Warranty of Quality

ED Corporation guarantees the quality of this product.

Matters on Warranty of Quality

Warranty Conditions

The warranty term for this product is 2 years beginning on the date of purchase. If there arise defects or abnormalities from this product during the specified warranty term, you can be serviced by the Customer Support Center of ED Corporation, free of charge.

Customer's Burden of Expenses

Customer shall cover expenses at a minimum charge in the case of the following cases.

1) When the warranty term has expired

※ However, this condition is limited to 5 years after the expiry of the warranty term.

2) When the defect is caused by customer's fault or natural disasters

※ Please be noted that this article shall be applied even during the warranty term.

Limited Warranty

If a customer has arbitrarily disassembled or remodeled this product, we do not have responsibility for the warranty.

Service Guide

For a service application or consultation, please contact Customer Support Center, whose phone number is (031)730-7373 and FAX is (031)730-7317.

※ For safe and correct use of this product, be sure to read this manual before use, and operate the product according to the appropriate handling and operating procedures.

Notice of Change

The appearance or specifications of this product is subject to change without notice, for improvement of performances.

ED Corporation

PART Ⅰ ..... Introduction to Product

1-1. Overview 9

1-2. Cautions 10

1-3. Checking Constituents 10

1-4. Constitution of Product 12

1-4-1. Vertical Articulated Robot (ED-7255) 12

1-4-2. Robot Controller 13

1-4-3. Teaching Pendant 15

1-5. Specifications 17

1-5-1. Robot Body (ED-7255) 17

1-5-2. Robot Controller 17

1-6. Installation 18

PART Ⅱ ..... Teaching Pendant

2-1. Overview 23

2-1-1. Operation Mode 23

2-1-2. Key Function 24

2-1-3. Menu Manipulation 26

2-1-4. Program 26

2-2. Use of Teaching Pendant 27

2-2-1. Main Menu Screen 27

2-2-2. AUTO_RUN_MODE Menu 28

2-2-3. TEACHEDIT_MODE Menu 30

2-2-4. OPERATE_MODE Menu 43

PART Ⅲ ····· **ED-IRS (ED-Industrial Robot Simulator)**

3-1. Program Installation ····· 49

3-1-1. Install ····· 49

3-1-2. Uninstall ····· 52

3-2. Screen Configuration ····· 52

3-2-1. Menus ····· 52

3-2-2. Docking View ····· 52

3-2-3. Program View ····· 54

3-2-4. Arm View ····· 55

3-3. Menus ····· 55

3-3-1. File Menu ····· 55

3-3-2. Edit ····· 57

3-3-3. View ····· 62

3-3-4. Connect ····· 71

3-3-5. Debug ····· 72

3-3-6. Window ····· 75

3-3-7. Help ····· 76

PART Ⅳ ····· **Control Program Syntax**

4-1. Declaration Statement ····· 79

4-2. Control Statement ····· 85

4-3. Robot Command Statement ····· 91

4-4. Kinds of Functions and Operators ····· 102

PART Ⅴ ····· **Program Example**

5-1. Program Example ····· 107

PART VI ▪ ..... Maintenance

PART VII ▪ ..... Host Command Set

PART VIII ▪ ..... Host Program Example

PART 1 Introduction to Product

1-1. Overview

1-2. Cautions

1-3. Checking Constituents

1-4. Constitution of Product

1-5. Specifications

1-6. Installation

PART 1

Introduction to Product

1-1. Overview

A robot system used for real industrial entities is composed of a mechanical arm, a computer-based controller, an end-of-arm tooling, and a teaching pendant. This equipment is made as an education-dedicated system which includes all the elements of a real industrial robot system, i.e., a robot main body, a robot controller, a gripper, a teaching pendant, a program editing software, as shown in Figure 1-1.

In addition, a robot can be controlled by a teaching pendant or ED-IRS control program and simulator so that a user can personally control the robot. This is an experiment and practice device for a robot arm repeatedly movable by the program.

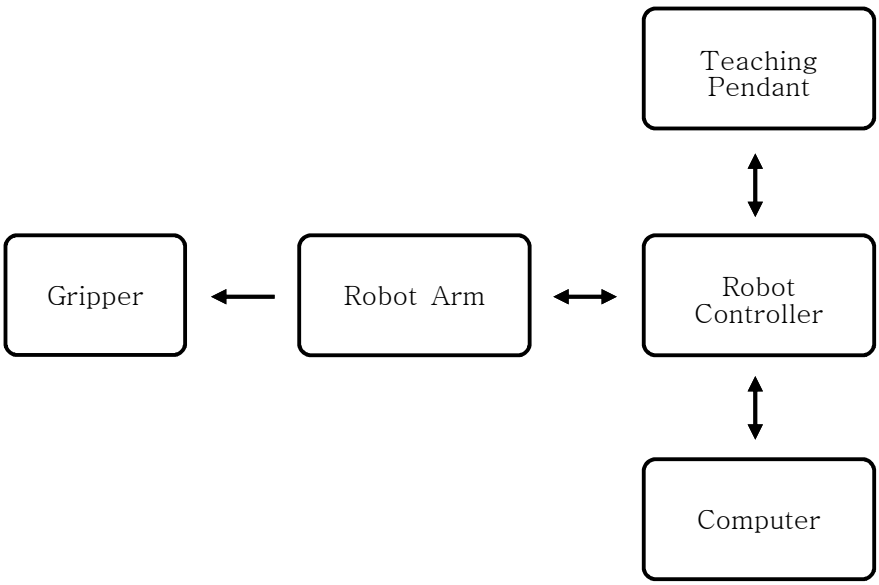


Figure 1-1 Constitution of Robot System

1-2. Cautions

1) Before using this equipment, read this manual and be sure to check the operation procedures and power voltage to be supplied to it.
2) Before powering on the equipment, check whether there are mechanical interferences or the foreign substances are jammed in gears. If there is any foreign substance, remove it.
3) Remove the objects located within the moving radius of a robot.
4) Do not operate DC motors for each joint with a black cap opened.
5) Be sure to check the voltage of a power source before connecting it to the equipment.
6) Avoid using this equipment in a humid place.

1-3. Checking Constituents

Please confirm that the following constituents are all included in the package when you unpack the product.

1) ED-7255 (ARM Robot)
2) Controller
3) Teaching Pendant
4) ED-IRS(Install CD)
5) Cross Cable
6) 9 Pin RS-232
7) Encode Cable
8) 25 Pin Motor Power Cable
9) Ex-Motor Cable 2 pieces
10) Storage Vinyl Bag
11) 220V Power Cord
12) User Manual
13) Textbook for Experiment and Practice

To install ED-IRS in a PC, the following system environment is required.

- O.S. : Windows XP or higher
- Memory : 516MB or more
- HDD : 100MB or more

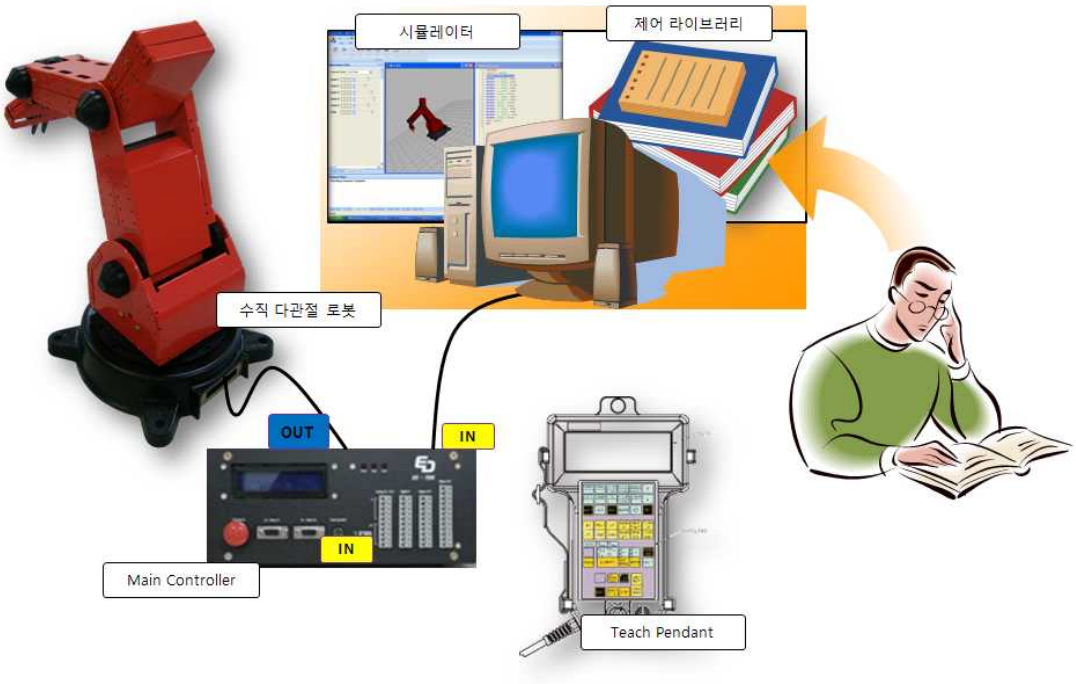


Figure 1-2 Constituents of Product

시뮬레이터: Simulator, 제어 라이브러리: Control Library, 수직 다관절 로봇: Vertical Multi-Joint Robot

- If there is any missing or damaged one of the above components, please contact ED Corporation.
- Version of the supplied program is subject to change for improvement of performance.

1-4. Constitution of Product

1-4-1. Articulated or Multi-Joint Robot (ED-7255)

[Figure 1-3] Motor Positions and Names of ED-7255

팔꿈치: Elbow, 손목 (피치+회전): Wrist (Pitch & Roll), 어깨: Shoulder, 그리퍼: Gripper, 베이스: Base

No.	Subject	Description
1	Number of Joint	5 Joints + Gripper
2	Construction	Vertical Articulated Arm
3	Precision	± 0.5mm
4	Movement Speed	Max. 750mm/sec
5	Load Capacity	1kg
6	Actuator	DC Motor (Magnetic Absolute Encoder Feedback)
7	Motion Range	Body Joint : 350° Shoulder Joint : 120° Elbow Hoint : 135° Elbow Joint Wrist Joint : 180° Wrist Roll : 350°
8	Gripper Opening	80mm (84mm Excluding Rubber Pad)
9	Range of Motion	Max. 680mm
10	Physical Size	Spreaded Form to top : 280x280x786(H)mm Arm Length : 599mm

Table 1-3. Mechanical Specifications of ED-7255

This robot has 5 axes for a base, shoulder, elbow, wrist pitch, and wrist roll, all of which are driven by DC motors. To the wrist, a DC motor-driven gripper is attached.

The 6 DC motors respectively have a reduction gear and an encoder which is attached to a real axis. The reduction gear reduces the rotating speed of a motor to operate a robot. The encoder detects the rotating speed of an axis and the right and left rotation directions. In addition, using Absolute Encoder, the rotating angle can be sufficiently limited by means of software limiting, even though there is not a limit switch.

Primary electrical components of a robot body are as follows.

- **DC Motor: 5 axes and fingers can move right and left, up and down according to the rotation direction of DC motors. The rotation direction of a DC motor varies as being supplied positive or negative voltage from a controller. Additionally,**

each of the 5 axes and the gripper respectively include an absolute encoder therein.

- Encoder : Encoders employed to the 6 motors continuously detect the position of each joint, rotation direction, and rotating speed of a robot arm.

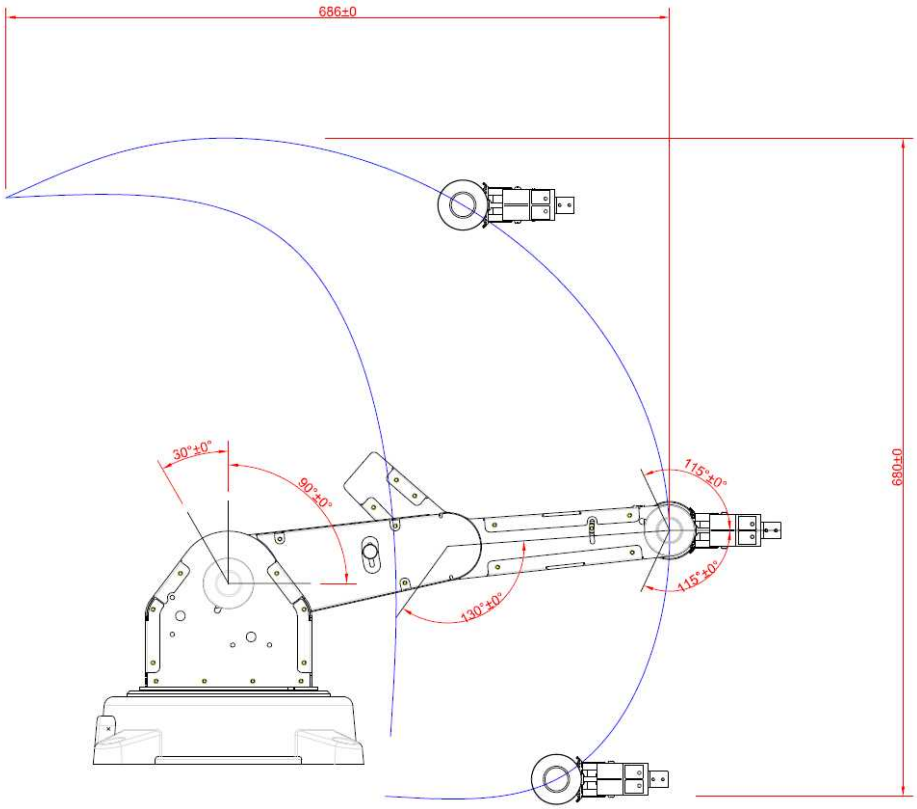
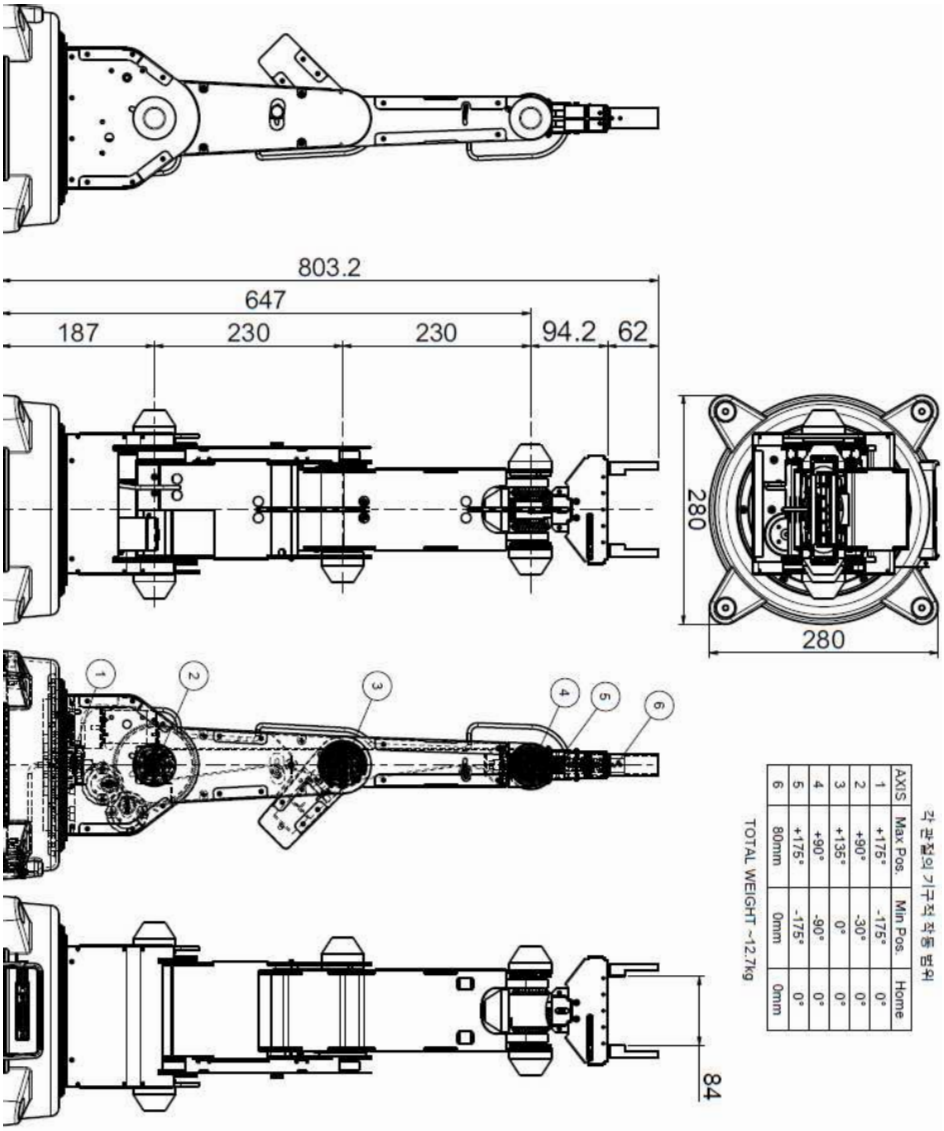


