Tallinn Technical University
Department of Electrical Drives and Power Electronics

TÕNU LEHTLA

INTRODUCTION TO ROBOTICS

Tallinn
2008

T. Lehtla. Introduction to robotics. TTU, Dept. of Electrical Drives and Power Electronics. Tallinn, 2008. 96 p.

Edited by Mare-Anne Laane

Content

1. INTRODUCTION

1.1. General principles of robot control

A robot is the main component of a flexible production system (*FPS*). Other components of this system are machine tools, transport machines, control devices, and different auxiliary elements. A flexible production system is an automatically operating production system that can be easily reprogrammed and adapted to manufacture different products. Robot centered modules of FPS, called robot modules or robot systems are intended for specified technological operations like welding, surface coating, packaging, etc. The robot module includes one or more robots (with manipulators and control devices), pallets for details or products, auxiliary positioning, transport devices, etc. Therefore, robot control means control of a complete robot module and a certain part of the production process. Fig. 1.1 shows main hardware and software components of the IRB1600 robot from ABB.

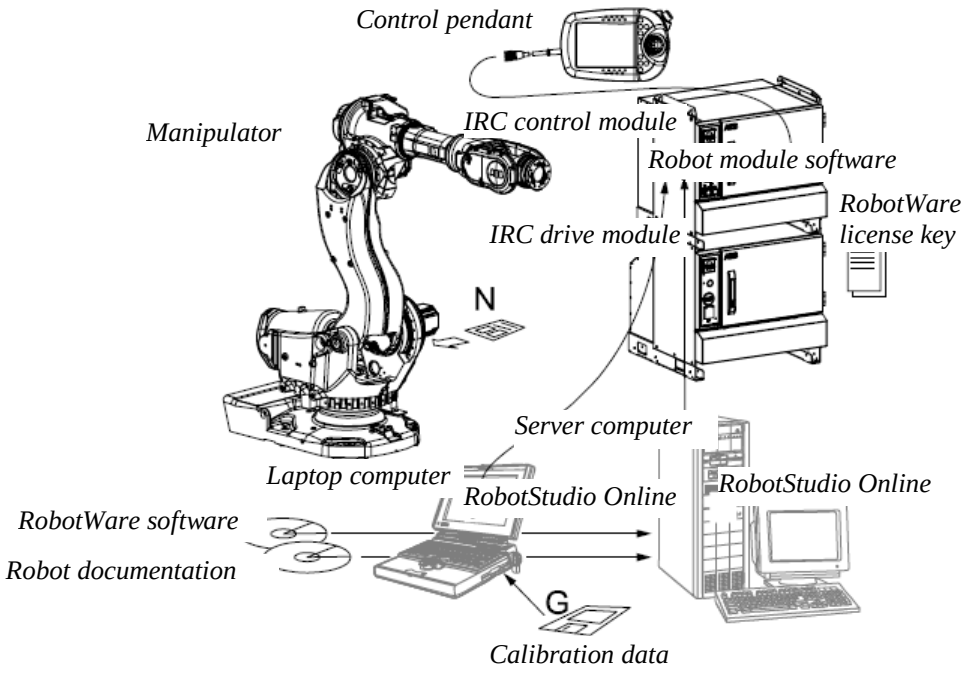

Figure 1.1 Industrial robot IRB 1600 of ABB and its components

Hardware of the control system of the robot IRB 1600 is a multiprocessor system with different types of memory, e.g. reprogrammable *Flash* memory and hard disc memory. Because of higher reliability for the power supply of this system, an uninterruptible power supply (UPS) device is used.

Software used for robot control has an object oriented structure. For ABB robots the *RobotWare* software products and high level programming language *RAPID* is used.

The control device of the ABB IRB robot, IRC5 is located in a separate cabinet, that can be completed with one or more drive modules or a technological control module. For example, Fig. 1.2 shows the robot's system consisting of three manipulators, controlled by three drive modules and a central control module. Fig. 1.3 shows the robot control cabinet with a spot welding control section.

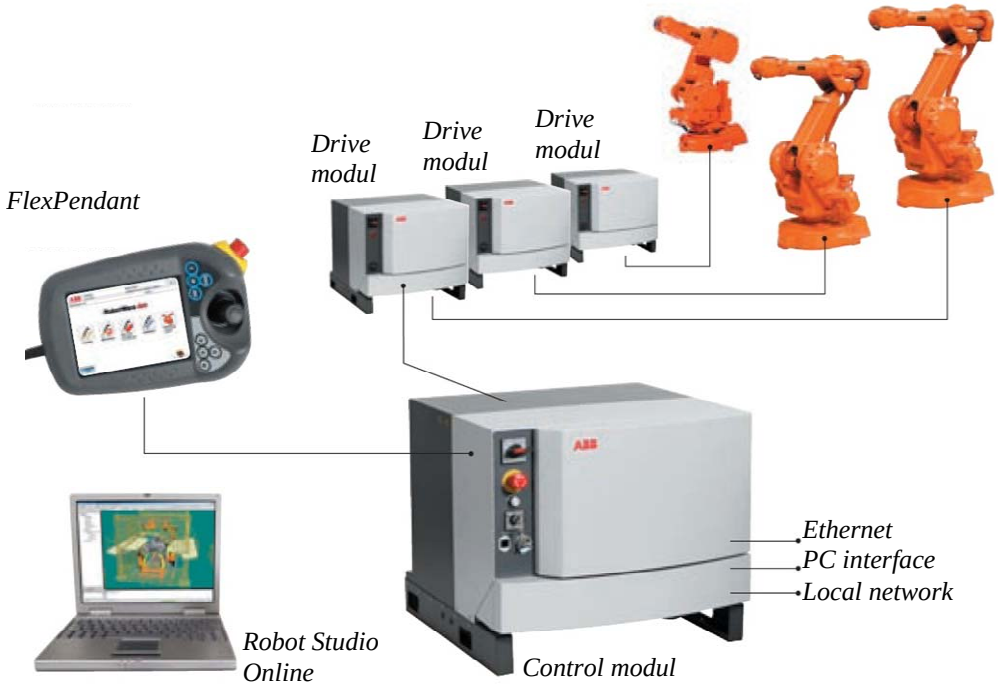

Figure 1.2 Robot's system with three manipulators

Figure 1.3 Robot control cabinet with a spot welding control section

Control Module contains all electronic control devices for the control of a robot system including the main control computer of a robot or a central controller.

Drive Module contains power electronics of electrical drives. In the case of multi-manipulator or so called Multi-Move system, several drive modules will be used.

RobotWare software is stored on a CD-disk memory and consists of software components needed for robot adjustment, control and maintenance.

Robot documentation is the electronically stored technical information on a CD-disk, consisting of installation, application and safety information for robot users.

Robot system software is stored in the controller memory and will be used for operation control of a robot system. Software can be loaded to the controller memory from the server computer via the local area network (LAN).

FlexPendant is connected to the robot's controller and used for robot manual programming and control. FlexPendant has a color touch screen, a joystick and only 8 hardware push buttons for robot control.

RobotStudio Online is the multi-functional base software package for a personal computer, used for robot control in combination with hardware control from FlexPendant. This package can be installed in an ordinary personal computer and it complies with Windows 2000 or later versions. Normally it will be installed a first to a notebook computer and then to the network server. It is used for initial configuration of a robot system and for loading of all software components to the central controller. Mainly, RobotStudio Online is used for text based programming and control of a robot system. The software helps to compose complex programs with a high number of complex logical structures.

Calibrating data are stored to disk memory and used for the adjustment of absolute positions of the manipulator and the full robot system in the case of special technological options (*absolute accuracy option only*).

Network server is used for storage and maintenance of software (e.g. *RobotWare*) and documentation files of a robot system. A server computer duplicates the functions of a notebook computer and in some cases one computer can be used for both. A robot system can operate without a server computer if there is no data exchange between the server and the controller. Normally a server computer is connected via Ethernet network to one or more controller adapters. The software of a robot system can be stored and used for programming with the help of a notebook computer as well as with a server computer.

RobotWare license key is a special code defined and used by a robot producer. Without this code the use of a robot is impossible.

1.2. The description of robot motion

The main function of a robot control software (e.g. *RobotWare*) is the motion control of a robot. The motion of robot's manipulator joints, the tool or the gripper can be described in different coordinate systems. These coordinate systems are used for the realization of several control functions, including off-line programming, program adjustment, coordination of the motion of several robots or a robot and additional servodrives, jogging motion, copy of programs from one robot to another, etc.

The main coordinate systems used to describe the motion of a robot are shown in Fig. 1.4. In the motion control the control of the gripper or tool motion is the most important. Because different types of grippers and tools have different dimensions, a special point, not depending on the type of the tool and called **tool centre point** (TCP) is selected. This point is the origin point of the tool coordinate system. A similar point can be used to describe the gripper or the **wrist coordinate system**. The mutual connections of a tool, a wrist and other coordinate systems are shown in Fig 1.5.

