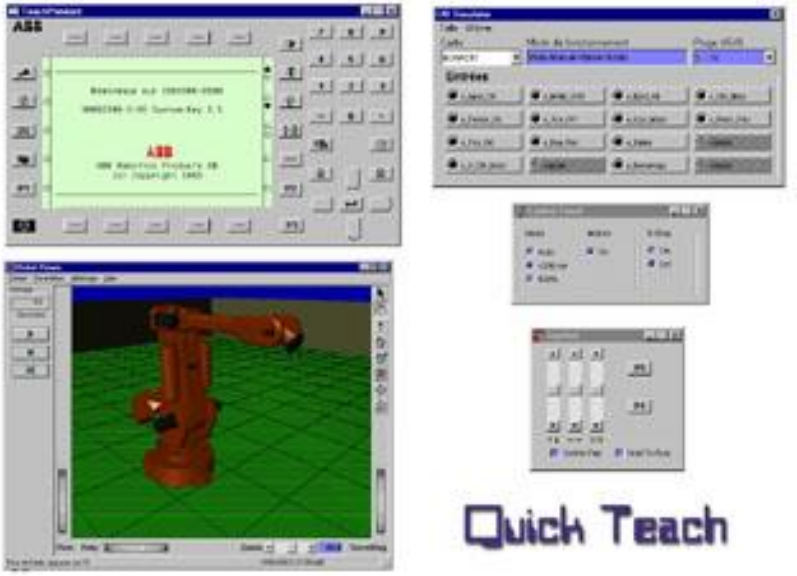

```
MoveL  p1, v200, fine, tool1
MoveL  Offs (p1, 100, 0, 0 ), v100, fine, tool1
MoveL  Offs (p1, 100, 50, 0 ), v100, fine, tool1
MoveL  Offs (p1, 0, 50, 0 ), v100, fine, tool1
MoveL  p1, v100, fine, tool1
```

More information: ABB RAPID Reference On-line Manual

ABB RAPID

- **ABB**

RAPID SyntaxChecker



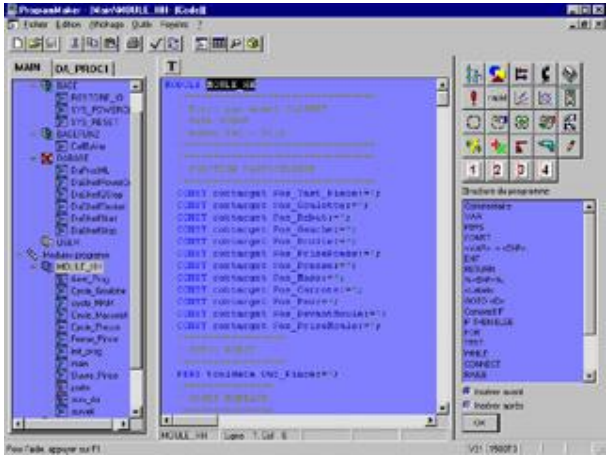
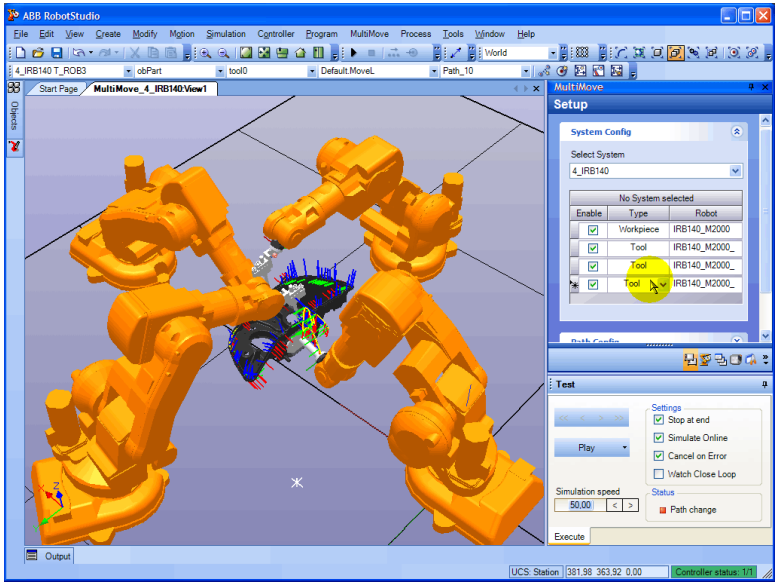