Figure 1-5 Work Space of Vertical Articulated Robot



각 관절의 기구적 작동 범위

AXIS	Max Pos.	Min Pos.	Home
1	+175°	-175°	0°
2	+90°	-30°	0°
3	+135°	0°	0°
4	+90°	-90°	0°
5	+175°	-175°	0°
6	80mm	0mm	0mm

TOTAL WEIGHT ~12.7kg

Figure 1-4 Plans and Measurements of Vertical Articulated Robot

각 관절의 기구적 작동 범위: Mechanical Operation Range of Each Joint

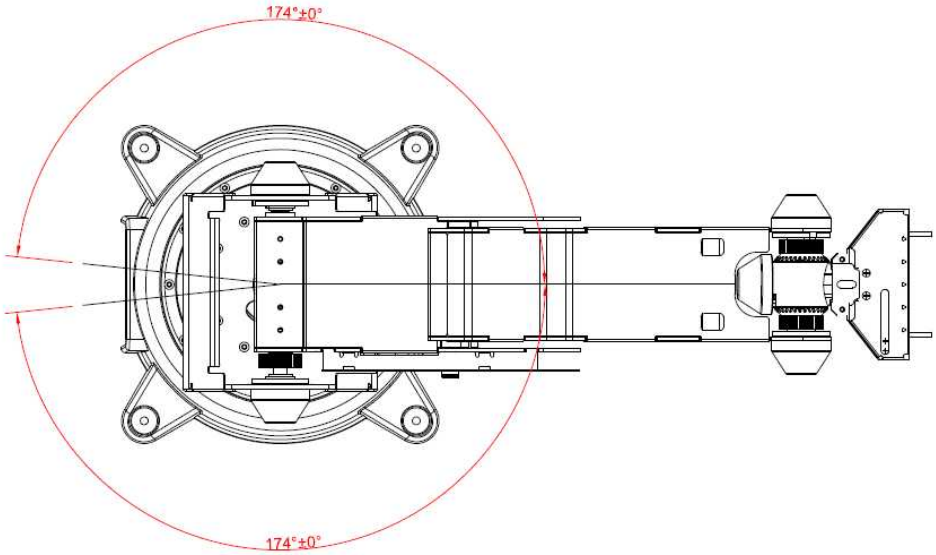


Figure 1-5-1 Work Space of Vertical Articulated Robot

3) Cable

A control box is connected with ED-7255 through a DC motor parallel cable and an encoder parallel cable. A teaching pendant and a control box are connected by RS-232 cable, and PC and a control box are connected by Ethernet cable.

1-4-2. Robot Controller

As in Figure 1-4, a robot controller includes Ethernet port for communication with a computer (host), RS-232 cable port for a teaching pendant, a power port, a motor power port, an encoder port, I/O port, an emergency key, and a state LED.

Figure 1-4-1. Front View of Robot Controller

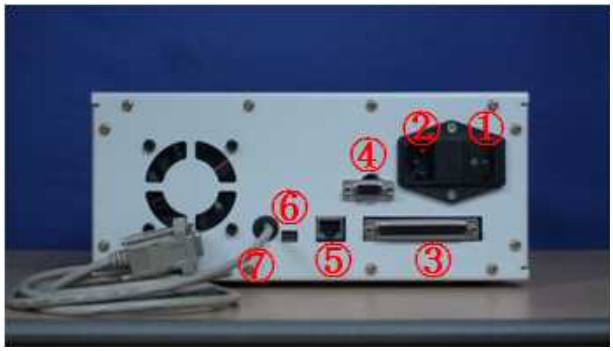


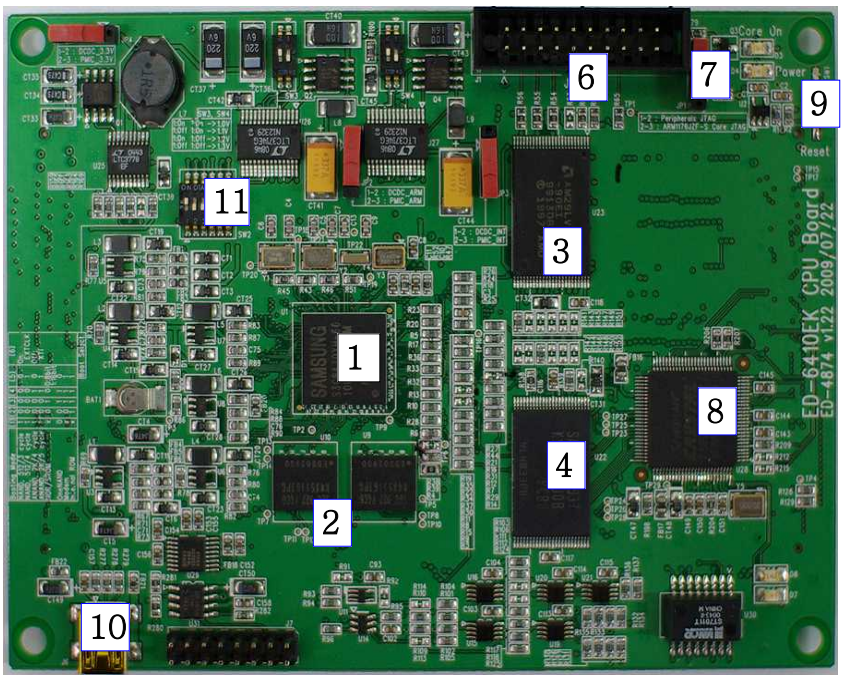
Figure 1-4-2. Rear View of Robot Controller

① Power Switch : Turn on/off a main power source. A state LED lights up after main board booting is completed.
② Main Power : Supply DC 220V power source.
③ Motor Port: A parallel port for supplying a motor of each joint with power.
④ Encoder Port: A serial port for reading encoder values of each joint.
⑤ Ethernet Port: Used for downloading commands, reading the robot status, operating a control library command, etc.
⑥ USB Port: For USB interface. Service is expected.
⑦ Connected to PC through a 9 pin cable to perform a serial communication, and is used for operation of a control library through ARM code.
⑧ EM Button: In an emergency situation, pushing EM button stops the system.
⑨ External I/O Terminals : There are DIO (digital input & output) for peripheral devices, each of which has 8 pins. 4-pin AUX terminal is also provided.

⑩ LCD : The status and the current operation state of a robot is displayed on LCD window.

⑪ Pendant Connector : Connected to a teaching pendant through a 9-pin cable to perform a serial communication.

⑫ Extra Motor Ports: Two extra motor ports are provided for control of peripheral devices.



No.	Name	No.	Name
1	CPU(S3C6410)	7	JTAG
2	mDDR SDRAM	8	Ethernet(CS8900)
3	NOR Flash	9	Reset Switch
4	NAND Flash	10	USB (OTG2.0)
5	SRAM	11	Dip Switch (Bootloader)
6	JTAG Connector		

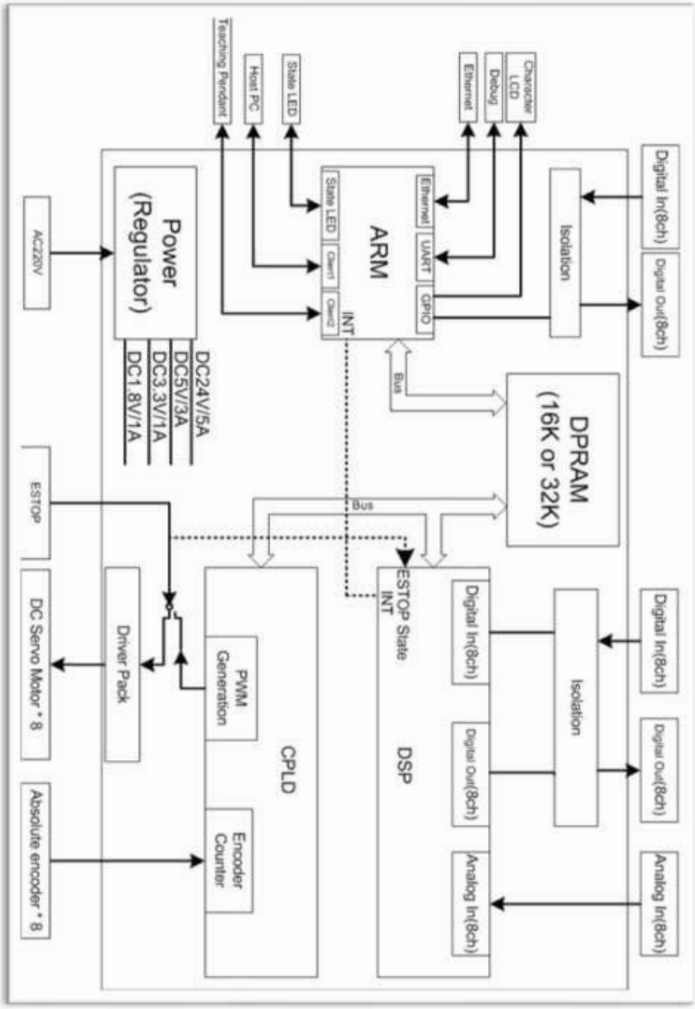


Figure 1-5 Hardware Architecture

Function	Item	Spec	수량
ARM Process	CPU(6410)	400MHz, Min	1
	FLASH	32MByte	
	SRAM	1MByte(32bits Access)	
	DPRAM	16K or 32k	
DSP	TMS320F2811	Digital In(8 Points)	1
		Digital Out(8 Points)	
		Analog In(8-Ch)	
	Motion Control	Closed loop operation	
		Available motor(max 8-axis)	
CPLD	Available PWM Generation	8 axis	1
	Absolute encoder counter	8-ch(PWM method)	
Driver Pack	Operating environment	24V, 120W Max	8
Interface	USB	USB 1,1 A-Type	1
		USB 1,1 B-Type(Host PC)	1
	Ethernet	10Base-T	1
	RS-232		1
	RTC	RTC 4513 Real Time Clock	1
	State LED	Power(3 color), Alarm, Start/stop	3

수량: Q'ty

By using input points, interfacing with external I/O devices or sensors becomes possible. These input points provide a power terminal of 5A, 24V DC to supply the external I/O devices or sensors with the power source.

8 external I/O terminals are used for interface with an external device.

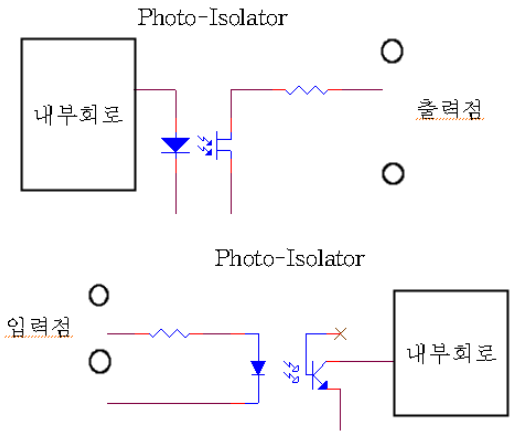


Figure 1-5 Circuit of I/O Points

내부회로: Internal Circuit, 출력점: Output Point, 입력점: Input Point

1-4-3. Teaching Pendant

A teaching pendant is connected to the pendant terminal on a front panel of a robot controller. The teaching pendant is a programming device controlled by a microprocessor and has 29 entry keys. It provides a two-lined LCD display, which consists of 16 characters per each line. Using only the teaching pendant, an application program can be written. Or, it may be used to set robot moving positions in the program of a host computer.

Figure 1-28 Teaching Pendant

Figure 1-28 shows a teaching pendant, which is composed of a key pad, LCD window, an emergency stop key, and a connection connector.

Using the teaching pendant, a complicated program can be written or the written program can be executed. That is to say, a variety of commands may be operated, such as input point check, output point check, delay time setting, program execution order control, etc. The program a user has written is stored in EEPROM of a controller.

The teaching pendant has the following 5 operation modes.

① PLAY Mode : Move a motor. Set an output point. Check an input point.

② EDIT Mode : Identical to the PLAY mode, except a function that commands are stored in a file. Program control commands including conditional or non-conditional branch can be stored. The program file can be displayed, modified, and executed in sequence.

③ RUN Mode : Program files are executed. The program can be infinitely executed by using program control commands, or be stopped according to the external conditions.

④ CONFIG Mode : A system environment setup, such as a left table value of a home of an absolute encoder, maximum value of an encoder, PID gain, etc. may be performed. Set information is stored in EEPROM.

1-5. Specifications

1-5-1. Robot Body (ED-7255)

1) Number of Joint : 5 Joints + Gripper
2) Construction : Vertical Articulated Arm
3) Precision (Position) : ± 0.5mm
4) Movement Speed : 740mm/sec max
5) Load Capacity : 1kg
6) Actuator : DC Motor
7) Motion Range
   - Body Joint : ±175°
   - Shoulder Joint : +80°/-20°
   - Elbow Joint : +130°

- Wrist Pitch : +115°/-115°
- Wrist Roll : +170°/-170°

8) Gripper Opening : 84mm

9) Operation Radius : 655mm (max.)

10) Outer Dimension : Base : 280×280×803.2(H)mm

11) Arm Length : 647mm

1-5-2. Robot Controller

1) Functions

One of the most important functions is a motion control of a vertical articulated, or multi-joint, robot ED-7255. A robot can be moved by calculating the moving distance according to the sampling time for each axis, if the moving speed for a target calculated in ARM core is sent to DSP processor. As other functions, a main controller performs all works related to a vertical articulated robot, such as linkage to PC, command execution by I/O, emergency handling, etc.

1) Processor: ARM(6410) Processor, DSP(2812) Processor, Teaching Pendant 16bit AVR

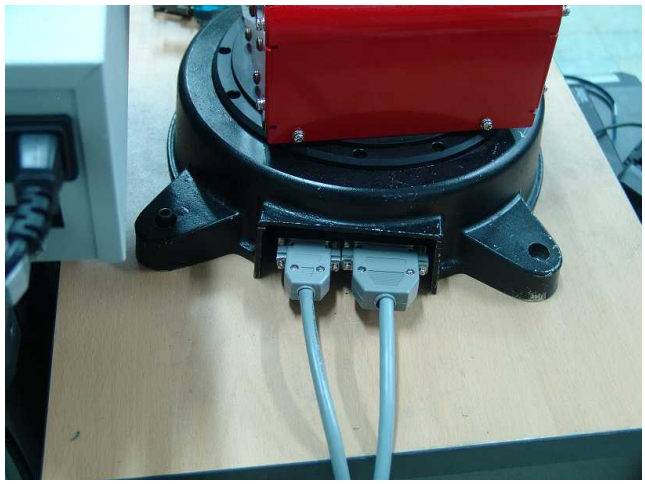
2) Interface: RS-232C Serial Port, Ethernet Port, I/O Port, and ARM Serial Port

3) Software : ED-IRS (Robot Simulator and Control Program Writer)

1-6. Installation

① Fix a robot arm controller (ED-MC1) at a desired location.