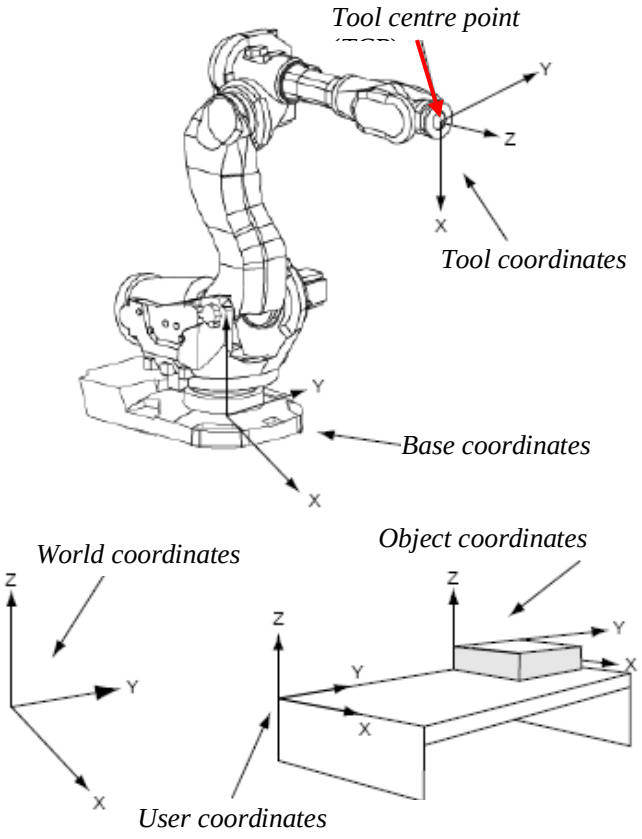

Figure 1.4 Coordinate systems of an industrial robot

The position of the robot and its movements are always related to the **tool centre point (TCP)**. This point is normally defined as being somewhere on the tool, e.g. on top of the welding electrode or at the centre of a gripper. When a position is recorded, it is the position of the TCP that is recorded. This is also the point that moves along a given path at a given velocity.

If the robot holding a work object and is working on a stationary tool, a stationary TCP is used. If that tool is active, the programmed path and speed are related to the work object.

Tool coordinate system The orientation of a tool at a programmed position is given by the orientation of the tool coordinate system. The tool coordinate system refers to the wrist coordinate system, defined at the mounting flange on the wrist of the robot. The tool mounted on the mounting flange of the robot often requires its own coordinate system to enable the definition of its TCP, which is the origin of the tool coordinate system (Fig. 1.5, *a*). The tool coordinate system can also be used to get appropriate motion directions when jogging the robot. If a tool is damaged or replaced, the tool coordinate system must be redefined.

Wrist coordinate system In a simple application, the wrist coordinate system can be used to define the orientation of the tool; here the z-axis is coincident with axis 6 of the robot (Fig. 1.5, *b*). The wrist coordinate system cannot be changed and is always the same as the mounting flange of the robot in the following respects: *The origin* is situated at the centre of the mounting flange (on the mounting surface). *The x-axis* points in the opposite direction, towards the control hole of the mounting flange. *The z-axis* points outwards, at right angles to the mounting flange.

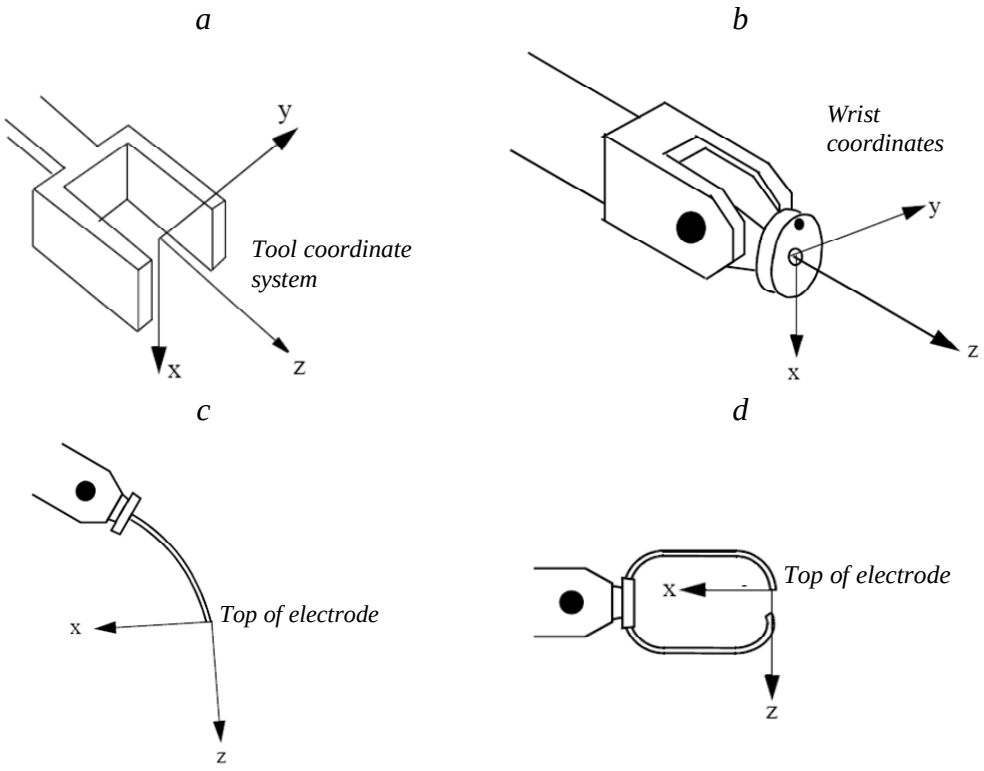

Figure 1.5 Tool and wrist coordinates *a*, *b* and use of these coordinates with arc and spot welding electrodes *c*, *d*

Base coordinate system is linked to the mounting base and stationary base of a robot. In a simple application, programming can be done in the base coordinate system; here the z-axis is coincident with axis 1 of the robot (Fig. 1.6). In this case:
- *The origin* is situated at the intersection of axis 1 and the base mounting surface.
- *The xy plane* is the same as the base mounting surface.

- *The x-axis* points forwards.
- *The y-axis* points to the left (from the perspective of the robot).
- *The z-axis* points upwards.

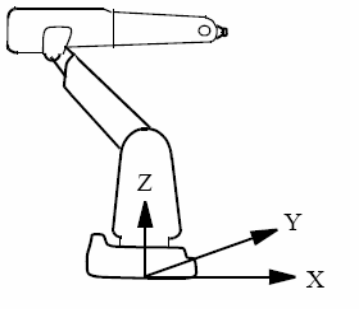

Figure 1.6 Base coordinate system of a manipulator

For example, **floor-mounted robot** can be easily programmed in the base coordinate system. If, however, the robot is mounted upside down (suspended), programming in the base coordinate system is more difficult because the directions of the axes are not the same as the principal directions in the working space. In such cases, it is useful to define a world coordinate system.

World coordinate system will be coincident with the base coordinate system if it is not specifically defined. If several robots work within the same working space at a plant, a common world coordinate system is used to enable the robot programs to communicate with one another. It can also be advantageous to use this type of a system when the positions are to be related to a fixed point in the workshop (Fig. 1.7).

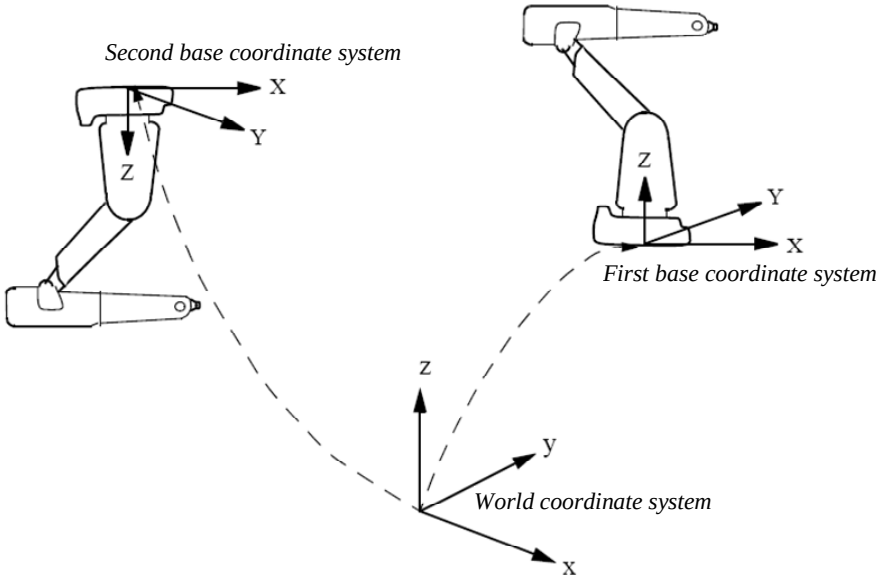

Figure 1.7 Two robots (one of which is suspended) with a common world coordinate system

The user coordinate system is related to the essential points of the technological process. A robot can work with different fixtures or working surfaces having different positions and orientations. A user coordinate system can be defined for each fixture. If all positions are

stored in object coordinates, you will not need to reprogram if a fixture must be moved or turned. By moving (translating or turning) the user coordinate system as much as the fixture has been translated or turned, all programmed positions will follow the fixture and no reprogramming will be required. The user coordinate system is defined based on the world coordinate system (Fig. 1.8).

Object coordinate system is a coordinate system targeted to an object. Normally the user coordinate system is used to get different coordinate systems for different fixtures or working surfaces. A fixture, however, may include several work objects that are to be processed or handled by the robot. Thus, it often helps to define a coordinate system for each object in order to make it easier to adjust the program if the object is moved or if a new object, the same as the previous one, is to be programmed at a different location. This coordinate system is also very well suited for off-line programming since the positions specified can usually be taken directly from a drawing of the work object. The object coordinate system can also be used when jogging the robot. The object coordinate system is defined based on the user coordinate system (Fig. 1.9).