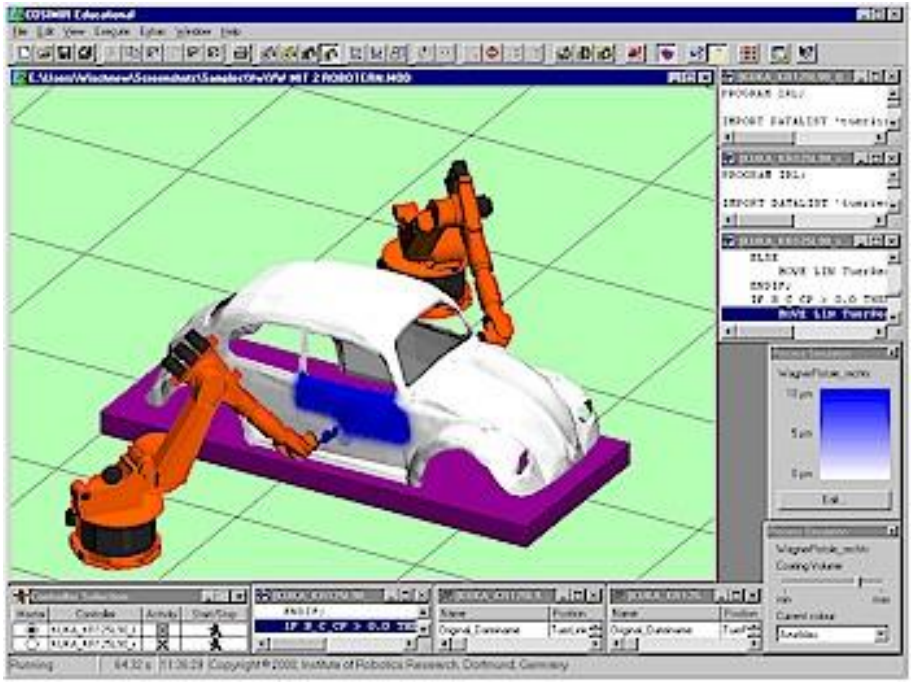
ABB Deskware

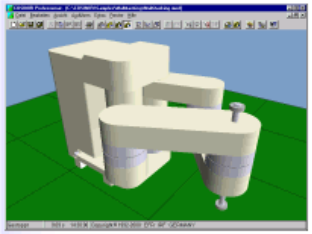


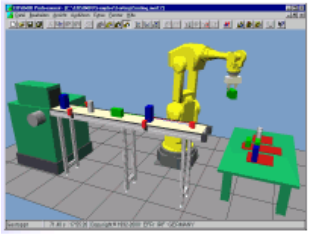
RobotStudio™

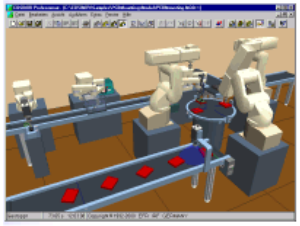
ABB RAPID

- **FESTO COSIMIR ®**







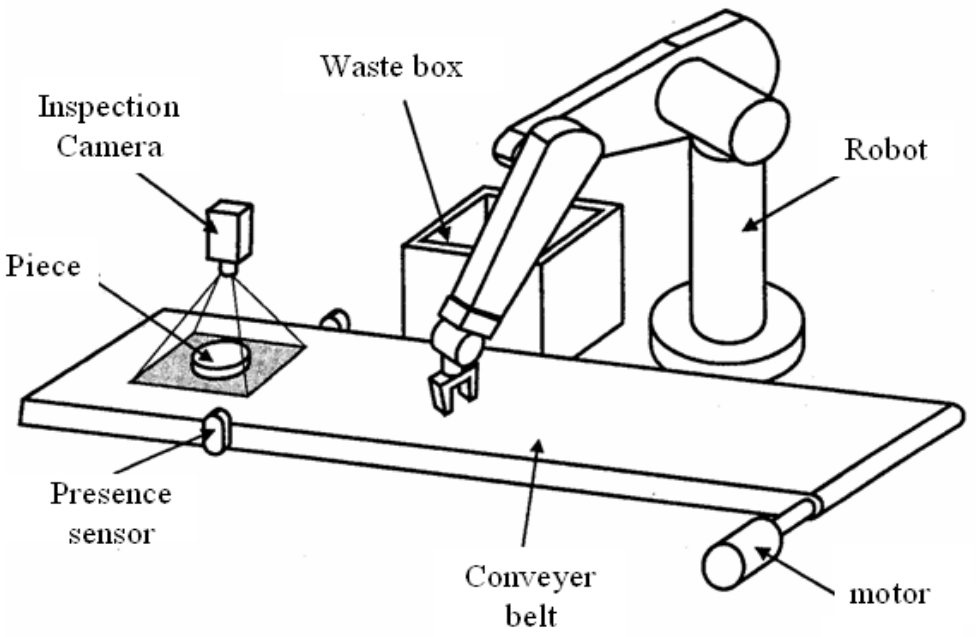


Outline

- Industrial Robot Programming
- Programming Methods
- Robot Language Development
- ABB RAPID
- **Case Study: Inspection System**
- Summary