② Connect a motor port, located at the rear of a robot base, and a motor port, located at the front bottom of a controller, through a parallel cable.



③ Connect a teaching pendant to the pendant terminal at the bottom of a front panel of a controller, by using a serial cable.

④ Connect a host terminal at the bottom of a front panel of a controller and a LAN port of a computer, using a cross cable.

⑤ Connect AC220V source to a power source terminal at the rear of a controller, through a power cord supplied.

⑥ Fasten the screws of all connector hoods and recheck the overall connection.

☞ **When connecting cables, be sure to cut off the power for all units.**

■ Product Checking ■

① Check that a power LED, which indicates power application, is turned on by pressing a power switch at the bottom of a front panel of a controller.

② If the controller booting is completed, a motor power is enabled as the robot remains the present state.

③ On LCD display of LD-7255, the following message is displayed. If not, check whether a teaching pendant and a controller are normally interconnected, or power is normally supplied to a controller.

```
ED-7255 MODULE v1.0
running...
```

④ Refer to How to Use Teaching Pendant in Part 3, and try to move all axes of a robot arm in the positive/negative directions.

⑤ Try to operate all the 6 joints by using a teaching pendant.

PART 2 Teaching Pendant

2-1. Overview

2-2. Use of Teaching Pendant

PART 2

Teaching Pendant

2-1. Overview

Teaching Pendant comprises 38 keys and an LCD display window of two lines, each of which can display 20 letters. It is connected with a robot controller through a connection cable.

Using Teaching Pendant, a complicated program can be written or the written program can be executed. The user-written program is stored in a controller, and the program provided through the extra software can also be executed.

2-1-1. Operation Mode

Teaching Pendant has four operation modes (AUTO RUN, TEACH EDIT, OPERATE, SYSTEM).

1) AUTO RUN MODE

AUTO RUN MODE is a mode which calls a program stored or written and controls a robot, using the program. Inquiry, selection, or execution of a program can be performed. Program execution includes one-cycle execution, step (line-by-line) execution, and iterative execution.

2) TEACH EDIT MODE

TEACH EDIT MODE is for writing a program. Functions of creation, modification, saving as different name, deletion, etc. can be performed in this mode. Even while in program writing, the position data can be saved by directly moving a robot to the target position by means of manipulation keys of the teaching pendant.

3) OPERATE MODE

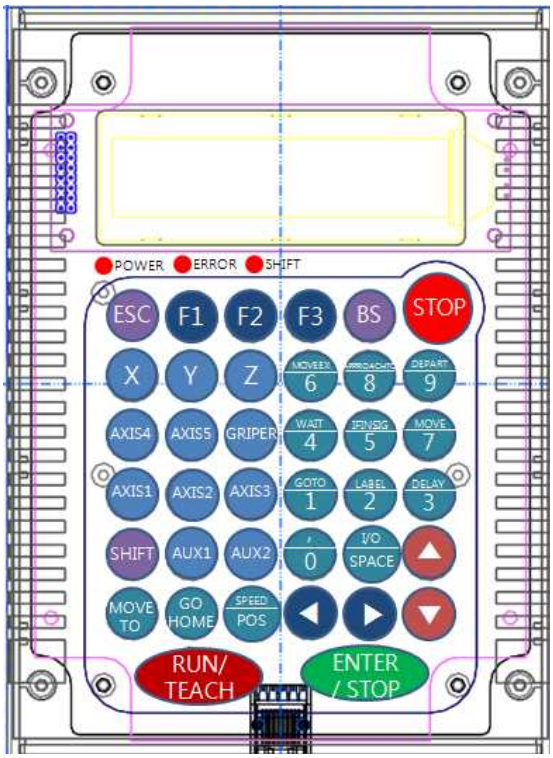
OPERATE MODE is for moving and controlling a robot by using manipulation keys of a teaching pendant.

4) SYSTEM MODE

SYSTEM MODE is a mode for changing or processing the default setting values and environment setting values of a robot.

2-1-2. Key Function

1) Key Arrangement



2) Key Configuration

A key pad of a teaching pendant is divided into the following three

function groups.

For keys, to which two function names are assigned with a division line, if a shift key is not being pressed, the lower named function operates, and if a shift key is being pressed, the upper named function operates.

① Keys for controlling execution, menu, and program writing.

- RUN/TCH : Execute program, and teach a robot while in program writing.
- ENT/STOP : Enter the data and select a menu while in program writing, and stop the program execution.
- UP/DOWN : Move menus, increase the value when teaching a robot, and move lines while in program writing.
- LEFT/RIGHT : Move a cursor while in program writing.
- ESC : Move to the higher menu, cancel the operation, and finish the program writing.
- BS : Delete one letter while in program writing.
- SHIFT : Use double-functioned keys in combination with SHIFT key. Precisely control the robot moving while teaching a robot.
- EMSTOP : For emergency stop of a robot.

② Keys for selecting axes according to the robot coordinate systems.

- X~Z : Select X, Y, or Z axes in a Cartesian coordinate system.
- AXIS1~5 : Select each axis in a joint coordinate system.
- GRIP : Select a gripper
- AUX1~2 : Select a added control axis.

③ Keys for entering commands and numbers in a program.

- MVTO : Select X, Y, or Z axes in a Cartesian coordinate system.
- GOHM : Select each axis in a joint coordinate system.
- SPD/POS : Select a gripper
- I/O / SPACE : Select I/O. Make a space while in program writing.

- ,/0 : Comma and number 0 (zero)
- GOTO/1 : Enter GOTO command and number 1
- LABEL/2 : Enter LABEL command and number 2
- DELAY/3 : Enter DELAY command and number 3
- WAIT/4 : Enter WAIT command and number 4
- IFSIG/5 : Enter IFINSIG command and number 5
- MOVE/6 : Enter MOVE command and number 6
- MVEX/7 : Enter MOVEEX command and number 7
- APRTO/8 : Enter APPROACHTO command and number 8
- DPRT/9 : Enter DEPART command and number 9

2-1-3. Menu Manipulation

LCD of a teaching pendant has 2 lines. Each line displays 20 letters.

[ ] stands for LCD display.

1) How to Use Menu

① Select Menu

Selectable menus can be moved by pressing UP/DOWN key.

```
*1. AUTO_RUN_MODE
 2. TEACHEDIT_MODE
```

=> DOWN =>

```
 1. AUTO_RUN_MODE
*2. TEACHEDIT_MODE
```

```
 1. AUTO_RUN_MODE
*2. TEACHEDIT_MODE
```

=> UP =>

```
*1. AUTO_RUN_MODE
 2. TEACHEDIT_MODE
```

② Execute Menu

A menu can be executed by pressing ENTR/STOP after selecting the menu, or pressing the number key relevant to the menu.

```
*1. AUTO_RUN_MODE
 2. TEACHEDIT_MODE
```

=>ENTR/STOP=>

```
*1. SELECT PROJECT
 2. INPUT PRJ NUM
```

*1. AUTO_RUN_MODE 2. TEACHEDIT_MODE	=>	GOTO/1	=>	*1. SELECT PROJECT 2. INPUT PRJ NUM

2-1-4. Program

A program written by a teaching pendant is composed of a program file, a position data file, and a system configuration file, and is stored on a project basis.

A position data file and a system configuration file are automatically stored and loaded. Thus, a user has only to write a program file.

The program file is a script-based program using robot control commands provided.

Basic structure of the program file is as follows.

```
PROGRAM
...
Unser Control Commands
...
END
```

2-2. Use of Teaching Pendant

2-2-1. Main Menu

1) Main Menu Screen

*1. AUTO_RUN_MODE 2. TEACHEDIT_MODE	3. OPERATE_MODE *4. SYSTEM_MODE

① AUTO_RUN_MODE : A mode which calls a program stored or written and controls a robot, using the program. Inquiry, selection, or

execution of a program can be performed. Program execution includes one-cycle execution, step (line-by-line) execution, and iterative execution.

② TEACHEDIT_MODE : A mode for writing a program. Functions of creation, modification, saving as different name, deletion, etc. can be performed in this mode. Even while in program writing, the position data can be saved by directly moving a robot to the target position by means of manipulation keys of the teaching pendant.

③ OPERATE_MODE : A mode for moving and controlling a robot by using manipulation keys of a teaching pendant.

④ SYSTEM_MODE : A mode for changing or processing the default setting values and environment setting values of a robot.

2-2-2. AUTO_RUN_MODE Menu

1) AUTO_RUN_MODE Menu Screen

```
*1. SELECT PROJECT
 2. INPUT PRJ NUM
```

① SELECT PROJECT : A menu for selecting a project from a list of the already-written program.

② INPUT PRJ NUM : A menu for selecting a project by directly typing the project name. This menu can call only the project written by a teaching pendant. It can select the project number 1 to 999. The project written by a teaching pendant has a name starting with “PRJ”.

2) SELECT PROJECT Selection Screen

If selecting SELECT PROJECT, a list of the project currently being stored in a robot is displayed as follows.

```
*1. PRJ001
 2. IRSSAMPLE
```

If pressing ENTR/STOP key on a project list or directly entering the list number of the pertinent project by using number keys of a teaching pendant, the relevant program is loaded and the following screen appears.

```
PROGRAM
 GOHOME
```

If there is no project stored, the following error screen appears, and by pressing any key of a teaching pendant, a screen is returned to AUTO_RUN_MODE Menu Screen.

```
No Project List
Press Any Key
```

If it is failed to call the selected project, the following error screen appears, and by pressing any key of a teaching pendant, a screen is returned to AUTO_RUN_MODE Menu Screen.

```
Can't Open Project
Press Any Key
```

3) INPUT PRJ NUM Selection Screen

If selecting INPUT PROJECT, a screen for typing a project number is displayed as follows.

```
INPUT PROJECT
PRJ No. :
```

If pressing ENTR/STOP key after typing the project number, the

relevant program is loaded and the following screen is displayed.

```
PROGRAM
 GOHOME
```

If it is failed to call the selected project, the following error screen appears, and by pressing any key of a teaching pendant, a screen is returned to AUTO_RUN_MODE Menu Screen.

```
Can't Open Project
Press Any Key
```

4) Program Run

If the program selected through SELECT PROJECT or INPUT PRJ NO is loaded as a following screen, the program can be executed. The program can be executed by pressing RUN/TCH key.

```
PROGRAM
 GOHOME
```

By pressing RUN/TCH key, the following screen is displayed.

```
*1. 1 CYCLE RUN
 2. STEP RUN
```

```
 2. STEP RUN
*3. ITERATE RUN
```

① 1 CYCLE RUN : Execute the program only one cycle.

② STEP RUN : Execute the program by 1 step on a line-by-line basis. When this STEP RUN operates, the program temporarily stops after execution of one step. If RUN/TCH key is again pressed, the next step is operated and temporarily stopped again.

③ ITERATE RUN : Iteratively execute the program. When this ITERATE RUN operates, the program is continuously executed until a user issues a stop command.

5) Program Stop

To stop the program run while the program is running by RUN/TCH key, press STOP/TCH key. When the program is being run by 1 CYCLE RUN or STEP RUN, it is automatically stopped by the Program Stop. However, if the infinite iteration is performed by a loop statement in the program, it may not be stopped.

If the program is stopped, a screen moves to a screen that the program was loaded.

2-2-3. TEACHEDIT_MODE Menu

1) TEACHEDIT_MODE Menu Screen

*1. NEW PROJECT	3. IMPORT PROJECT
2. LOAD PROJECT	*4. DELETE PROJECT

① NEW PROJECT : Create a project to make a new program.
② LOAD PROJECT : Load a project to change and refer the already-written program.
③ IMPORT PROJECT : Create a new project based upon the already-written program.
④ DELETE PROJECT : Delete the already-created project.

2) NEW PROJECT Selection Screen

If selecting NEW PROJECT, the following screen is displayed. As to a project name, new name available to use is automatically set.

```
PRJNAME : PRJ002
Press Any Key
```

If pressing any key of a teaching pendant, the following program editing window is displayed.

```
PROGRAM
```

3) LOAD PROJECT Selection Screen

If selecting LOAD PROJECT, the following project list screen is displayed.

```
*1. PRJ001
 2. IRSSAMPLE
```

If selecting the project to be loaded from the project list, the following screen is displayed.

```
PRJNAME : PRJ001
Press Any Key
```

If pressing any key of a teaching pendant, the following program editing window is displayed.

```
PROGRAM
 GOHOME
```

4) IMPORT PROJECT Selection Screen

If selecting IMPORT PROJECT, the following project list screen is displayed.

```
*1. PRJ001
 2. IRSSAMPLE
```

If selecting the project to be loaded from the project list, the following screen is displayed. As to a project name, new name available to use is automatically set.

```
PRJNAME : PRJ002
Press Any Key
```

If pressing any key of a teaching pendant, the following program editing window is displayed. As when selecting NEW PROJECT, the program whose contents is empty is created, when selecting IMPORT PROJECT, the contents of the selected program is created as a new project name.

```
PROGRAM
 GOHOME
```

5) DELETE PROJECT Selection Screen

If selecting DELETE PROJECT, the following project list screen is displayed.

```
*1. PRJ001
 2. IRSSAMPLE
```

Select the project to be deleted from the project list and press ENT/STOP key.

6) Writing Program Using Teaching Pendant Commands

Writing a program by selecting NEW/LOAD/IMPORT PROJECT can be done by using commands of a teaching pendant. Use of the commands are as follows.

① GOHOME : Write GOHOME statement. GOHOME statement is a command for moving a robot to the predetermined HOME position.

(1) GOHM Key

```
PROGRAM
 GOHOME
```

(2) ENT/STOP Key

```
GOHOME
```

② MOVETO : Write MOVETO statement. MOVETO statement is a command for moving a robot to a particular absolute coordinate.

If pressing MVTO key, the following interpolation selection screen is displayed.

```
*1. POINT TO POINT
 2. LINE
```

If selecting POINT TO POINT, only "MOVETO" is set up internally.
If selecting LINE, “MOVETO L," is set up internally for linear moving.

After interpolation selection, the following position data setup screen is displayed for entry of the moving position.

```
*1. TEACHING POS
 2. SELECT POS
```

If selecting TEACHING POS, the following coordinate selection screen is displayed. In this screen, the position to be controlled can by selected by directly moving a robot.

```
*1. JOINT SPACE
 2. CARTESIAN SPACE
```

For robot control, refer to 2-2-4 OPERATE_MODE.
If pressing ENT/STOP key after controlling a robot to the position

to be moved, the following screen is displayed.

```
GOHOME
 MOVETO L, JPOS1
```

In a variable JPOS1, robot position at the teaching pont is automatically stored. It is automatically stored in the form of JPOS+number for a joint space, and is in the form of POS+number for Cartesian space.

Press ENT/STOP key to complete the line.

```
MOVETO L, JPOS1
```

If selecting SELECT POS in the above position data setup screen, the position coordinate system selection menu is displayed.

```
*1. JOINT POS
 2. CARTESIAN POS
```

If selecting JOINT POS, a list of the joint space coordinates stored up to now is displayed. If selecting CARTESIAN POS, a list of the Cartesian space coordinates stored up to now is displayed.
If there is not a list of the relevant coordinates, the following error message is displayed.

```
No Position List
Press Any Key
```

③ MOVE : Write MOVE statement. MOVE statement is a command for moving a robot by the particular absolute coordinate. Basic usage is similar to that of MOVETO statement.