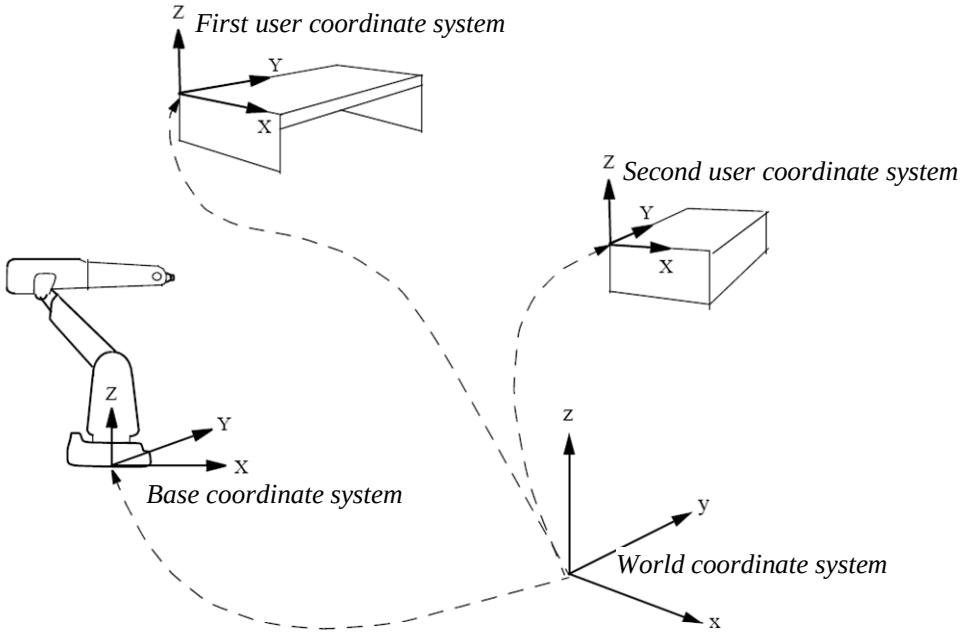

Figure 1.8 User coordinate systems that describe the position of two different fixtures

The object coordinate system can be programmed by the numerical definition of characteristic points or by a manual jogging manipulator via characteristic points and automatic storage of these coordinate values in memory.

The programmed positions are always defined relative to an object coordinate system. If a fixture is moved or turned, it can be compensated for by moving or turning the user coordinate system. Neither the programmed positions nor the defined object coordinate systems need to be changed. If the work object is moved or turned, it can be compensated for by moving or turning the object coordinate system.

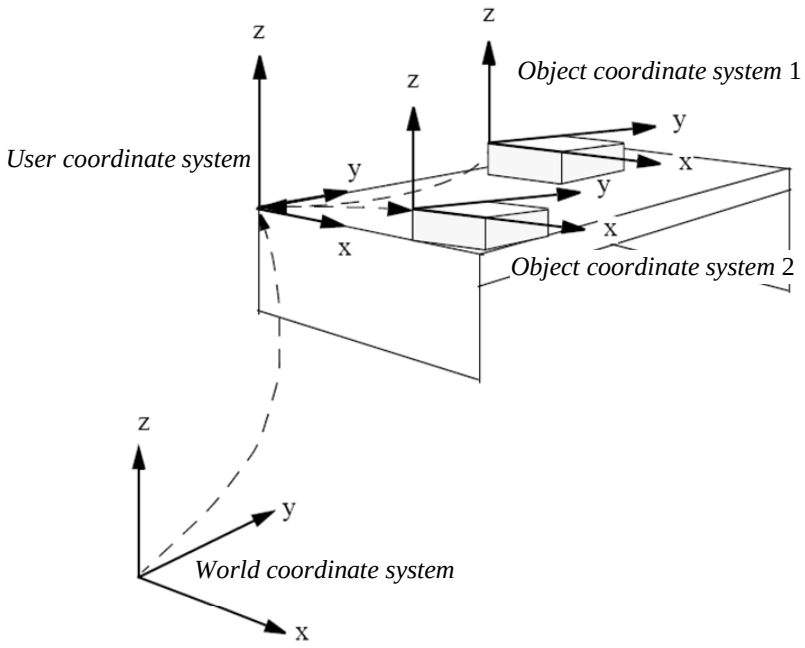

Figure 1.9 Object coordinate systems that describe the position of different work objects located in the same fixture

If the user coordinate system is movable, i.e. coordinated external axes are used, then the object coordinate system moves with the user coordinate system. This makes it possible to move the robot in relation to the object even when the workbench is being manipulated.

Sometimes, the same path is to be performed at several places on the same object. To avoid having to re-program all positions each time, a **displacement coordinate system** can be defined. This coordinate system can also be used in conjunction with searches to compensate for differences in the positions of the individual parts. The displacement coordinate system is defined based on the object coordinate system (Fig. 10).

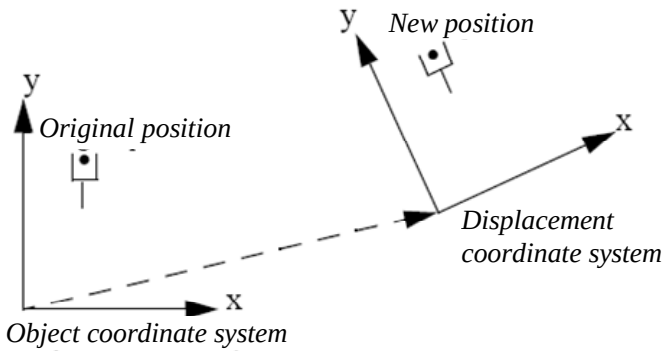

Figure 1.10 Translated and rotated displacement coordinate system

Robot's movements must be initially planned and programmed. Movement planning is based on the selection of coordinate systems and the description of movements in the selected

coordinates. After that the movement will be realized with the help of installed robot software.

During program execution, positioning instructions in the robot program control all movements. The main task of positioning instructions is to provide the following information on how to perform movements:

- the destination point of the movement (defined as the position of the tool centre point, tool orientation, robot configuration, and the position of the external axes);
- the interpolation method used to reach the destination point, e.g. joint interpolation, linear interpolation or circle interpolation;
- the velocity of the robot and external axes;
- the zone data (defines how the robot and the external axes are to pass the destination point);
- the coordinate systems (tool, user and object) used for the movement

As an alternative to defining the velocity of the robot and the external axes, the time for the movement can be programmed.

The location of TCP is calculated by robot software on the basis of measurement data received from the position sensors of the drives of manipulator joints. The measured data characterize the joint position in the **joint coordinate system**. By the calculations and solving the direct task of manipulator kinematics, the position of TCP in the base coordinate system is found. Then the real position of TCP is compared with the programmed reference position. The planned trajectory will be realized by solving the inverse task of manipulator kinematics after the calculation of reference positions for all joints of the manipulator in joint coordinates. The reference positions of joints will be realized with the help of servo drives. The methods for solving the tasks of manipulator direct and inverse kinematics and formulas for calculations are given in Chapter 2 of this book. The methods for trajectory and path planning of a robot are given in Chapter 3.

Programmed motion of a robot

A robot has the following types of programmed movements:

- **Joint motion** (joint interpolation) is the independent movements of joints to the destination position. Reaching the position happens at the same time moment.
- **Linear motion** (linear interpolation). The TCP is moving along a straight line.
- **Circle motion** (circular interpolation). The tool centre point is moving along a circle.

Joint interpolation

When the accuracy of the path is not too important, this type of motion is used to move the tool quickly from one position to another. Joint interpolation also allows an axis to move from any location to another within its working space in a single movement. All axes move from the start point to the destination point at constant axis velocity. The velocity of the tool centre point is expressed in mm/s (in the object coordinate system). As interpolation takes place axis-by-axis, the velocity will not be exactly the programmed value.

During interpolation, the velocity of the limiting axis, i.e. the axis that travels fastest relative to its maximum velocity in order to carry out the movement, is determined. Then the velocities of the remaining axes are calculated so that all axes reach the destination point at

the same time. All axes are coordinated in order to obtain a path that is independent of the velocity. Acceleration is automatically optimized to the max performance of the robot.

Linear interpolation
During linear interpolation, the TCP travels along a straight line between the start and destination points. To obtain a linear path in the object coordinate system, the robot axes must follow a non-linear path in the axis space. The more non-linear the configuration of the robot is, the more accelerations and decelerations are required to make the tool move in a straight line and to obtain the desired tool orientation. If the configuration is extremely non-linear (e.g. in the proximity of wrist and arm singularities), one or more of the axes will require more torque than the motors can give. In this case, the velocity of all axes will automatically be reduced.

The orientation of the tool remains constant during the entire movement unless a reorientation has been programmed. If the tool is re-orientated, it is rotated at constant velocity. A maximum rotational velocity (in degrees per second) can be specified when rotating the tool. If this is set to a low value, reorientation will be smooth, irrespective of the velocity defined for the tool centre point. If it is a high value, the reorientation velocity is only limited by the maximum motor speeds. As long as no motor exceeds the limit for the torque, the defined velocity will be maintained. If, on the other hand, one of the motors exceeds the current limit, the velocity of the entire movement (with respect to both the position and the orientation) will be reduced. All axes are coordinated in order to obtain a path that is independent of the velocity. Acceleration is optimized automatically.

Circular interpolation
The trajectory circular interpolation is used for tool motion as well as for interpolation near the intermediate via points of nonlinear trajectory. A circular path is defined using three programmed positions that define a circle segment. The first point to be programmed is the start of the circle segment. The next point is a support point (circle point) used to define the curvature of the circle, and the third point denotes the end of the circle. The three programmed points should be dispersed at regular intervals along the arc of the circle to make this as accurate as possible. The orientation defined for the support point is used to select between the short and the long twist for the orientation from the start to the destination point. If the programmed orientation is the same relative to the circle at the start and the destination points, and the orientation at the support is close to the same orientation relative to the circle, the orientation of the tool will remain constant relative to the path.

Soft servo
In some applications there is a need for a servo, which acts like a mechanical spring. This means that the force from the robot on the work object is a function of the distance between the programmed position and the contact position (e.g., distance between the robot tool and work object).

The relationship between the position deviation and the force is defined by a parameter called ***softness.*** The higher the softness parameter, the larger the position deviation required to obtain the same force.

The softness parameter is set in the program and it is possible to give softness values for all joint drives, including additional servo drives of robot system. The use of the softness parameter gives the flexibility to the robot hand and excludes the break of the tool or workpiece in the case of collision. With high softness values there is a risk that the servo

position deviations may be so large that the axes will move outside the working range of the robot.