Case Study: Inspection System

1. The robot **waits** until receiving a signal from the presence sensor, which indicates the existence of a workpiece over the conveyer belt.
2. The robot **stops** the conveyer belt and picks the defected piece and deposits it in the waste box.



3. The robot **reactivates** the movement of the conveyer belt after depositing the defected piece.
4. After the operation, the robot **returns** to its initial position and the cycle repeats itself again.

Case Study: Inspection System

- **Variable Declaration**

tool: a *tooldata* variable, which represents the gripper of the robot.

piece: a *loaddata* variable to define the piece to be manipulated.

conf_wait: a *robtarget* variable, which defines axis configurations of the robot in the waiting position.

```
PERS tooldata tool:=[TRUE,[[97,0,223],
       [0.924,0,0,0.383,0]],[5,[-23,0,75],[1,0,0,0],0,0,0]]
PERS loaddata piece:=[5,[50,0,50],[1,0,0,0],0,0,0];
```

```
VAR robtarget conf_wait:=[[600,500,225],[1,0,0,0],[1,0,0,0],
                  [9E9,9E9,9E9,9E9,9E9,9E9]];
```

```
VAR signaldo gripper          !activation signal of the gripper
VAR signaldo activate_belt    !activation signal of the conveyer belt
VAR signaldi defected_piece   !signal of defected piece
VAR signaldi finish           !signal to end the program
```

Case Study: Inspection System

- **Gripper Control Subroutines**

```
PROC pick()
  Set gripper            !close gripper
  WaitTime 0.3           !Wait 0.3 seconds
  GripLoad piece         !Indicating that the piece is picked
ENDPROC

PROC place()
  Reset gripper          !Open gripper
  WaitTime 0.3           !Wait 0.3 seconds
  GripLoad LOAD0         !Indicating that there is no piece
ENDPROC
```

Case Study: Inspection System

- **Pick Subroutines**

```
PROC pick_piece()
  MOVEJ *,VMAX,z60,tool    !move the robot quickly to a certain point
  MOVEL *,V500,z20,tool    !move the robot in straight line
  MOVEL *,V150,FINE,tool   !go down with maximum resolution
  pick                     !pick the piece
  MOVEL *,V200,z20,tool    !go up with piece taken
ENDPROC
```

Case Study: Inspection System

- **Place Subroutines**

```
PROC place_piece()
  MOVEJ *,VMAX,z30,tool   !Move to the waste box
  MOVEJ *,V300,z30,tool
  place                   !place the piece
ENDPROC
```

Case Study: Inspection System

- **Go to Waiting Position Subroutine**

```
PROC go_wait_position()
  MOVEJ conf_wait,VMAX,z30,tool  !Move to the initial position
ENDPROC
```

Case Study: Inspection System

- **Main Program**

```
PROC main()
  go_wait_position;                    !Move to initial position
 WHILE Dinput(finish)=0 Do             !wait end program signal
  IF Dinput(defected_piece)=1 THEN     !Wait defected piece signal
  SetDO activate_belt,0;               !Stop belt
  pick_piece                           !Pick the defected piece
  SetDO activate_belt,1;               !Activate the belt
  place_piece                          !Place the defected piece
  go_wait_position;                    !Move to the initial position
  ENDIF
 ENDWHILE
ENDPROC
```

Outline

- Industrial Robot Programming
- Programming Methods
- Robot Language Development
- ABB RAPID
- Case Study: Inspection System
- **Summary**

Summary

- Industrial Robot is a **reprogrammable** multifunctional manipulator designed to move material, parts, tools or specialized devices through variable programmed motions for the performance of a variety of tasks. A key feature of robots is their capability for being **reprogrammed** for different tasks.
- Robot programming places **special requirements** on computer languages and systems.
- In addition to the data manipulation handled by normal programs, robot programs have to control motion, operate in parallel, communicate with programs which may be in other computers, synchronize with external events, respond to interrupts in real time, operate on sensor variables, and initialize and terminate in **physically safe ways**.

End of the course

Best wishes!