MOVE Key (SHIFT + No. 6 Key)

If pressing MOVE key, an interpolation selection screen is displayed as follows.

```
*1. POINT TO POINT
 2. LINE
```

If selecting POINT TO POINT, only "MOVETO" is set up internally.

If selecting LINE, “MOVETO L," is set up internally for linear moving.

After interpolation selection, the following position data setup screen is displayed for entry of the moving position.

After interpolation selection, the following position data setup screen is displayed for entry of the moving position.

※**Caution!** - MOVE command is identical to MOVETO command. However, the data stored by teaching is the relative position data at the position prior to the teaching.

```
*1. TEACHING POS
 2. SELECT POS
```

If selecting TEACHING POS, the following coordinate selection screen is displayed. In this screen, the position to be controlled can by selected by directly moving a robot.

```
*1. JOINT SPACE
 2. CARTESIAN SPACE
```

For robot control, refer to 2-2-4 OPERATE_MODE.

If pressing ENT/STOP key after controlling a robot to the position to be moved, the following screen is displayed.

```
GOHOME
 MOVETO L, JPOS2
```

In a variable JPOS2, the robot's relative position at the teaching position versus the position prior to the teaching was automatically stored.

Press ENT/STOP key to complete the line.

```
MOVETO L, JPOS2
```

If selecting SELECT POS in the above position data setup screen, the position coordinate system selection menu is displayed.

```
*1. JOINT POS
 2. CARTESIAN POS
```

If selecting JOINT POS, a list of the joint space coordinates stored up to now is displayed. If selecting CARTESIAN POS, a list of the Cartesian space coordinates stored up to now is displayed.

※**Caution!** - Since MOVE command uses the relative coordinate, if the data stored as the absolute coordinate from the data stored by teaching, a robot may erroneously operate.

If there is not a list of the relevant coordinates, the following error message is displayed.

```
No Position List
Press Any Key
```

④ DELAY : Write DELAY statement. DELAY statement is a command for standing by a robot for a particular time.

DELAY Key (SHIFT + No. 3 Key)

If pressing DELAY key, a delay time setting screen is displayed as follows.

```
DELAY VALUE :
```

Enter a delay time using numeral keys. Although a SHIFT key is turned on, the numeral keys are available.

```
DELAY VALUE :
 10
```

Press ENT/STOP key to complete entering the delay time.

```
MOVE JPOS2
 DELAY 10
```

Press again ENT/STOP key to complete entering the line.

⑤ LABEL : Write LABEL statement. LABEL statement is a command for adding LABEL to a particular sentence.

LABEL Key (SHIFT + No. 2 Key)

If pressing LABEL key, a label entering screen is displayed as follows.

```
LABEL NAME :
```

Enter a label name by using X/Y/Z keys and numeral keys. E.g.) X1 -> X key & 1 key

```
LABEL NAME :
 X1
```

Press ENT/STOP key to complete entering the label name.

```
DELAY 10
 X1 :
```

After the label name entered, continue writing commands to be written.

⑥ GOTO : Write GOTO statement. GOTO statement is a command for moving control to the sentence the particular label is added.

GOTO Key (SHIFT + No. 1 Key)

If pressing GOTO key, a label entering screen is displayed as follows.

```
INPUT LABELNAME :
```

Enter a label name of the position to be moved by using X/Y/Z keys

and numeral keys. E.g.) X1 -> X key & 1 key

```
LABEL NAME :
 X1
```

Press ENT/STOP key to complete entering the label name.

```
DELAY 10
 GOTO X1
```

Press again ENT/STOP key to complete entering the line.

⑦ IFINSIG : Write IFINSIG statement. IFINSIG statement is a command for writing a conditional statement dependent on the status of a particular digital input signal.

IFSIG Key (SHIFT + No. 5 Key)

If pressing IFSIG key, ON/OFF selection screen is displayed as follows.

```
*1. INSIG ON
 2. INSIG OFF
```

If ON/OFF selection is completed, DI entering screen is displayed as follows.

```
*1. DI 1
 2. DI 2
```

```
*3. DI 3
 4. DI 4
```

If selecting the relevant DI number, a screen for entering additional DI or THEN is displayed as follows.

```
INPUT , OR SPACE
<FINSIG ON DI1
```

Here if pressing COMMA(,) key, DI selection screen is again displayed and Press again if pressing SPC key, DI entry is completed.

```
GOTO X1
<IG ON DI1 THEN
```

After THEN, continue writing commands to be written. The commands to be written is executed when the entered DI is ON; otherwise, the next line of sentence is executed.

⑧ WAIT : Write WAIT statement. WAIT statement is a command for waiting the status change of a particular I/O or time.

WAIT Key (SHIFT + No. 4 Key)

If pressing WAIT key, ON/OFF selection screen is displayed as follows.

```
*1. INSIG ON
 2. INSIG OFF
```

If ON/OFF selection is completed, DI entry screen is displayed as follows.

```
*1. DI 1
 2. DI 2
```

```
*3. DI 3
 4. DI 4
```

If selecting the relevant DI number, a screen for entering additional DI or completing the entry is displayed as follows.

```
INPUT , OR ENTER
<TINSIG ON DI1
```

Here if pressing COMMA(,) key, DI selection screen is again displayed and if pressing SPC key, DI entry is completed.

```
GOTO X1
<TINSIG ON DI1
```

Press again ENT/STOP key to complete entering the line.

⑨ APPROACHTO : Write APPROACHTO statement. APPROACHTO statement is a command for moving a robot to the position departed by a particular distance in the Z-axis direction from a particular position.

APRTO Key (SHIFT + No. 8 Key)

If pressing APRTO key, an interpolation selection screen is displayed as follows. The whole command writing method is similar to that for MOVETO command.

```
*1. POINT TO POINT
 2. LINE
```

If selecting POINT TO POINT, only "APPROACHTO" is set up internally.

If selecting LINE, “APPROACHTO L," is set up internally for linear moving.

After interpolation selection, the following position data setup screen is displayed for entry of the moving position.

```
*1. TEACHING POS
 2. SELECT POS
```

If selecting TEACHING POS, the following coordinate selection screen is displayed. In this screen, the position to be controlled can be selected by directly moving a robot.

```
*1. JOINT SPACE
 2. CARTESIAN SPACE
```

For robot control, refer to 2-2-4 OPERATE_MODE.

If pressing ENT/STOP key after controlling a robot to the position to be moved, the following screen is displayed.

```
APPROACH VALUE :
<ROACHTO L, JPOS1,
```

.

.

If pressing ENT/STOP after entering the approach distance, the following screen is displayed.

```
GOHOME
<ROACHTO L, JPOS1, 10
```

Press again ENT/STOP key to complete entering the line.

⑩ DEPART : Write DEPART statement. DEPART statement is a command for retracting in Z-axis direction a robot by the particular distance from the current position.

DPRT Key (SHIFT + No. 9 Key)

If pressing DPRT key, a screen for entering the departing distance is displayed as follows.

```
DEPART VALUE :
```

If pressing ENT/STOP key after entering the departing distance, the following screen is displayed.

```
GOHOME
 DEPART 10
```

Press again ENT/STOP key to complete entering the line.

⑪ GRIPPER : Write GRIPPER statement. GRIPPER statement is a command for opening or closing a gripper of a robot.

GRIP key

If pressing GRIP key, a gripper operation selection screen is displayed as follows.

```
*1. OPEN
 2. CLOSE
```

If selecting the operation and press ENT/STOP, the following screen is displayed.

```
GOHOME
 OPEN
```

Press again ENT/STOP key to complete entering the line.

⑫ SPEED : Write SPEED statement. SPEED statement is a command for changing the fundamental moving speed of a robot.

SPD Key (SHIFT + No. POS Key)

If pressing SPD key, a speed entering screen is displayed as follows.

```
SPEED VALUE :
```

If pressing ENT/STOP after entering the speed value, the following screen is displayed.

```
GOHOME
 SPEED 50
```

Press again ENT/STOP key to complete the line.

7) Writing Program Using Teaching Pendant Commands

If pressing ESC key after the program writing is completed, the following screen is displayed.

```
1. SAVE&END
2. SAVE&RUN
```

```
2. SAVE&RUN
3. NOSAVE&END
```

If selecting SAVE&END, the project is saved and the writing program is finished.

If selecting SAVE&RUN, the project is saved and the relevant program is executed.

If selecting NOSAVE&RUN, the project is not saved and the writing

program is finished.

Here, if there arises a syntax error in a program when saving it, the following message is produced.

```
Syntax Error
Press Any Key
```

2-2-4. OPERATE_MODE Menu

1) OPERATE_MODE Menu Screen

```
*1. JOINT SPACE
 2. CARTESIAN SPACE
```

① JOINT SPACE : Control a robot in a joint coordinate system.

② CARTESIAN SPACE : Control a robot in a Cartesian coordinate system.

2) JOINT SPACE Selection Screen

If selecting JOINT SPACE, the following screen is displayed.

```
OPERATE : AXIS_1
0, 0, 0, 0, 0
```

If pressing UP/DOWN key, an axis No. 1 of a robot moves and the position data is changed on a screen.

```
OPERATE : AXIS_1
8, 0, 0, 0, 0
```

If keeping UP/DOWN key being pressed, a robot moves continuously. If releasing UP/DOWN key, a robot is stopped.

If pressing AXIS1-AXIS5/GRIP key, the relevant axis can be controlled, respectively.

```
OPERATE : AXIS_5
16, 0, 0, 0, 8
```

3) CARTESIAN SPACE Selection Screen

If CARTESIAN SPACE is selected, the following screen is displayed.

```
OPERATE : AXIS_X
0, 0, 0
```

If pressing UP/DOWN key, a robot moves in the X-axis direction and the position data is changed on a screen.

```
OPERATE : AXIS_X
8, 0, 0
```

If keeping UP/DOWN key being pressed, a robot moves continuously. If releasing UP/DOWN key, a robot is stopped.

If pressing X to Z key, the relevant axis can be controlled, respectively.

```
OPERATE : AXIS_Z
16, 0, 8
```

4) Precise Control

To move a robot more precisely, use SHIFT key.

SHIFT+UP/DOWN : Precise Control

UP/DOWN : Ordinary Control

PART 3 ED-IRS
(ED-Industrial Robot Simulator)

3-1. Program Installation

3-2. Screen Configuration

3-3. Menu

ED-IRS (Industrial Robot Simulator)
산업용 로봇 시뮬레이터

PART 3

ED-IRS (ED-Industrial Robot Simulator)

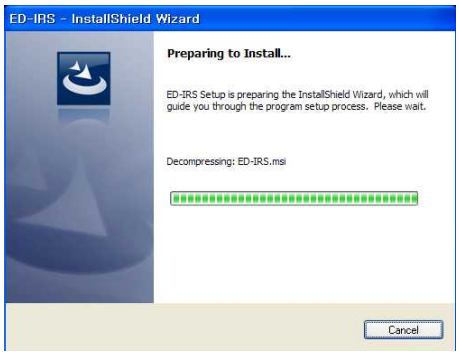
3-1. Program Installation

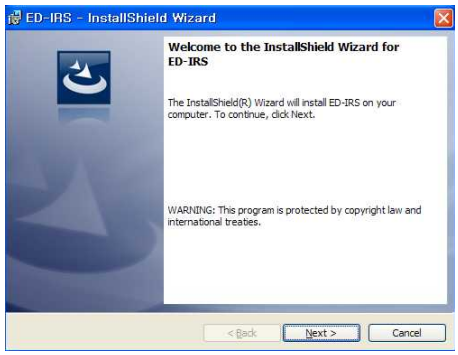
3-1-1. Install

When inserting a CD to install ED-IRS in a PC, the installation program is automatically executed.

If the program is not automatically executed, double click "My Computer" or double click "setup.exe" file in a CD-ROM driver by using Windows Explorer.

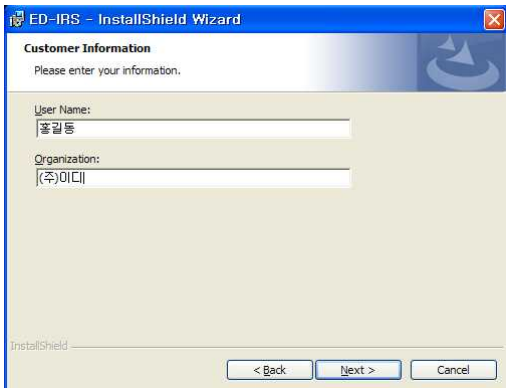
When the installation program starts, Install Shield Wizard is executed as follows.



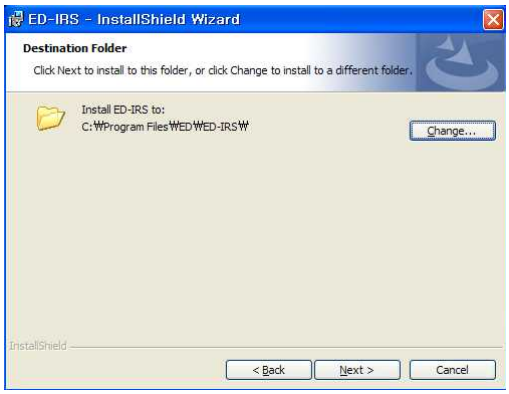


If pressing Next button, the following License Agreement screen appears. Read it fully, and select “I accept the terms in the license agreement”. Next button is activated.

If pressing Next button, a screen for entering user and organization name appears. Type the names.

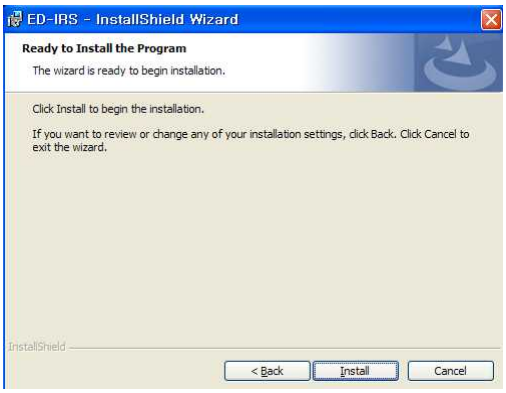


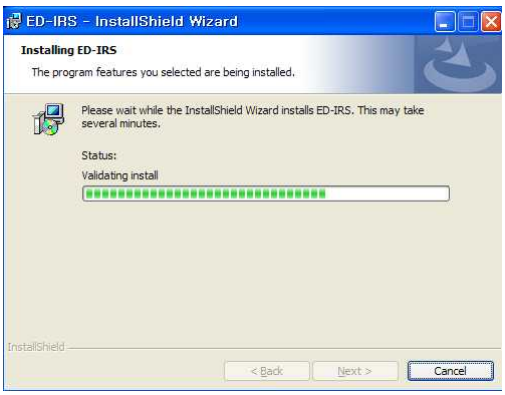
If pressing Next button, a screen for specifying a folder, in which ED-IRS will be installed, appears. Default installation location is “C:\Program Files\ED\ED-IRS\”. If you want to change the location, press ‘Change...’.

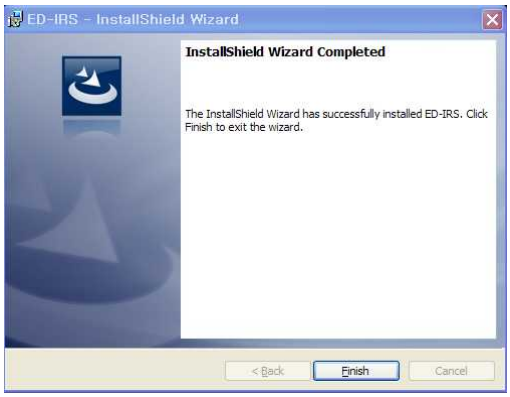


If pressing Next button, the following screen appears. Pressing Install button starts the installation.