Jogging is a low speed manually controlled motion. The jogging motion of a robot can be manually controlled in the following ways:
- motion axis-by-axis, i.e. one axis at a time
- linear motion. The tool centre point is moving linearly in the coordinate system
- modified motion around the tool centre

In the case of **incremental jogging** the step size can be selected. The incremental jogging movement allows an exact positioning of the TCP because during manual control by the joystick of the robot the manipulator moves very slowly with very short steps.

During manual control with **FlexPendant,** location information can be displayed for the manipulator as well as for additional servos of the robot system.

Singularities
Some positions in the robot working space can be attained using an infinite number of robot configurations to position and orient the tool. These positions, known as singular points (singularities), constitute a problem when calculating the robot arm angles based on the position and orientation of the tool.

Generally, a robot has two types of singularities: arm singularities and wrist singularities. Arm singularities are all configurations where the wrist centre (the intersection of axes 4, 5, and 6) ends up directly above axis 1 (Fig. 1.11). Wrist singularities are configurations where axes 4 and 6 are on the same line, i.e. axis 5 has an angle equal to 0.

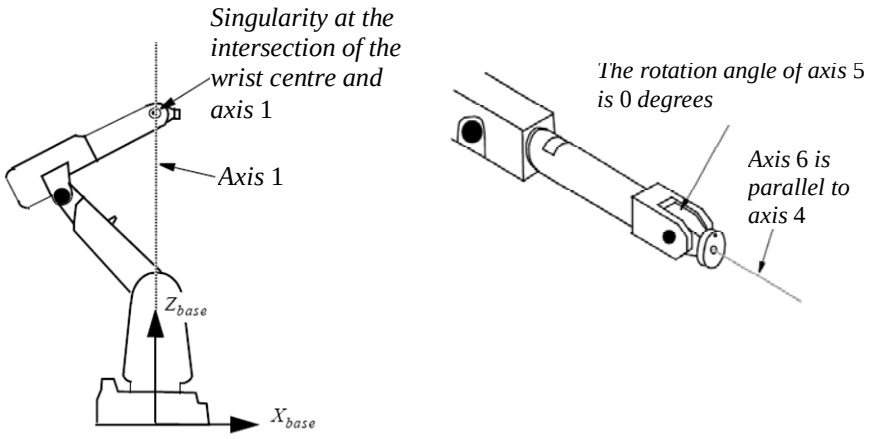

Figure 1.11 Arm and wrist singularities

Singularity handling
When executing a linear or circular path close to a singularity, the velocities in some joints may be very high. In order not to exceed the maximum joint velocities, the linear path velocity is reduced.

Motion supervision is the name of functions for high sensitivity, model based supervision of the robot's movements. Motion supervision includes the detection of collision, jamming, and incorrect load definition. This functionality is called collision detection.

The **collision detection** may trig if the data for the loads mounted on the robot are not correct. This includes load data for tools, payloads and arm loads. If the tool data or payload data are not known, the load identification functionality can be used to define it. Arm load data cannot be identified.

When the collision detection is triggered, the motor torque is reversed and the mechanical brakes are applied in order to stop the robot. The robot then backs up a short distance along the path in order to remove any residual forces which may be present if a collision or jam occurred. After this, the robot stops again and remains in the motors on state. A typical collision is illustrated in Fig. 1.12. Motion supervision is active only when at least one axis (including external axes) is in motion. When all axes are standing still, the function is deactivated. This is to avoid unnecessary trigging due to external process forces.

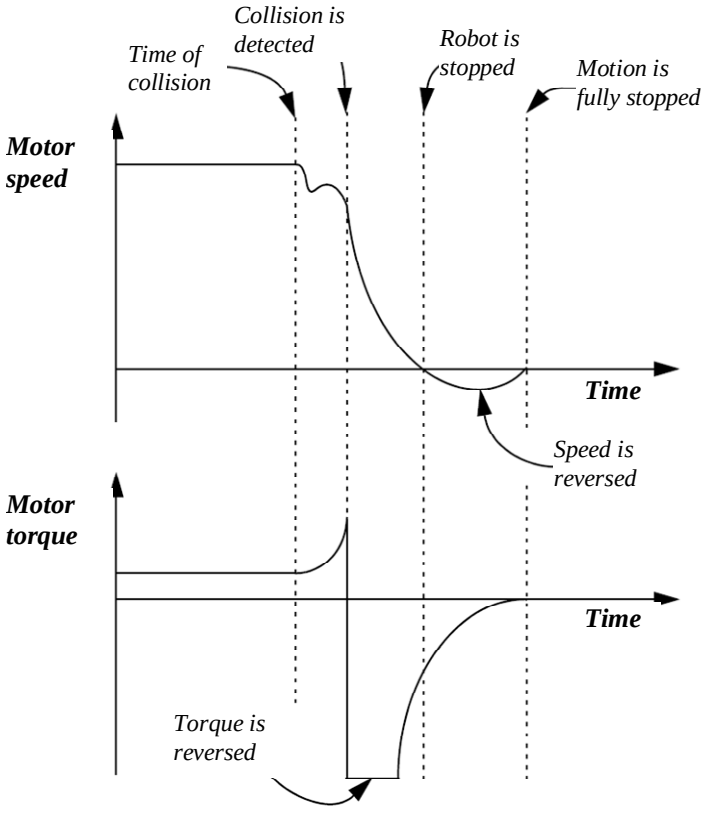

Figure 1.12 Motor speed and torque diagrams during collision detection

The distance the robot backs up after a collision is proportional to the speed of the motion before the collision. If repeated, low speed collisions occur, the robot may not back up sufficiently to relieve the stress of the collision. As a result, it may not be possible to jog the robot without the supervision triggering. In this case the jog menu is used to turn off the collision detection temporarily and the robot is jogged away from the obstacle.

Operation with high inertia
Using the dynamical modeling and the **dynamic model** of a manipulator drive helps us to work if the load has high inertia. The drive parameter is to be adjusted accordingly to the load inertia. This function helps us also in the case of transportations of large resilient objects. The adjustment of drive's dynamical parameters allows one to avoid mechanical oscillations during the transportation of resilient objects. The dynamic modeling is based on automatic load detection.

A robot has the function of **automatic load identification** that allows adjusting parameters of drive's dynamic model according to the load torques and forces. As the result of using this function, the robot system has higher reliability and life time. During the installation of the robot there is no need to have detailed measurements or calculations of the load. For example, the function of automatic load detection is used to control different ABB robots (IRB 140, 1400, 1600, 2400, 4400, 6400RF, etc).

Motion supervision function is activated only in the case of a moving robot. Otherwise, the function is deactivated.

Additional degrees of freedom (DOF) and additional motors
Additional servos giving additional degrees of freedom for robot operation can be added to the robot system. The robot's control system can control additional servo drives of added mechanisms. The motion of a robot and additional mechanisms will be coordinated by common programming.

1.3. Development environment for robotics - virtual robotics

High speed of operation and large memory capacities of modern computers allow generating and using of virtual environments for industry automation. These virtual environments include virtual programming and operation of machine tools and robots. The operation of virtual robots in a virtual environment is based on the same software and on the same programs as the operation of real robots in a real environment. The programming of different production operations, for example, arc welding, spot welding, assembly, cleaning, spraying, cutting, deburring, gravity die casting, grinding, polishing, machine tending, handling, painting, packing or palletizing will be made in a virtual environment for virtual robots. After the transfer of programs from a virtual to a real environment the real robots will start to work.

That kind of virtual robotics environment allows the development of new robot systems, selecting and programming robots long before the real robot system will be developed, transported and installed. The composition of programs and their debugging in a virtual environment cuts sufficiently the time for preparing the robot system and guarantees higher reliability of the system.

An example of the virtual robotics development environment is the ***COSIMIR*** (*cell oriented simulation of industrial robots*) used for robot simulation in flexible manufacturing system (FMS) (http://www.cosimir.com/). Development environment *COSIMIR* is used for the simulation of Festo robots, but it is also used by other companies. Similar to the *COSIMIR* simulation environment are also ***Camelot ROPSIM***, developed in Denmark, and ***Robot Studio Online***, developed by ABB Company (Fig. 1.13). The upper part of the figure shows the Festo *COSIMIR* environment and the lower part the ABB *RobotStudio* environment.

Development software *COSIMIR, Camelot ROPSIM* and *RobotStudio* are be considered to the new field of robotics - **virtual robotics**. In the virtual robotics environment different complex production systems with several robots can be developed and programmed. For example, with the help of ABB *RobotStudio MultiMove* system motion planning, programming and operating of multi-robot system from one controller are possible.

In the case of automation of an arc welding operation using robot, the geometry of welding trajectory is defined first. In the second phase, technological parameters, such as the slope of the welding electrode, parameters of welding junction, angles of turning, conditions of starting and finishing are determined.

Modern robotics software includes standard technological packages for production tasks. For example, ABB has a software package *ArcWeld PowerPac* that automatically generates welding programs based on geometrical dimensions of trajectory and welding objects (technological documentation), defines right directions for the welding electrode during the welding process as well as during starting and finishing of this. It is taking into consideration all the limitations of manipulator motion (including singularities) and will inform the user about possible problems during the technological process.