On a screen the installation is finished, if pressing Finish button, the installation program is finished and the installation is completed.





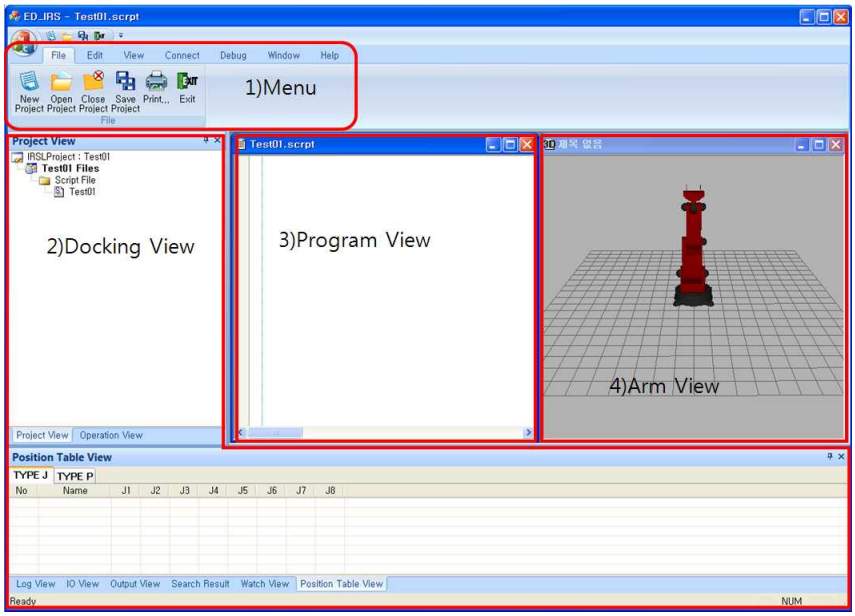


If the installation is completed, a shortcut icon is made on a wallpaper, and an execution file is registered in Start menu.

3-1-2. Uninstall

Uninstallation of the program can be performed through "Program Add/Remove" in Control Panel or through "Uninstall" of ED-IRS item in Start menu.

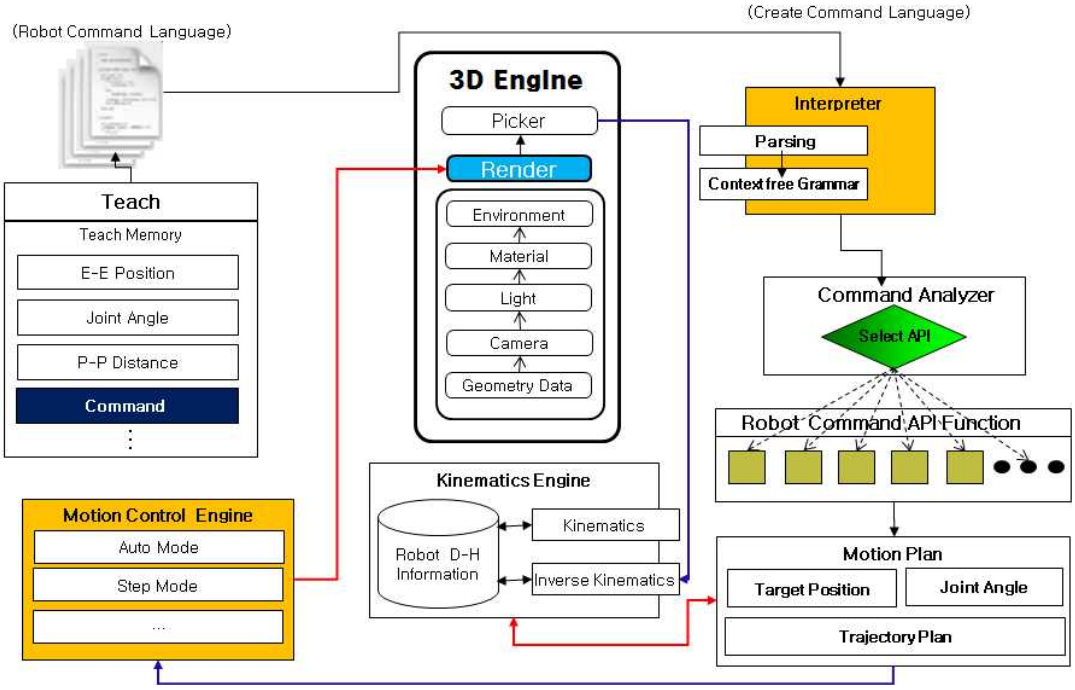
3-2. Screen Configuration



3-2-1. Menus

Menu is composed of 7 menu groups. The function of each menu group is as follows.

- **File : A menu group providing menus for control of files in a project**
- **Edit : A menu group providing menus related to program writing**
- **View : A menu group providing menus for opening and closing views**
- **Connect : A menu group providing menus related to connection to a real robot**
- **Debug : A menu group providing menus for executing and debugging the written program**



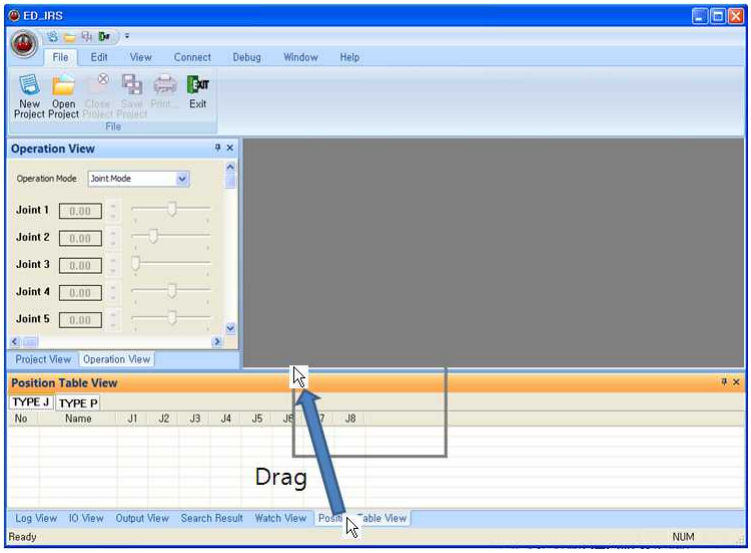
Simulator Software Architecture

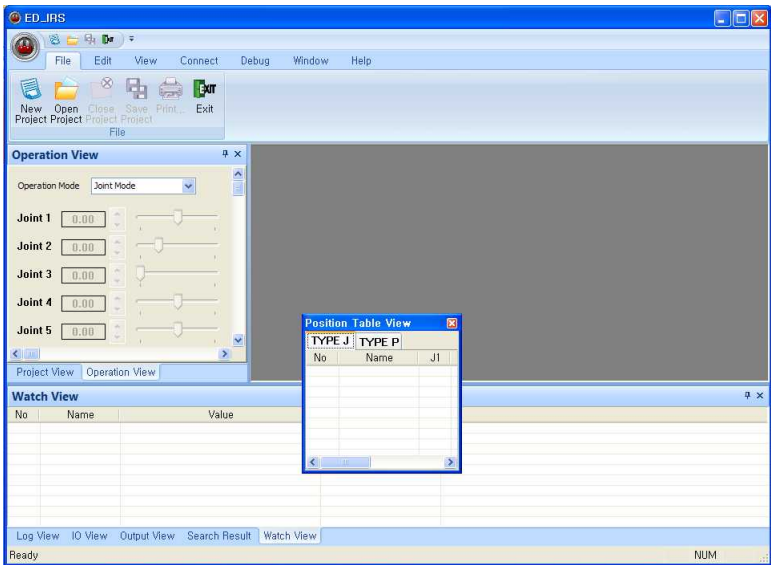
- **Window : A menu group providing menus for configuring the window arrangement**
- **Help : A menu group providing helps for the program**

3-2-2. Docking View

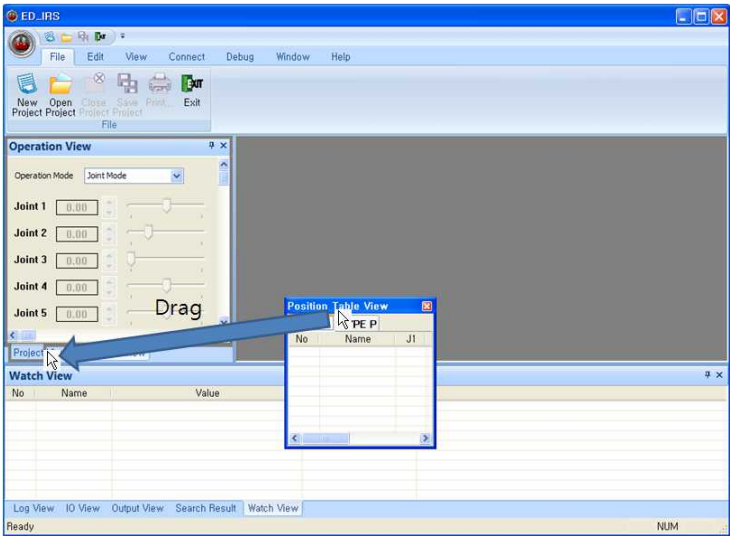
Each view in Docking View can be changed its position or into an independent window.

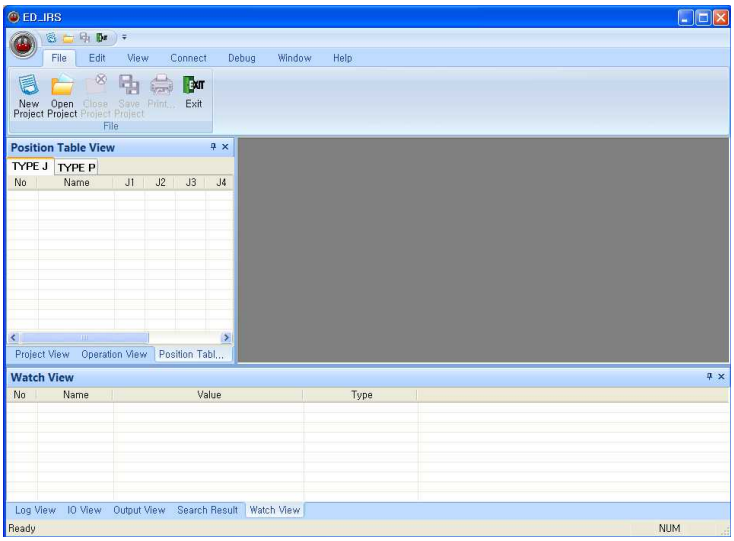
The following drawing shows how to change 'Position Table View' in the docking view into an independent window.





The following drawing shows how to move an independent window and to dock again to the docking view.





Like the foregoing, views in the docking view can be changed its positions and status by a user's desire.

3-2-3. Program View

Program View is an edit window in which a program for the project currently being operated can be written. The program written as the syntax of a robot control language provided by this program view can

operate either a real robot or a virtual 3D robot.

3-2-4. Arm View

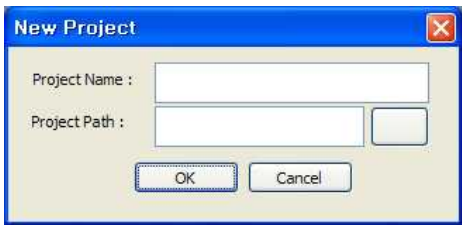
Arm View shows a figure of a virtual 3D robot. It shows that the figure is simulated as moving coordinate of the robot is taught and the program is executed for a target of a virtual 3D robot.

3-3. Menus

3-3-1. File Menu

1) New Project ( )

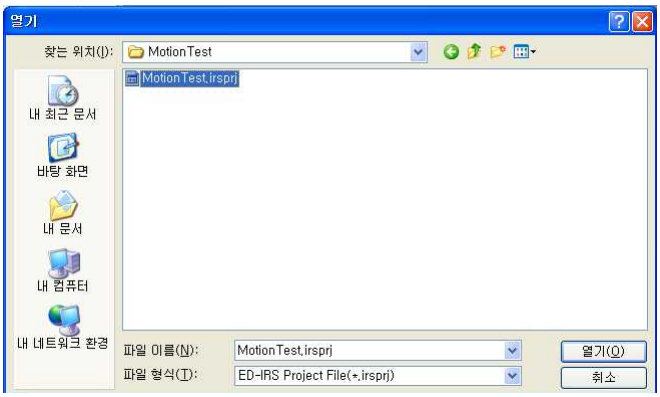
Create a new project. If selecting this menu, a dialog box for project creation appears. By entering a project name and storage path and pressing OK button, a new project is created.



If creation is completed, a program view screen is created, and a program file having a project name entered in Project Name is created.

2) Open Project ( )

Open the already stored project. If selecting this menu, the following Open window appears. By searching an ED-IRS project to be opened and pressing Open button, the project is loaded.



When loading is completed, if there is not a project being currently operated, a program view screen appears, and on the other hand, if there is a project being currently operated, the current project is closed and a loaded project is opened.

3) Close Project ( )

Close the project currently being operated. If selecting this menu, the project currently being operated is finished, and the program view is also closed if it is being opened.

4) Save Project ( )

Save the project currently being operated.

5) Print ( )

Print out the execution contents of the current program view. If selecting this menu, a print window appears as follows. If pressing Confirm button, the contents of the program file is printed by a printer.

6) Exit ( )

Exit the program.

3-3-2. Edit

1) Undo ( ) (Ctrl + Z)

Recover the previous execution in a program view.

2) Redo ( )

Redo the undone execution in a program view.

3) Cut ( ) (Ctrl + X)

Cut the selected contents in a program view.

4) Copy ( ) (Ctrl + C)

Copy the selected contents in a program view.

5) Paste ( ) (Ctrl + V)

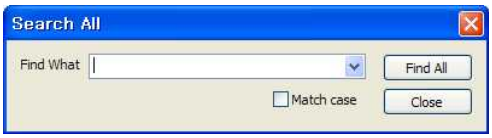
Paste the cut or copied contents in a program view.

6) Select All ( )

Select all contents in a program view.

7) Search All ( )

Search all of the specified words from a program being written, in a program view. If selecting this menu, the following screen appears.



If entering the search word in 'Find What' and pressing 'Find All' button, the following search results are listed in 'Search Result' View. If 'Match Case' is selected, search is performed distinguishably for large and small letters.

If double clicking the result list, the relevant position is shown in 'Program View'.

8) Find ( )

Search sequentially a specified word from a program being written, in a program view. If selecting this menu, the following screen appears.

If entering the search word in 'Find What' and pressing 'Find Next' button, the relevant position is shown in 'Program View'.

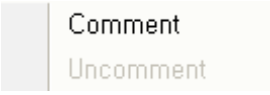
If 'Match whole word only' is selected, search is performed for the exactly coincident whole word divided by blanks.

If 'Match Case' is selected, search is performed distinguishably for large and small letters.

If 'Search Up' is selected from 'Direction' item, search is performed in the upward direction from the current position, and if 'Search Down' is selected, search is performed in the downward direction.

9) Comment Block ( )

A menu for adding or removing the comments to or from the line a cursor is located now. If selecting this menu, the following two sub-menus appear. If the current line is a comment statement, 'Uncomment' menu is activated, and if it is a command statement, 'Comment' menu is activated.



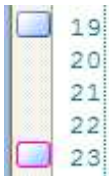
① Comment : The line a cursor is located is changed to a comment statement.

② Uncomment : The line a cursor is located is changed to a command statement.

10) Bookmark ( )

A menu for setting a bookmark to a line a cursor is now located, and for clearing or searching the bookmarks. If selecting this menu, the following four sub-menus appear.

① Toggle : Set or clear a bookmark to or from a line a cursor is located. If a bookmark is set, a symbol "" is marked at the left of 'Program View' as follows.



② Next : Move to search downward the bookmarked positions from a line a cursor is located.

③ Previous : Move to search upward the bookmarked positions from a line a cursor is located.

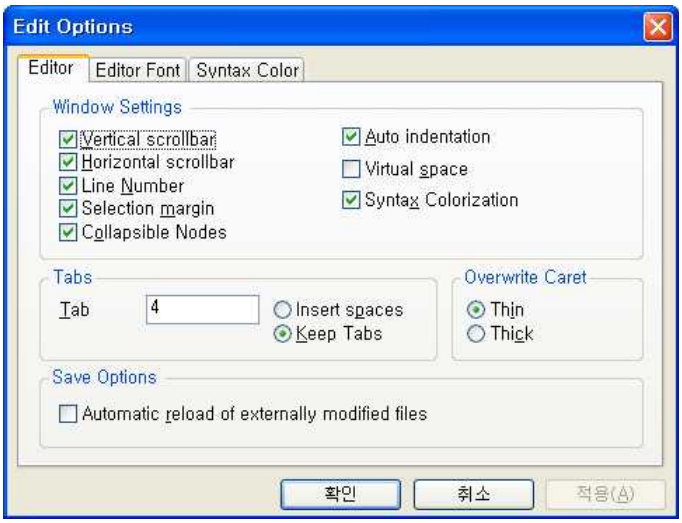
④ Clear All : Clear all bookmarks currently set.

11) Option ( )

A menu for setting the editing environment of 'Program View'. If selecting this menu, the following screen having 3 tabs appears.

① Editor

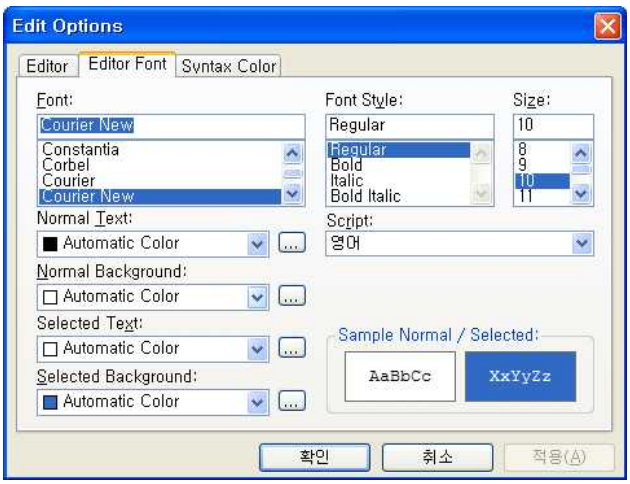
A tab for setting the general environment of an editor including the shape of an editor window, the size of a tab key, etc.



확인: Confirm　취소: Cancel　적용: Apply

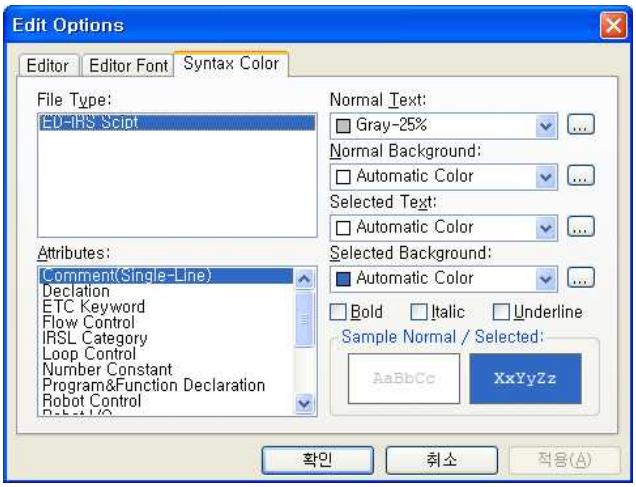
② Editor Font

A tab for setting the font of texts used in the editor window.



③ Syntax Color

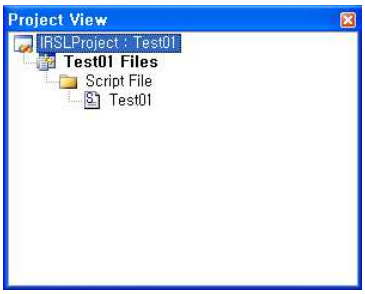
A tab for setting the color or shape of texts according to the syntactic attributes of the program being written in the editor window.



3-3-3. View

1) Project View ( )

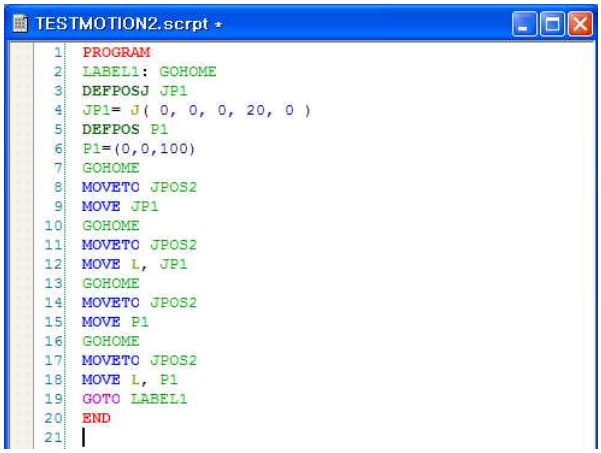
Show 'Project View' window showing the structure of a project currently being operated.



2) Program View ( )

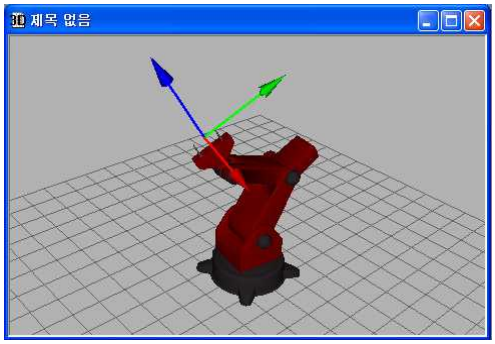
Show 'Program View' window for writing a program of a project currently being operated.

If pressing a mouse right button in 'Program View', the following pop-up menu appears. The items of each pop-up menu is the same as those of a main menu and performs the same functions.

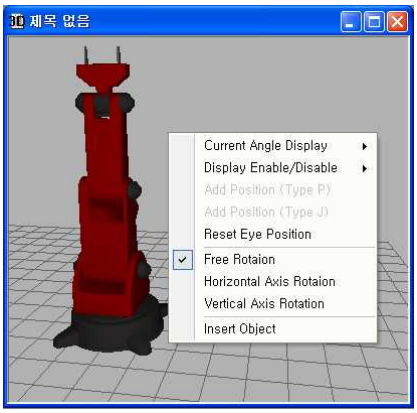


3) Arm View ( )

Show 'Arm View' window for showing a virtual 3D ED-7255 robot. The robot's moving coordinates are taught and stored in 'Arm View', or the motions of a robot during a simulation is shown in 'Arm View'.

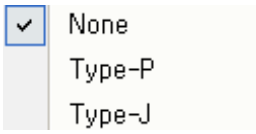


If pressing a mouse right button in 'Arm View', the following pop-up menu appears.

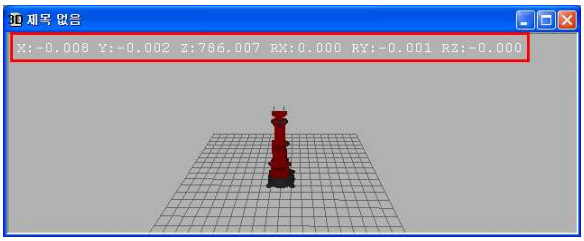


① Current Angle Display

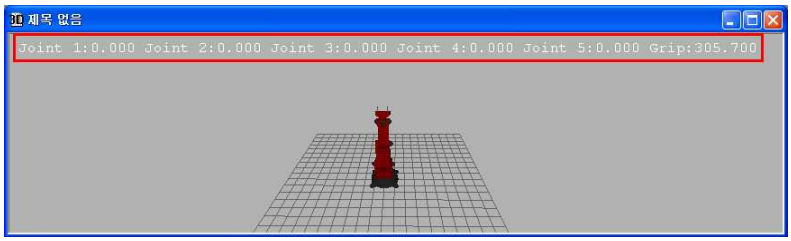
If selecting the relevant pop-up menu, the following 3 sub-menus appear.



- None : Do not show the coordinate value of a robot.
- Type-P : Show the position of an end point, as the Cartesian coordinate value, in the top portion of 'Arm View' window.

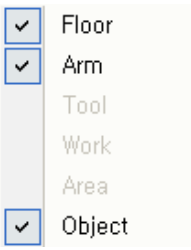


- Type-J : Show the position of an end point, as the joint coordinate value, in the top portion of 'Arm View' window.

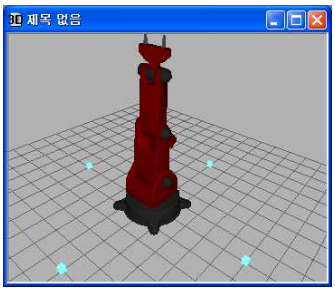


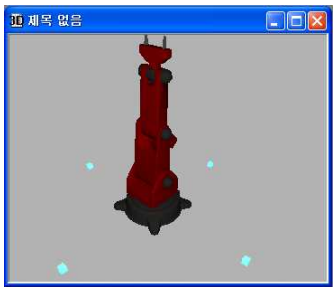
② Display Enable/Disable

If selecting the relevant pop-up menu, the following 6 sub-menus appear.

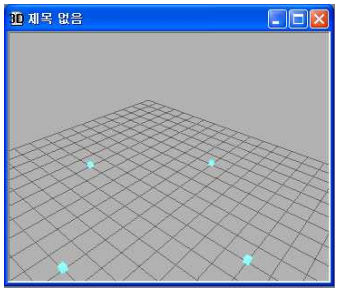


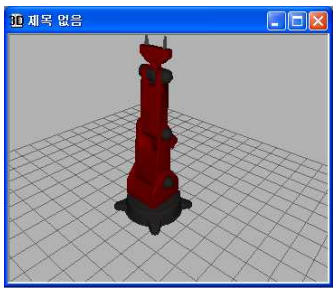
- Floor : Show/Hide cross stripes pattern of a floor.



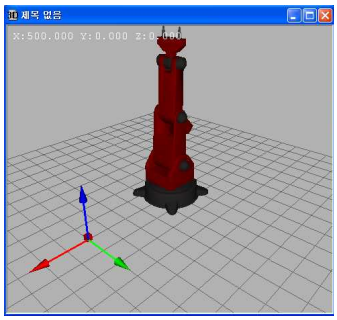


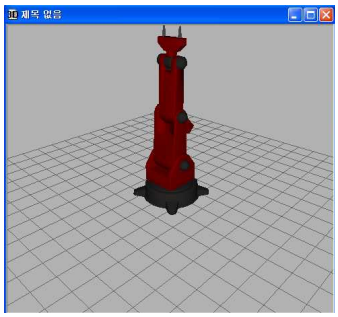
- Arm : Show/Hide a robot figure.





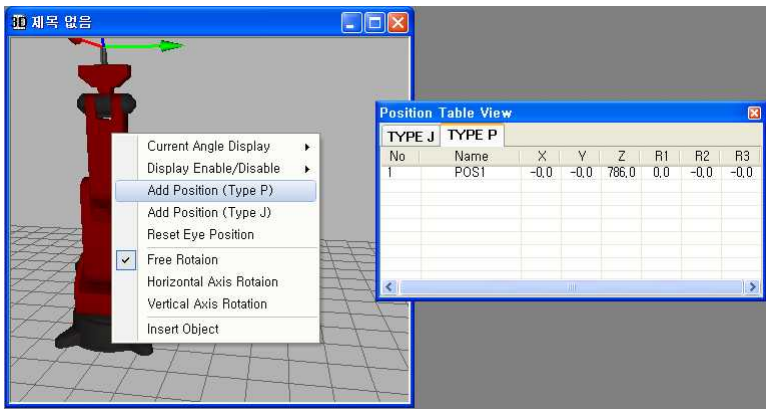
- Object : Show/Hide an object.





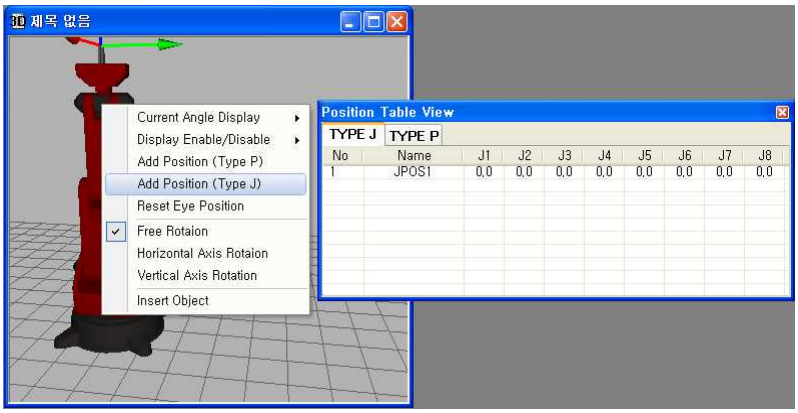
③ Add Position (Type P)

A menu for storing the present coordinate value of a robot as the Cartesian coordinate value. This Add Position item is activated by calling Pop-up menu while a mouse point is being on a robot. If selecting the relevant pop-up menu, the present coordinate value is stored. The stored values can be confirmed at 'Type P' tab in 'Position Table View'.



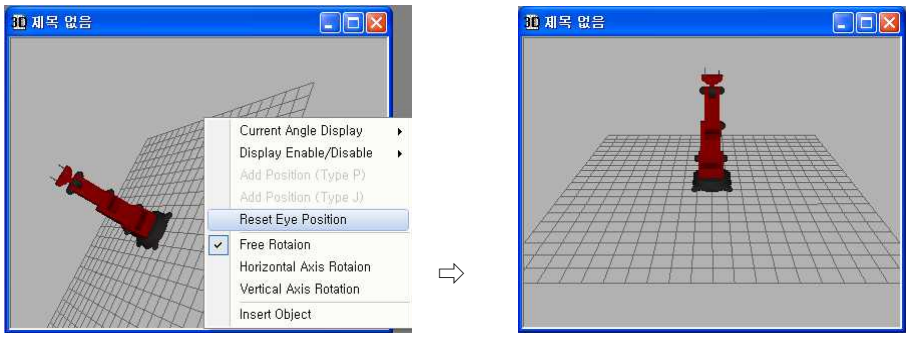
④ Add Position (Type J)

A menu for storing the present coordinate value of a robot as the joint coordinate value. This Add Position item is activated by calling Pop-up menu while a mouse point is being on a robot. If selecting the relevant pop-up menu, the present coordinate value is stored. The stored values can be confirmed at 'Type J' tab in 'Position Table View'.



⑤ Reset Eye Position

A menu for resetting the eye position of the present 'Arm View'. If selecting the relevant pop-up menu, the eye position is reset as follows.

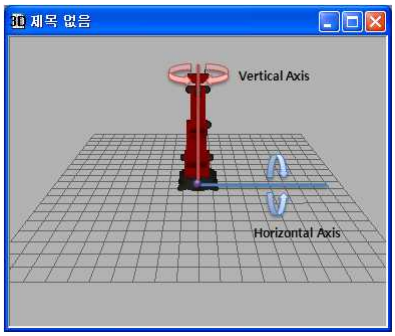


⑥ Free Rotation / Horizontal Axis Rotation / Vertical Axis Rotation

A menu which sets options for changing the eye position of 'Arm View'. One of three options is selectable.

- Free Rotation : Eye position of a camera can be freely adjusted by using horizontal and vertical dragging of a right mouse.
- Horizontal Axis Rotation : Only vertical dragging of a right mouse can be applied to change the eye position of a camera.
- Vertical Axis Rotation : Only horizontal dragging of a right mouse

can be applied to change the eye position of a camera.