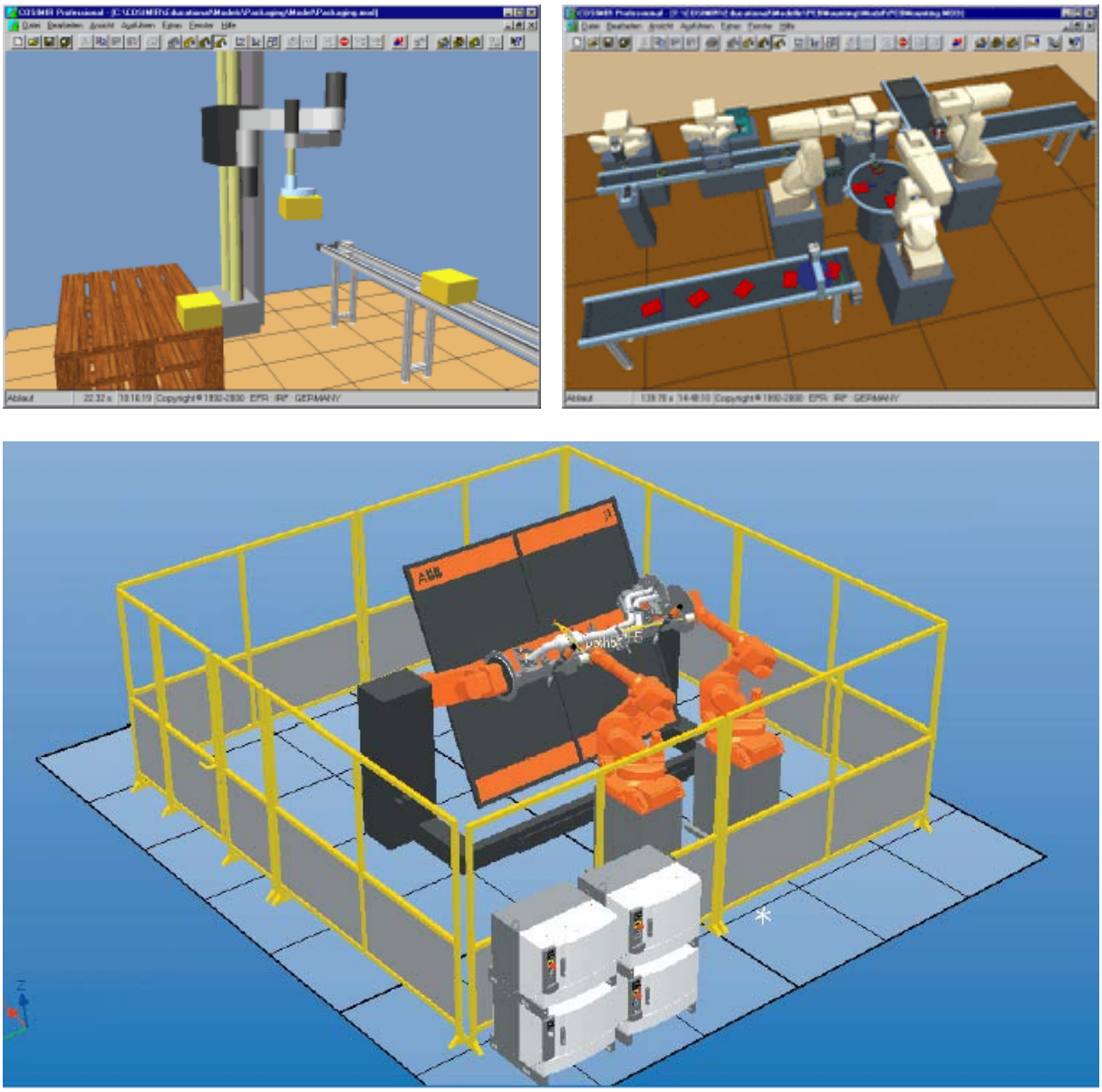

Figure 1.13 Virtual robotics development environments for composing and programming of robot systems: *Festo Cosimir* shown above and ABB *RobotStudio* shown below

Next, a short overview of the components and functions of the ABB development software package *RobotStudio Online* is given.

The functions of *RobotStudio Online* can be used with the help of a special progamme (window) called *robot view explorer*. This window helps us to choose robot type if more than one robot is used and the other components of the robot system. The windows and menus of *RobotStudio Online* environment are shown in Fig. 1.14.

The main components of *RobotStudio Online* are:

- *system builder* for system composing, installation and maintenance,
- *configuration editor* for parameter editing of the installed system,
- *program editor* for *on-line programming* of the system,

- *event log* for *monitoring* of the robot system and for logging of *robot events*,
- *system restoring* and *backing up* software tools,
- *administration* and *user authorization* software tools,
- *file manager* for file change and between PC and system controllers,
- *task window* for task operation in the system controller
- others tools for monitoring and control of the robot system

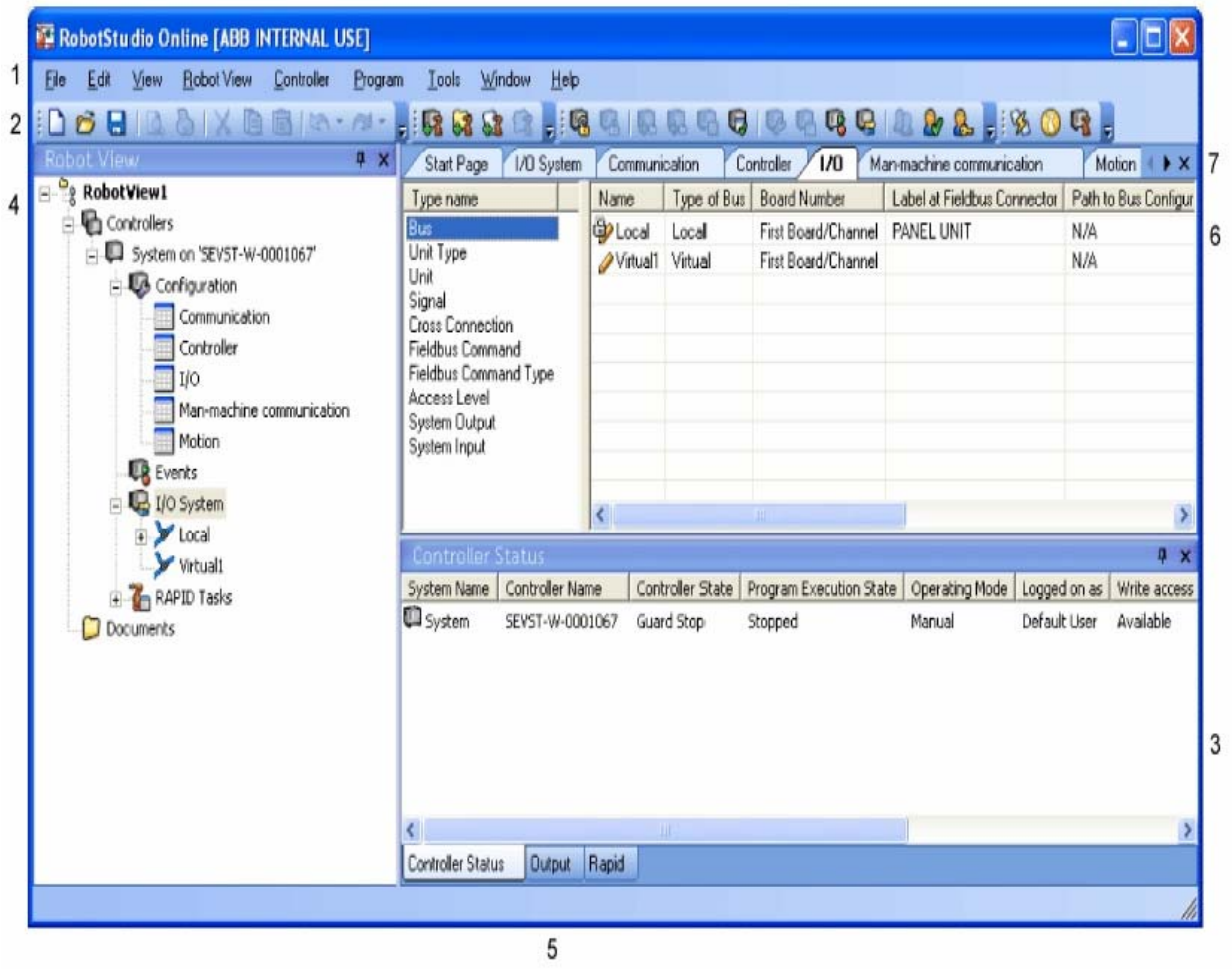

Figure 1.14 Menus and windows of *RobotStudio Online*

Figure 1.14 shows the following objects and windows of *RobotStudio Online*:

1. The ***menu bar*** contains the following menus: file, edit, view, robot view, controller, program, tools, window, and help.
2. The ***toolbars*** contain buttons for quick access to commands in the menus.
3. The ***controller status window*** shows status and access information for each controller in the open robot.
4. The ***robot view explorer*** shows the controllers that are included in the current robot view. From here you access the controller and select which part of it to work with.
5. The ***Output window*** displays messages and responses on actions from the controllers you work with, but this is not the same as the controller's events or log file. The Output window also has a specific tab for RAPID output, which displays programming messages from the controller when working with robot programs.
6. The ***workspace*** displays the active windows, for instance, the start page, a program editor window, or an I/O window as in the picture above.
7. The ***windows tabs*** let you select which window to view in the workspace if you have several windows open at the same time. These tabs are only available when the view is set to the workbook mode.

1.4. Programming of robots and robot programming languages

All robot programming methods can be divided into two groups: manual and automatic programming methods. The manual programming of robots (Fig. 1.15) can be divided into text-based programming and graphical programming methods. Text based programming is the programming in robot programming languages that can be divided in relation to language structure and commands. Robot program is based on the sequence of robot commands. The program algorithm includes different subroutines, cycles and branches of command sequences. In terms of the character of robot commands and program structure, robot programming languages are divided into *device-based*, *procedure-based* and *behaviour-based* languages. To create better images of robot technological operation and more convenient programming, during last years, more attention has been paid to graphical programming methods. In this case the program is given with algorithm flow diagrams, graph schemes, or functional schemes. In many cases, the graphical programming, comparison with text based programming, needs shorter time for learning and program generation.