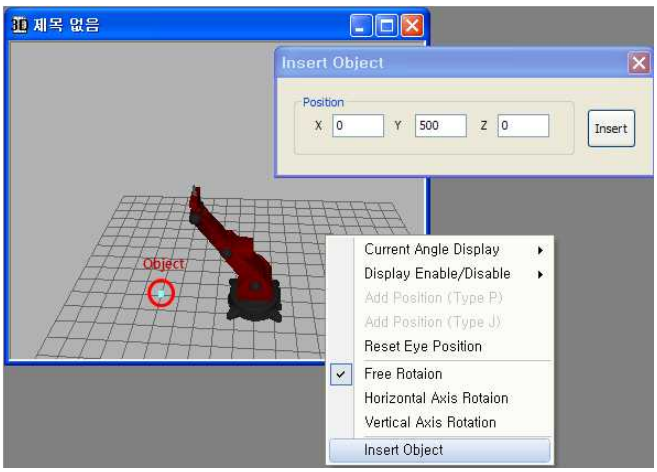


※ **Shift + Vertical dragging of right mouse : Horizontal Axis Rotation**
Ctrl + Horizontal dragging of right mouse : Vertical Axis Rotation

⑦ Insert Object

A menu for adding an object to the environment. If selecting the relevant pop-up menu, 'Insert Object' window appears as follows. If entering the coordinate value of an object to be added and pressing 'Insert' button, it is added to 'Arm View'.

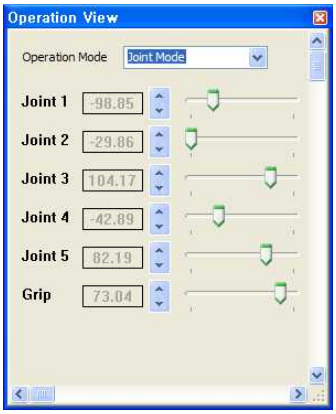
※ **Caution** : Since an origin (0,0,0) is a center coordinate value of a robot, the coordinate of an object can be shown on a screen only if it is departed by 200 or more along X or y-axis.



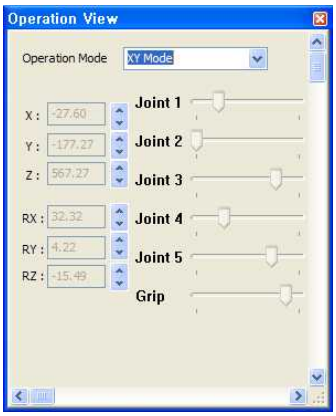
4) Operation View ( )

Show 'Operation View' for controlling a virtual 3D ED-7255 robot. 'Operation View' can control a robot by two 'Operation Modes'

- Joint Mode: A mode for controlling a robot by changing the angle of the respective joints of a robot.

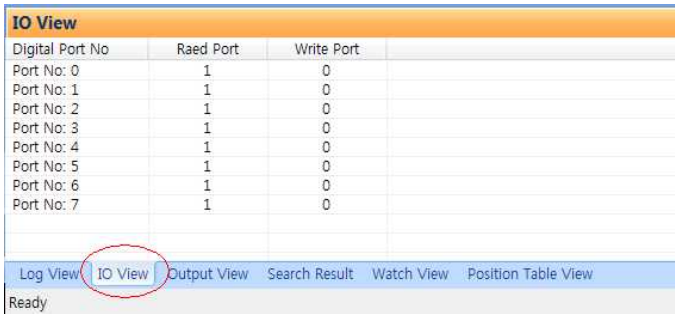


- XY Mode: A mode for controlling a robot by changing the position value of an end point to a Cartesian coordinate system.

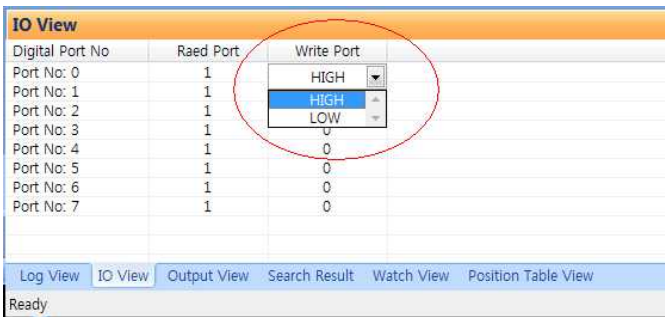


5) IO View( )

Through the following IO View window, 24V signal can be written to and read from an IO port attached to a main controller, under the state that the simulator is on-line with the main controller.



Writing a signal to a digital out of a main controller



6) Log View ( )

Show 'Log View' window for showing the contents of a log of a robot system.

7) Watch View ( )

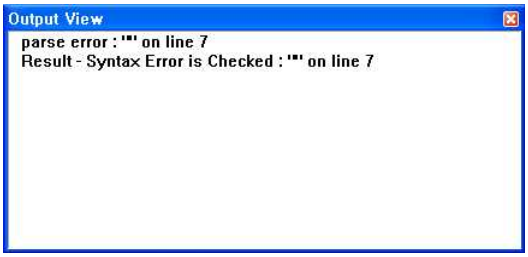
Show 'Watch View' window for showing the status of variables registered in 'Program View'. When the simulation is stopped at a break point while it is being performed, the current value of the relevant variables registered to 'Watch View' is output.

If a right mouse button is clicked in 'Watch View', the following pop-up menu for management of 'Watch View' appears.

- Add Variable : A variable to be watched is added to 'Watch View.' If selecting this menu, a variable named "..." is added to a list of 'Watch View'. By double clicking the relevant name, the variable name is directly entered.
- Delete Variable : A selected variable item is deleted.
- Clear All : All of the variable lists registered are cleared.

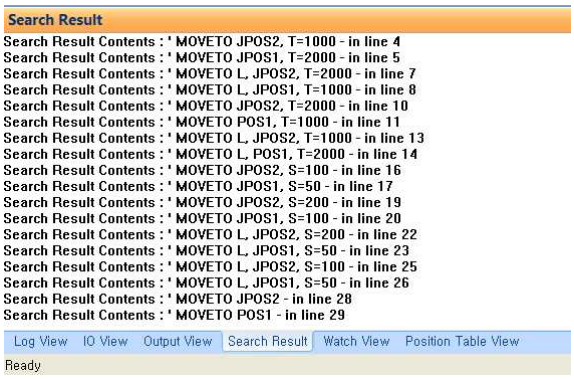
8) Output View ( )

Show 'Output View' window for showing the message output from a system.



9) Search Result ( )

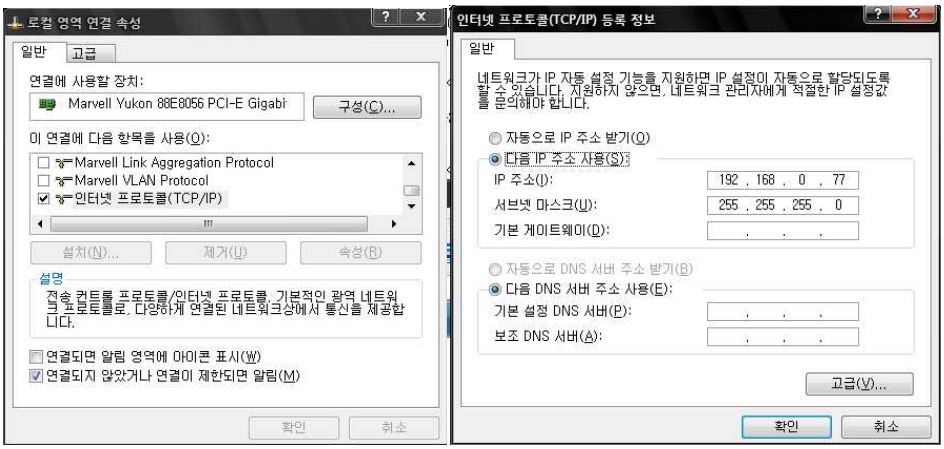
Show 'Search Result' window showing the list of searched result produced by 'Search All'. By double clicking an item of the list, the relevant position is displayed in 'Program View'.



3-3-4. Connect

※ Setting IP

Activate Network Connections of Control Panel, and select a network card connected with the equipment through a cross cable. Then, set IP as follows.

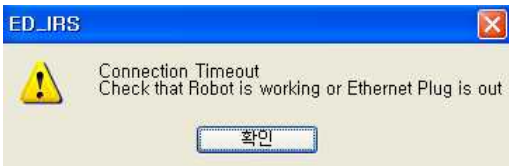


1) Connect ( )

A menu for connecting the communication with a robot so as to control a real robot. If selecting this menu, the following connection status window appears. When connection is normally completed, the status window disappears, and 'Transfer' menu is activated.

Being connected...
Now connection to a robot is being performed. Please wait for a while.

If connection is failed, the following message appears.



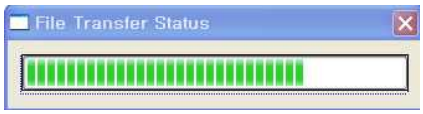
The connection fail may occur if a physical communication wires are not connected or a robot is currently working.

2) Disconnect ( )

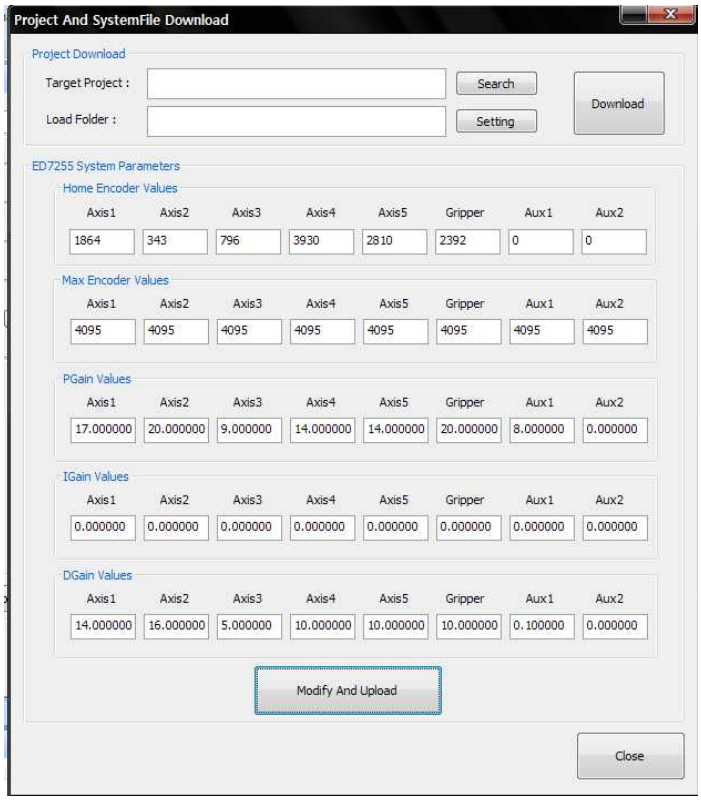
A menu for disconnection of a real robot connected. If selecting this menu, connection is cut off.

3) Transfer ( )

A menu for transferring files of the project currently being operated to a real robot, when connected to a real robot. If selecting this menu, the following file transfer status window appears. If the file transfer is completed, this window is closed.



4) Upload



-Home Encode Values

A home position vertically poses, and the encode values of an absolute encoder relevant to the home position can be changed.

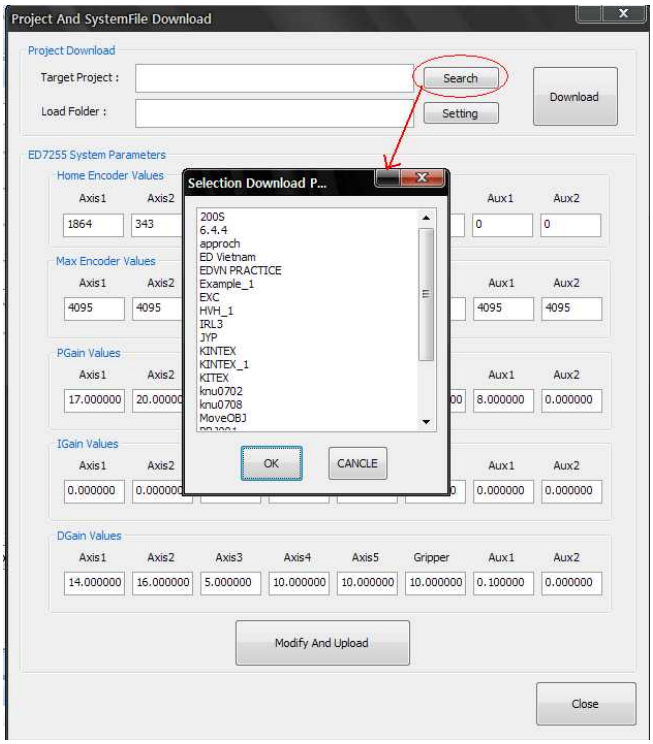
-Max Encode Values

Maximum count value of the encoder can be entered. This can be rapidly performed by changing the maximum encoder value entry even though the encoder is replaced with other type of encoder. At present, the resolution of the absolute encoder is 12 bits, and thus 4095 is the maximum count.

-PGain, IGain, DGain Values

For ED-7255, PID control is applied in order to control the error of the current value versus the target value. Each gain value can be changed.

If all the changes are completed, press Modify And Upload button to apply the changes.



A project to be executed stored in a main controller can be uploaded to PC. Load Folder is a path to store it in PC.

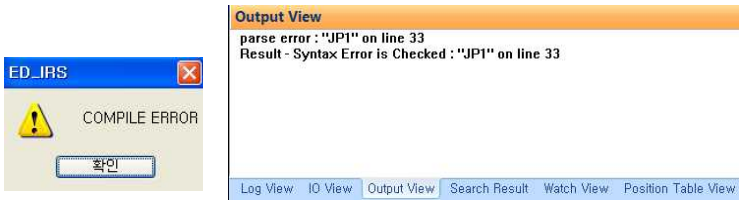
3-3-5. Debug

1) Check Grammar ( )

A menu for checking grammatical errors from the written program. This menu can be used when 'Program View' is activated. Unless this menu is executed, other debugging menus cannot be used. If selecting this menu, the following results are provided according to the execution result.

If a check is normally completed without any grammatical error, the message of “Checking Grammar Complete” is displayed in 'Output View'.

If the grammatical error has occurred, the following message window appears. If pressing 'Confirm' button, the error description is displayed in 'Output View'. The following example shows that a grammatical error is found at a word "JP1" on the 33rd line.



2) Start (▶)

A menu for simulation of the written program for a virtual robot or a real robot. If this menu is selected while being connected to a real robot, the following target robot selecting window appears. If 'Virtual Robot' is selected here, the simulation is performed for a virtual 3D robot, and if 'Real Robot' is selected, the simulation is performed for a real robot.

※ **Caution** : To select 'Real Robot' for simulation, the project currently being operated must have been transmitted to a robot.

If a program writer inserts a break point in 'Program View' or selects 'Pause' menu, execution is temporarily stopped at the point and goes into standby state.

3) Iterate Start ( )

A menu for iteratively executing the written program. This menu is identical to 'Start' menu, from the functional view. The difference is that 'Start' menu executes the program only one cycle, but 'Iterate Start' menu repeats the execution until the stop command is issued by a user.

4) Step In ( )

A menu for executing the written program on a line basis. This menu is identical to 'Start' menu in the functional view, but 'Step In' menu executes one line of an executable command, and then is automatically stopped and goes to a standby mode. If 'Step In' is selected again in the standby mode, the next one line of a command is executed.

5) Continue ( )

A menu for resuming the execution of the program temporarily stopped. If this menu is selected even for the program executed by 'Step In' menu, the function of step (line-by-line) execution is abolished and the execution is performed continuously.