The automatic programming of robots is based on learning systems, demonstration systems or instructive systems. As the result of automatic programming robot generates work program for robot action. In the case of industrial robots, the programming by demonstration, using teach pendant, is more common. The demonstrated by operator reference trajectory for the manipulator, will be automatically stored to robot's memory and will be later exactly executed by the programmed control of the robot.

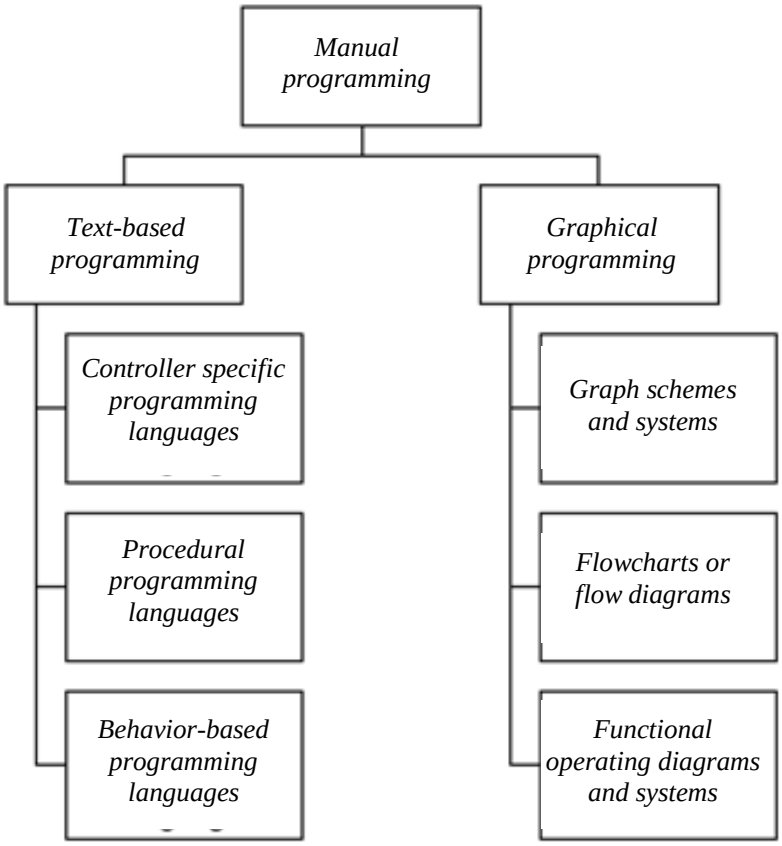

Figure 1.15 Manual programming of robots

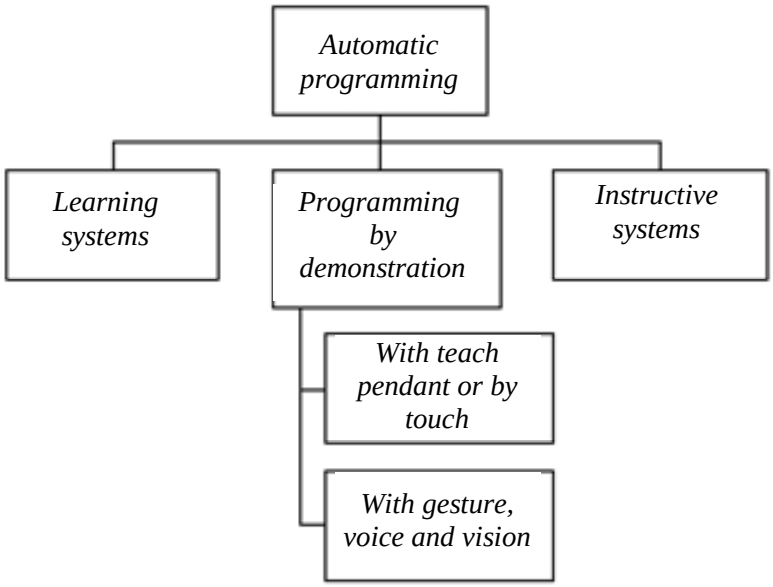

Figure 1.16 Automatic programming of robots

Robot's programming languages

The robot system development environment *COSIMIR® Professional* supports programming in different robot languages. Some of the languages are specialized for a defined type of a robot, but others are universal languages and allow programming of several robot types. Examples of robot programming languages are: *Industrial Robot Language* (shortly *IRL*), VAL developed by Unimation Company, *Kuka Robot Language* (*KRL*) developed by Kuka Company, *RAPID* developed by ABB, *Movemaster Command* (*MRL*) and MELFA *Basic*, *Bosch Automatisierungsprogrammiersprache* (*BAPS*), developed by Bosch Company, and robot languages *Simple Robot Programming Language* (*SRPL*), V+ etc.

In the Laboratory of Industry Automation of the Department of Electrical Drives and Power Electronics at Tallinn University of Technology, the programming languages *Movemaster Command* (*MRL*) and *MELFA Basic* for programming of Mitsubishi robots are used. For programming of an ABB robot, in the Laboratory of Robotics, the robot language *RAPID* is used.

Work planning
Work operations are planned in the first stage of robot programming

- general planning of the production task and the determination of robot's role
- description of the production process with flow diagrams to specify the concrete working tasks for the robot and other machines in the robot system, including:
  - decomposition of the working process and definition of detailed working operations
  - definition of positioning locations and numbers for these
  - definition of input and output signals
  - composition of algorithm diagrams, subroutines, routines and program modules for a robot

For robot programming the following software objects are defined:

Program module
Each program module contains data and routines for a certain purpose. The program is divided into modules mainly to enhance overview and facilitate handling of the program. Each module typically represents one particular robot action or a similar one. All program modules will be removed when deleting a program from the controller program memory.
Program modules are usually written by the user.

Data
Data are values and definitions set in the program or system modules. The data are refined to by the instructions in the same module or in a number of modules (availability depending on data type).

Routine
A routine contains sets of instructions, i.e. defines what the robot system actually does. A routine may also contain data required for the instructions.

Entry routine
A special type of routine, in English sometimes referred to as "main", is defined as the program execution starting point. Each program must have an entry routine called "main", or it will not be executable. The default name for the main can be changed by the system parameter configurations type Task.

Instruction
Each instruction is a request for a certain event to take place, e.g. "Run the manipulator TCP to a certain position" or "Set a specific digital output". The instructions, their syntax and function are thoroughly described in the Rapid instructions of the Technical Reference manual.

System module
Each system module contains data and routines to perform a certain function. The program is divided into modules mainly to enhance an overview and facilitate handling the program. Each module typically represents one particular robot action or a similar one. All system modules will be retained when "Delete program" is ordered. System modules are usually written by the robot manufacturer or line builder.

Each robot is programmed in a convenient robot language. Different programming languages of industrial robots have relatively similar structure and similar systems of commands. Because of this, the acquired programming skills can be used for future programming in other languages. The following programming examples for simple operations of a robot are written in the robot programming language *Movemaster Command*. Normally the instructions (commands) for robot programming are divided between the following groups:

- **Instructions for motion control and positioning (location)**
  contain information about positioning location, co-ordinates, trajectory interpolation, tool motion speed and time intervals (timers).
- **Instruction for program control**
  contain information about program start and stop, branching and cycle repeating, event counting and program interruption conditions.
- **Instructions for gripper or tool control**
  contain information about the gripper closing and opening conditions, sometimes about force control of tool or gripper fingers.
- **Input and output instructions**
  contain information for input and output signal control and for cooperation with other machines and devices in the robot system.
- **Communication instructions**
  for data communication and transfer of robot’s inner information to the computer.
- **Other instructions**
  parameter settings, program selection, resetting of fault information, etc.

The following **programming examples** are based on describing of a simple use of a robot for *pick and place* type actions. Figure 1.17 shows the use of a robot for picking an object in position 1, the object transfer and placing it in destination position 2. In this case the following positions are described:

position 1 – position for taking (picking) the object
position 2 – position for putting (placing) the object
position 10 – initial position for taking (picking) the object
position 20 – initial position for putting (placing) the object

Positions 1 and 2 will be programmed and stored by the help of the teaching pendant and positions 10 and 20 will be numerically defined during robot programming.

Task execution is beginning by robot gripper motion 1 to the start point (position 10) above the object that will be transferred. The following step is the motion 2 and gripping the object in position 1. Then the object will be lifted (motion 3) up back to position 10 and transferred (motion 4) to position 20. This is the start point for putting (placing) the object in destination point (position 2). After object placing, the gripper opens and moves back to position 20 (motion 6). Robot's motion 7 moves the gripper back to start point location (position 10) after that the robot is ready for the next cycle of object transfer.

The flow diagram of the algorithm for a pick and place type operation of the robot is shown in Fig. 1.18. The flow diagram helps us to compose a robot's operating program in the defined programming language and use the defined program instructions (Table 1.1).

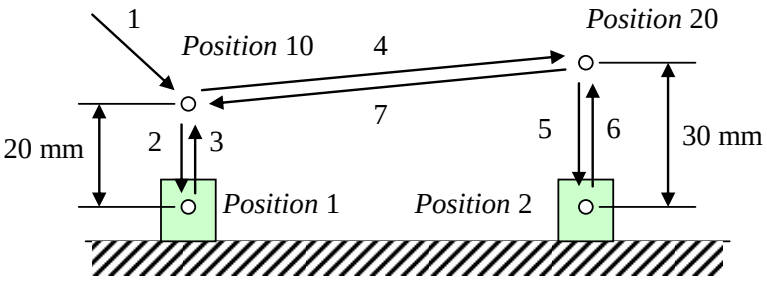

Figure 1.17 Pick and place type transfer operation

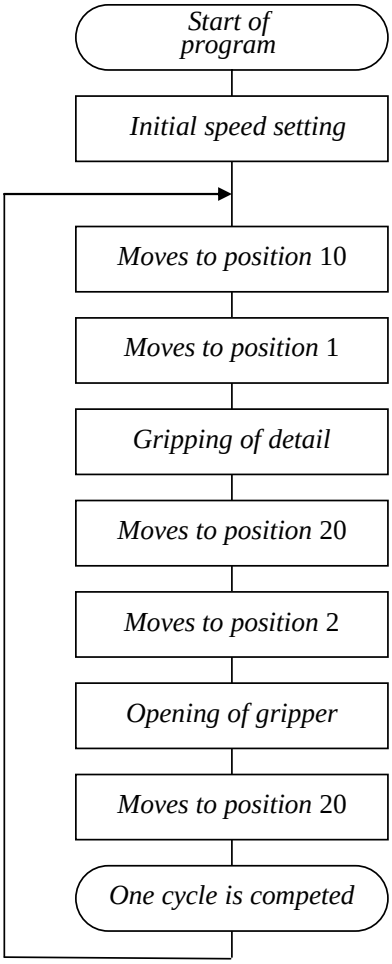

Figure 1.18 Flow diagram for pick and place type transfer operation

Table 1.1

Example of a robot program written in the language *Movemaster Command* for picking and placing operations in positioning locations 1 and 2

Row number	Program instruction	Description
10	PD 10,0,0,20,0,0,0	Defines the aerial distance (*position definition*) of travel from position 1 (Z = 20 mm)
20	PD 20,0,0,30,0,0,0	Defines the aerial distance (*position definition*) of travel from position 2 (Z = 30 mm)
30	SP 18	Sets the initial speed (*speed*)
40	MA 1,10,O	Opens hand and moves 20 mm above the position 1
50	MO 1,O	Moves (*move*) with open hand to position 1
60	GC	Closes hand (*gripper close*) and grasps the workpiece
70	MA 1,10,C	Moves (*move*) 10 mm above position 1 with hand closed
80	MA 2,20,C	Moves (*move*) 30 mm above position 2 with hand closed
90	MO 2,C	Moves (*move*) and puts the workpiece in position 2
100	GO	Opens hand (*gripper open*) and releases the workpiece
110	MA 2,20,O	Moves (*move*) 30 mm above position 2 with hand opened
120	GT 40	Repeats this program (Jumps to line 40) (*go to*)

Use of input signals and program interruptions

For the detection of picked objects on the working plane often special sensors are used. The sensor signals will be transferred to the robot's controller and used during program execution as input information for the control system. The input information is used for logical decision-making and branching in the program. Depending on the signal value, the robot can interrupt the execution of a current program and start some new operations. For example, if the robot sensor detects any object near the gripper (in this case the detection distance is 50 mm along z-axis) during motion, the robot control system will disable other interruptions, gripping the object and continue the program execution. If occasionally the object did not do its gripping position, the robot goes back to position 1 and repeats the gripping process (Fig. 1.19).

Robot's gripper is moving 50 mm along -*z* axis direction (program row 40). If the object (*workpiece*) is situated in its defined place, the limit switch signal interrupts the program sequence and the following instruction that must be executed is on row 60. Robot grips the object and moves back to position 1. If there is no object in the gripping place, the program continues from row 30 (jumps from row 50 to row 30) and the gripper moves back to position 1. Then the operation is repeating cyclically. The flow diagram of the algorithms is shown in Fig. 1.20 and the program text in Table 1.2.

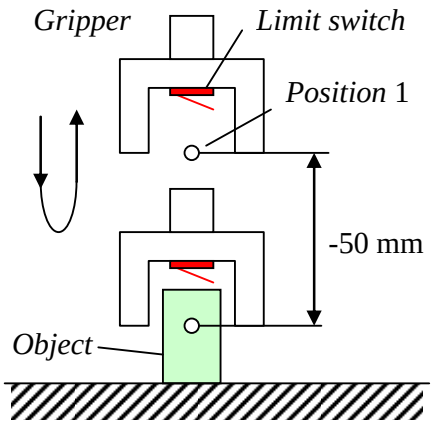

Figure 1.19. Use of input signal and interruption of programming

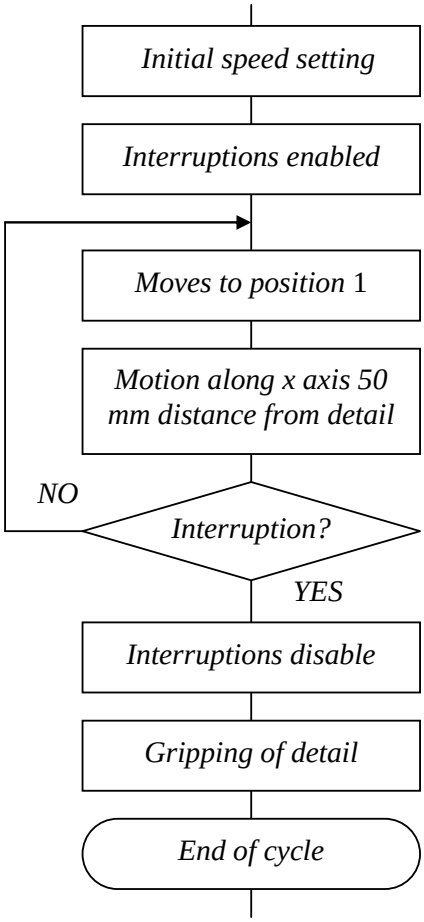

Figure 1.20. Program flow diagram in the case of input signal and interruption

Table 1.2

Example of a robot program written in the language *Movemaster Command*, using an input interrupion signal for the picking operation

Row number	Program instruction	Description
10	SP 20	Sets the initial *speed*
20	EA +900,60	*Enables interrupt* of bit 900
30	MO 1,O	Moves with open fingers above the workpiece (*move, open*) to position 1
40	DS 0,0,-50	Moves 50 mm in the –*z* direction (linear interpolation)
50	GT 30	Jumps (instruction *go to*) to row 30 if no interruption signal
60	DA 900	Disables interrupt of bit 900
70	GC	Closes gripper (*gripper close*) and grasps the workpiece
80	MO 1,C	Moves to position 1 with closed gripper (*move, closed*)

Objects transfer from place to place is more complicated if many objects are palletized and the robot has multiple positioning points that are systematically organized on the picking place as well as on the putting place. Picking of objects from one pallet, transfer to intermediate testing device, forward transfer and placing objects into another pallet are shown in Fig. 1.21.

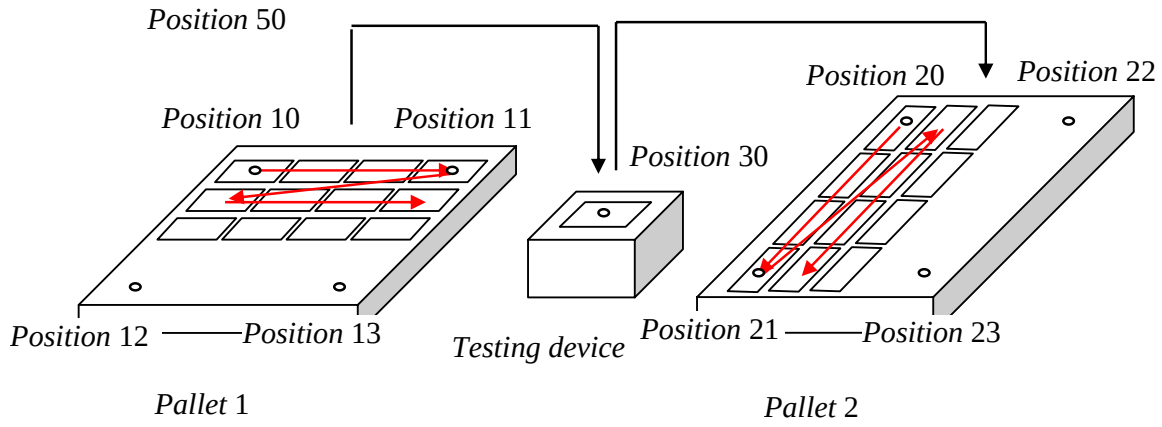

Figure 1.21 Picking of objects from one pallet, transfer to intermediate testing device, forward transfer and placing objects into another pallet

To solve of the picking and placing task in the case of pallets use is more complicated if the pallets have different configurations and orientation, for example, if the first pallets have 10 x 6 cells and the second 15 x 4 cells.

If the pallet has many regularly situated cells in rows and columns, it is possible to use the row and column position counters for palletizing. In the case of two pallets, a total of four counters must be set:

- counter 11 – column counter for pallet 1
- counter 12 – row counter for pallet 1
- counter 21 – column counter for pallet 2
- counter 22 – row counter for pallet 2

For programming of a robot, initially all characteristic positions must be defined and their numerical values must be determined. This operation will be done with the help of the teaching pendant. In the case shown in Fig. 1.21, the following positions described in Table 1.3 are used.

Table 1.3

Description of positioning locations

Number of position	Description of position
Position 1	Pallet 1 setting position
Position 2	Pallet 2 setting position
Position 10	Pallet 1 reference position
Position 11	Pallet 1 column terminating position

Position 12	Pallet 1 row terminating position
Position 13	Pallet 1 corner position opposite to reference
Position 20	Pallet 2 reference position
Position 21	Pallet 2 column terminating position
Position 22	Pallet 2 row terminating position
Position 23	Pallet 2 corner position opposite to reference
Position 30	Test equipment set position
Position 50	Distance of travel from pallets

In this program input signal “testing is finished” is used. Robot grasps and moves the work piece after the input signal “testing is finished” is received.

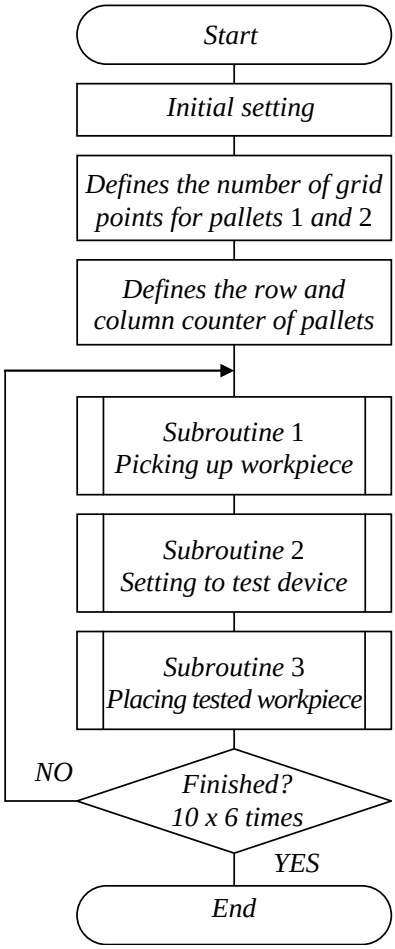

Figure 1.21. Main program

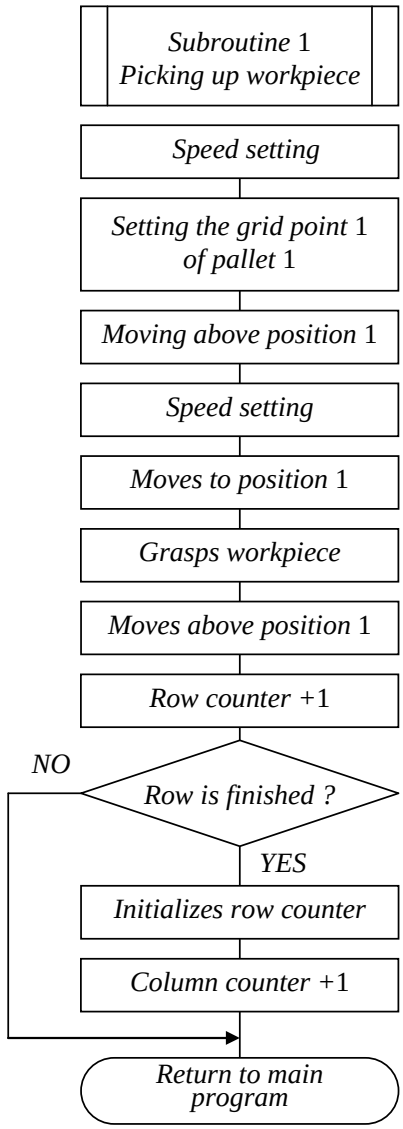

Figure 1.22. Subroutine 1

The program for grasp, transfer, testing and putting object consits of one main program and three subroutines: for grasp from pallet 1, for testing and for putting object to pallet 2. (Fig. 1.22).

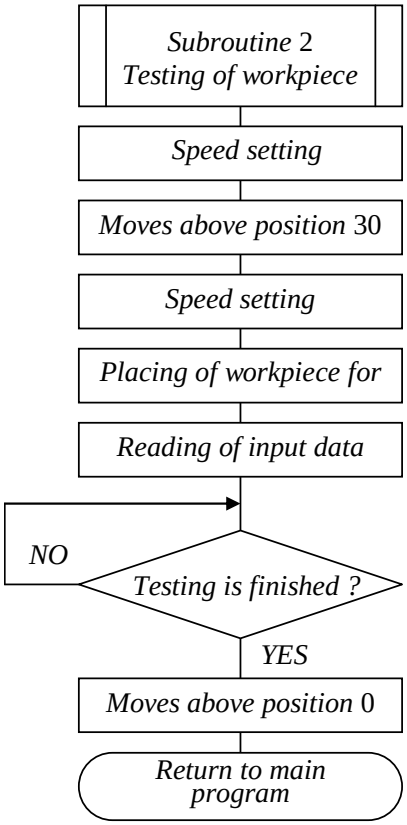

Figure 1.22. Subroutine 2

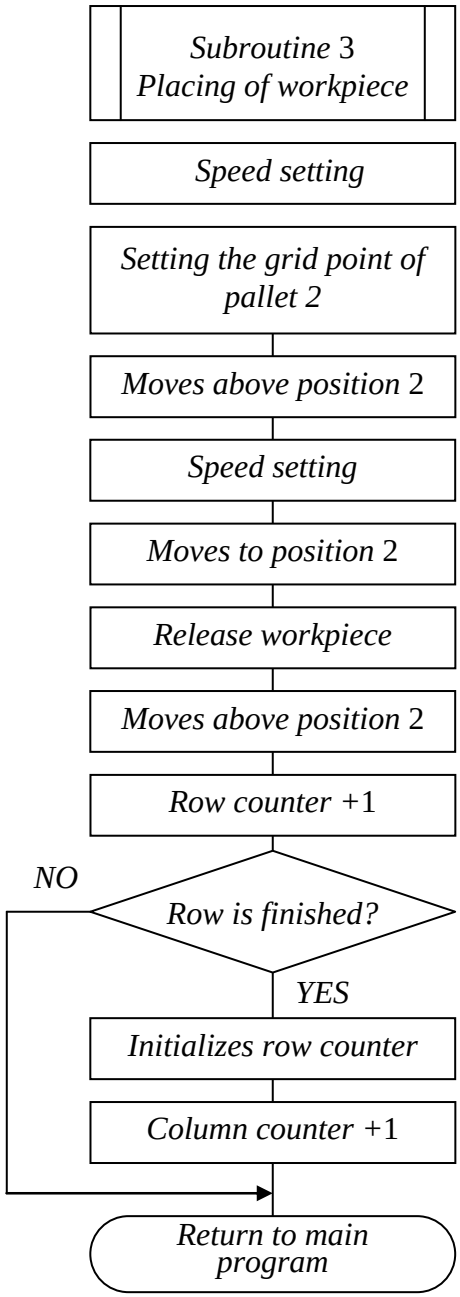

Figure 1.22. Subroutine 3

Table 1.4

Example of a program for picking of objects from one pallet, transfer to intermediate testing device, forward transfer and placing objects into another pallet

Row number	Program instruction	Description
10	PD 50,0,0,20,0,0,0	Defines the distance of travel (z = 20 mm) from position 50
15	TL 145	Sets tool length at 145 mm
20	GP 10,8,10	Sets hand open/closed parameters
25	PA 1,10,6	Defines the number of grid points in the column and row for pallet 1 (10 vertically and 6 horizontally)
30	PA 2,15,4	Defines the number of grid points in the column and row for pallet 2 (15 vertically and 4 horizontally)
35	SC 11,1	Initializes column counter 11of pallet 1 (sets 1)
40	SC 12,1	Initializes row counter 12 of pallet 1 (sets 1)
45	SC 21,1	Initializes column counter 21of pallet 2 (sets 1)
50	SC 22,1	Initializes row counter 22 of pallet 2 (sets 1)
Main program		
100	RC 60	Sets the number of repeat cycles 60 of loop up to line 140
110	GS 200	Calls subroutine1 (program line 200)
120	GS 300	Calls subroutine 2 (program line 300)
130	GS 400	Calls subroutine 3 (program line 400)
140	NX	Returns to line 100
150	ED	End
Subroutine 1		**Picking up the workpieces from pallet 1**
200	SP 25	Sets speed 25
202	PT 1	Sets grid point of pallet 1 to position 1
204	MA 1,50, O	Moves with open gripper to location 50 above position 1
206	SP 8	Sets speed 8
208	MO 1, O	Moves with open gripper to position 1
210	GC	Closes hand and grasps the workpiece
212	MA 1,50, C	Moves with grasped workpiece to location 20 mm above position 1 (position 50)
214	IC 11	Increments column counter 11 of the pallet 1 by +1
216	CP 11	Saves the value of counter 11 in the internal register
218	EQ 11,230	Jumps to line 230 on completing the column line (compares with value 11)
220	RT	Ends the subroutine otherwise, returns to main program
230	SC 11,1	Initializes counter 11 (sets 1)

232	IC 12	Increments row counter 12 of pallet 1 by +1
234	RT	Ends the subroutine and returns to main program
Subroutine 2		**Setting up the workpieces on the test equipment**
300	SP 25	Sets speed 25
302	MT 30,-50, C	Moves to position 30, 50 mm ahead of the test equipment
304	SP 8	Sets speed 8
306	MO 30, C	Moves the workpiece to the inspection equipment position 30
308	ID	Inputs external data
310	TB -7, 308	Waits for the test to complete (repeats from line 308)
312	MT 30,-50, C	Moves to the position 30, 50 mm ahead the inspection equipment
314	RT	Ends the subroutine and returns to the main program
Subroutine 3		**Placing the tested workpiece in pallet 2**
400	SP 25	Sets speed 25
402	PT 2	Sets the grid point of pallet 2 to position 2
404	MA 2,50, C	The robot moves with closed gripper, 20 mm above position 2 (position 50)
406	SP 8	Sets speed 8
408	MO 2, C	Moves with closed gripper to position 2
410	GO	Robot opens the gripper and puts the workpiece
412	MA 2,50, O	Moves with open gripper 20 mm above position 2 (position 50)
414	IC 21	Increments column counter 21 of pallet 2 by +1
416	CP 21	Saves the value of counter 21 in the internal register
418	EQ 16,430	Jumps to line 430 on completing the column line (compares with value 16)
420	RT	Ends the subroutine otherwise, returns to the main program
430	SC 21,1	Initializes column counter 21 (sets 1)
432	IC 22	Increments row counter 22 of the pallet 2 by +1
434	RT	Ends the subroutine and returns to the main program

The program operates in the following way: after testing of a workpiece + 1 will be added to the value of counters. At the end of the column, the program initializes the column counter, adds +1 to the row counter and starts the next row from the beginning of the column. (From program line 214 up to 232 and from program line 414 up to line 432.) After each taking of a workpiece from pallet 1 the robot puts it for testing in the inspection equipment, waiting the end of testing (program line 310) and after testing transports it to pallet 2. The realization of the complete program is defined by program cycle number (program line 100).