6) Pause ( )

A menu for temporarily stopping or pausing the program being operated by 'Start' or 'Iterate Start'.

7) Program Stop ( )

A menu for stopping the program being operated.

8) Register to Watch ( )

A menu for registering variables of the written program to 'Watch View' so as to debug it. By selecting the name of the variable in 'Program View', this menu is activated. Selecting this menu, the variable is automatically registered to 'Watch View'.

The state of the variable registered to 'Watch View' is updated if it is received 'Step In' or 'Pause' command. At this time, if no value has been yet assigned to the variable, "Not yet assigned" is displayed as follows.

Watch View

No	Name	Value	Type
1	A1	Not yet assigned	VARIANT_TYPE

Log View | IO View | Output View | Search Result | Watch View | Position Table View

If a particular value has been assigned to the variable, the relevant value is displayed as follows.

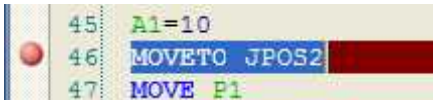
Watch View

No	Name	Value	Type
1	A1	10	INTEGER_TYPE

Log View | IO View | Output View | Search Result | Watch View | Position Table View

9) Set/Reset Break Point ( )

A menu for setting or resetting a break point to or from a line selected in 'Program View'. If selecting this menu, a break point is set to or reset from the line a cursor is now positioned. If the break point is set, is displayed at the left of 'Program View'. Alternatively, if directly clicking a mouse on the left portion of 'Program View' at which is displayed, the break point can be set or reset. If the break point is set, the program is automatically, temporarily stopped at this point while the program is being operated.



10) Reset All Break Point ( )

All set break points are cleared.

3-3-6. Window

A menu for setting the window configuration.

1) Window Cascade ( )

A menu for disposing ‘Arm View’ and ‘Program View’ in a cascade figure.

2) Window Tile Horizontally ( )

A menu for disposing horizontally 'Arm View' and 'Program View' for a maximum-sized window shape.

3) Window Tile Vertically ( )

A menu for disposing vertically 'Arm View' and 'Program View' for a maximum-sized window shape.

3-3-7. Help

1) Help ( )

A menu for showing a help window.

2) Program Info ( )

A menu for showing a program information window.

PART 4 Control Program Syntax

4-1. Declaration Statement

4-2. Control Statement

4-3. Robot Command Statement

4-4. Kinds of Functions & Operators

PART 4

Control Program Syntax

4-1. Declaration Statement

1) PROGRAM/END

① Syntax

```
PROGRAM
...
END
```

② Function Description

Key words of PROGRAM and END stand for a start and end of a program. All the control program must be started with PROGRAM and ended with END.

③ Use Example

```
PROGRAM
...
MOVETO (50,20,40)
...
END
```

2) REM Statement

① Syntax

REM|' <Contents>

<Contents> is an arbitrary text string on one line.

② Function Description

A command to handle the contents as a comment so that particular source codes are not executed in the program, or so as to insert an annotation sentences. All letters between REM and an end of line are not executed.

③ Use Example

The specified sentences of three lines are not executed.

```
...
REM this is comment example
' another comment keyword
REM MOVETO MOVETO (50,20,40)
...
```

3) Integer/Real Number/String Variable Declaration Statement and Assignment Statement

① Syntax

- Integer/Real Number/String Variable Declaration Statement

DIM <Variable Name>[, <Variable Name> ...]

- Integer/Real Number/String Variable Assignment Statement

<Variable Name> = <Constant>|<Variable Name>|<Operation Expression>

<Variable Name> must be started with a character. Others can be started with either a character or a numeral, but not with special symbols.

String constants of <Constant> values must be quoted by double quotation marks (" ").

② Function Description

Used to declare the variables for storing integer, real number, and string values.
The types of variable is determined by the values assigned by the assignment statement.
If declaring many variables, they are to be divided by a comma (,).

③ **Use Example**

Declare variables VAR1 to 4. Assign the integer type value to VAR1, assign the real number value to VAR2, and assign the string type value to VAR3.

```
...
DIM VAR1
DIM VAR2, VAR3, VAR4
REM integer type
VAR1 = 10
REM float type & operation
VAR2 = 10.1+VAR1
REM string type
VAR3 = "string"
REM integer type
VAR4 = VAR1
...
```

4) Position Variable Declaration Statement and Assignment Statement

① Syntax

- Position Variable Declaration Statement

DEFPOS <Variable Name>[, <Variable Name> ...]
DEFPOSJ <Variable Name>[, <Variable Name> ...]

- Position Variable Assignment Statement

<Cartesian Position Variable Name> = <Cartesian Position

Constant>|<Cartesian Position Variable Name>
<Joint Position Variable Name> = <Joint Position Constant>|<Joint Position Variable Name>

- Cartesian Position Constant (x,y,z)

(<Integer>,<Integer>,<Integer>)

- Joint Position Constant (j1,j2,j3,j4,j5)

J(<Integer>,<Integer>,<Integer>,<Integer>,<Integer>)

<Variable Name> has the same restriction as that of <Variable Name> of the item 3) above.

② Function Description

Declare Cartesian/Joint type position values. DEFPOS is used for declaration of Cartesian type position variables; and DEFPOSJ is used for declaration of joint type position variables. If declaring many variables, they are to be divided by a comma (,).
As for Position Variable Assignment Statement, the relevant position constants or variables are assigned according to each position variable type.

③ Use Example

Declare Cartesian type position variable POS1 and joint type position variable JPOS1 and JPOS2, and assign the respective values.

```
...
DEFPOS POS1
DEFPOSJ JPOS1, JPOS2
POS1=(10,10,0)
JPOS1=J(10,10,10,10,10)
JPOS2=JPOS1
...
```

5) Path Variable Declaration Statement and Assignment Statement

① Syntax

- Path Variable Declaration Statement

DEFPATH <Variable Name>[, <Variable Name> ...]
DEFPATHJ <Variable Name>[, <Variable Name> ...]

- Path Variable Assignment Statement

<Cartesian Path Variable Name> =
{ <Cartesian Position Constant>|<Cartesian Position Variable Name> [, <Cartesian Position Constant>|<Cartesian Position Variable Name>] }
<Joint Path Variable Name> = { <Joint Position Constant>|<Joint Position Variable Name> [, <Joint Position Constant>|<Joint Position Variable Name>] }

<Variable Name> has the same restriction as that of <Variable Name> of the item 3) above.

② Function Description

Declare Cartesian/Joint type path values. DEFPATH is used for declaration of paths expressed by Cartesian type position variables; and DEFPOSJ is used for declaration of paths expressed by joint type position variables.
As for Path Variable Assignment Statement, the relevant position constants or variables are assigned with them bound by braces, according to the path variable types.

③ Use Example

Declare a Cartesian type path variable PATH1 and joint type path variable JPATH1 and JPATH2, and assign the respective values.

```
...
DEFPATH PATH1
DEFPATHJ JPATH1, JPATH2
```

PATH1={(10,10,0), (20,20,10)}
JPATH1={J(10,10,10,10,10),JPOS1,J(0,0,10,10,10)}
...

6) LABEL Statement

① Syntax

<Label Name> : <Control Sentence>

<Label Name> has the same restriction as that of <Variable Name> of the item 3) above.

② Function Description

Indicate a label on the relevant sentence. This is used for a target for branch of GOTO statement.
Refer to GOTO Statement for the detailed description of usage of the label statement.

③ Use Example

Indicate a label LABEL1 on an executable statement MOVETO POS1, and move control by using GOTO statement.

...
LABEL1 : MOVETO POS1
...
GOTO LABEL1
...

7) SUB FUNCTION Declaration

① Syntax

```
SUB <Function Name> ()
...
ENDSUB
```

<Function Name> has the same restriction as that of <Variable Name> of the item 3) above.

② Function Description

Declare a sub-function. Declared sub-function is executed by GOSUB statement.
Used at the next of PROGRAM - END statements.

③ Use Example

If VAR1 is greater than 1, move control to a sub-function named SUBFUNC.

```
PROGRAM
...
IF VAR1>1 THEN GOSUB SUBFUNC
...
END
SUB SUBFUNC ()
...
ENDSUB
```

4-2. Control Statement

1) GOTO Statement

① Syntax

GOTO <Label Name>

<Label Name> must be a name declared by LABEL statement.

② Function Description

Branch the execution of a main program to the line that a given label name is specified.

③ Use Example

If VAR1 is greater than 0, move control to the executable statement of MOVETO POS2 that LABEL2 is specified; and if VAR1 is equal to or smaller than 0, move control to the executable statement of MOVETO POS1 that LABEL1 is specified.

```
...
LABEL1 : MOVETO POS1
MOVETO JPOS1, S=200
MOVETO L, (50,20,40)
MOVETO P, J(10,10,0,40,20), S=50
IF VAR1>0 THEN GOTO LABEL2
GOTO LABEL1
LABEL2 : MOVETO POS2
...
```

2) GOSUB Statement

① Syntax

GOSUB <Sub-Function Name>

<Sub-Function Name> must be a name declared by SUB statement.

② Function Description

Move control position to a function that a given sub-function name is specified.

③ Use Example

If VAR1 is greater than 1, move control to a sub-function named SUBFUNC. If the execution of the sub-function is completed by ENDSUB, move control to a line next IF statement.

```
PROGRAM
...
IF var1>1 THEN GOSUB SUBFUNC
...
END
SUB SUBFUNC ()
...
ENDSUB
```

3) IF-THEN-ELSE Statement

① Syntax

IF <Conditional Expression> THEN <Executable Statement> [ELSE <Executable Statement>]

② Function Description

If a conditional expression is true, execute an executable statement next THEN; and if it is false, execute an executable statement next ELSE. In the case ELSE is omitted, execute a sentence of the next line.

③ Use Example

In a first IF statement, if VAR1 is greater than 10, execute MOVETO (50,20,40), and otherwise, execute the next line. In a second IF statement, if VAR2 is equal to or greater than 10, move control to LABEL1, and otherwise, move control to LABEL2.

```
PROGRAM
...
IF VAR1>10 THEN MOVETO (50,20,40)
...
IF VAR2>=10 THEN GOTO LABEL1 ELSE GOTO LABEL2
...
END
```

4) IFINSIG Statement

① Syntax

IFINSIG [ON|OFF] <DI Variable> THEN <Executable Statement> [ELSE <Executable Statement>]

<DI Variable> is a system variable indicating digital input. 8 variables DI1-DI8 can be used for this.

② Function Description

If a compared status value of DI variable is true, execute an executable statement next THEN; and if it is false, execute an executable statement next ELSE. In the case ELSE is omitted, execute a sentence of the next line. In the case ON/OFF is not specified, ON is regarded to be specified as a default value.

③ Use Example

In a first IFINSIG statement, if DI1 is ON, move to LABEL1. In a second IFINSIG statement, if DI1 is OFF, move to LABEL2 and if it is ON, move to LABEL1.

```
...
IFINSIG DI1 THEN GOTO LABEL1
...
IFINSIG OFF DI1 THEN GOTO LABEL2 ELSE GOTO LABEL1
```

5) FOR-NEXT Statement

① Syntax

```
FOR <Variable Name> = <Integer>|<Integer Type Variable> TO
<Integer>|<Integer Type Variable> [STEP <Integer>|<Integer Type
Variable>]
...
[EXITFOR]
...
NEXT
```

② Function Description

Repeatedly execute the executable statement inside between FOR and NEXT statements, until the value that is assigned to the variable declared after FOR becomes equal to or smaller than the number defined after TO. Whenever once repeated, the variable declared after FOR is increased by the value declared after STEP or is increased by 1 if STEP is omitted. If EXITFOR is met, the execution of FOR statement is stopped and control is moved to a line next NEXT.

③ Use Example

Repeat 10 times the execution of IFINSIG DIO THEN EXITFOR and then execute the sentence next NEXT.

```
...
FOR VAR1=1 TO 10 STEP 1
...
```

```
IFINSIG DI0 THEN EXITFOR
...
NEXT
...
```

6) DOWHILE/DOUNTIL-LOOP Statement

① Syntax

```
DOWHILE|DOUNTIL <Conditional Expression>
...
[EXITDO]
...
LOOP
```

② Function Description

For DOWHILE, the executable statement inside between DOWHILE and LOOP is repeated while the conditional expression is true. For DOUNTIL, the executable statement inside between DOUNTIL and LOOP is repeated while the conditional expression is false. That is to say, for DOWHILE, if the conditional expression is false, the execution repetition is stopped, and for DOUNTIL, if the conditional expression is true, the execution repetition is stopped. In addition, if EXITDO is met while being repeated, control is moved to a line next LOOP.

③ Use Example

For DOWHILE, if VAR1 is greater than 10, execution is repeated, and if VAR2 is equal to 10, execution repetition is stopped. For DOUNTIL, the same function as for DOWHILE statement is executed.

```
...
DOWHILE VAR1>10
...
IF VAR2==10 THEN EXITDO
...
LOOP
```

```
...
DOUNTIL VAR1<=10
...
IF VAR2==10 THEN EXITDO
...
LOOP
...
```

7) DO-LOOPWHILE/LOOPUNTIL Statement

① Syntax

```
DO
...
[EXITDO]
...
LOOPWHILE|LOOPUNTIL <Conditional Expression>
```

② Function Description

Repetition conditions of DO-LOOPWHILE/LOOPUNTIL statement is substantially identical to that of DOWHILE/DOUNTIL-LOOP statement. However, the difference is that the inside executable statements are executed at least one cycle when using DO-LOOPWHILE/UNTIL statement. On the other hand, for DOWHILE/DOUNTIL-LOOP statement, the executable statement may not be executed even one cycle, depending on the conditional expression.

③ Use Example

The execution repetition is performed with the same condition as item 6) above. However, even though VAR1 is greater than 10, execution is performed at least one cycle.

```
...
DO
...
```

```
IF VAR2==10 THEN EXITDO
...
LOOPWHILE VAR1>10
...
DO
...
IF VAR2==10 THEN EXITDO
...
LOOPUNTIL VAR1<=10
...
```

4-3. Robot Command Statement

1) MOVETO/MOVE Statement

① Syntax

MOVETO|MOVE [<Interpolation Method>,] <Position Variable>|<Position Constant> [, <Speed Option>]

<Interpolation Method> : P|PTOP|L|LINE
<Speed Option> : S|SPEED = <Integer>

② Function Description

MOVETO is a command to move a robot, based upon the absolute coordinate. MOVE is a command to move a robot, based upon the relative coordinate.

In the case the position coordinate of a robot is currently J(10,10,10,10,10), if executing MOVETO J(10,10,10,10,10), a robot stays in place, and if executing MOVE J(10,10,10,10,10), a robot moves to J(20,20,20,20,20).

If inserting P or PTOP for <Interpolation Method>, a robot is moved on a point-to-point basis; and if inserting L or LINE for it, a robot is linearly moved from the current location to a target location. If it is omitted, P is set by default.

<Position Variable> or <Position Constant> is a target coordinate for move. Pre-stored position variable may be used or the position coordinate may be directly entered.

<Speed Option> can define the moving speed by the percentage, whose unit is %, of a basic moving speed.

③ Use Example

```
...
DEFPOS POS1
DEFPOSJ JPOS1
POS1= (10,10,0)
JPOS1=J(10,10,10,10,10)
MOVETO POS1
MOVETO JPOS1, S=200
```

```
MOVE L, (50,20,40)
MOVE P, J(10,10,0,40,20), S=50
...
```

2) MOVEPATHTO/MOVEPATH

① Syntax

MOVEPATHTO|MOVEPATH [<Interpolation Method>,] <Path Variable> [, <Speed Option>]

<Interpolation Method> : P|PTOP|L|LINE
<Speed Option> : S|SPEED = Integer

② Function Description

MOVEPATHTO is a command to move a path of a robot, based upon the absolute coordinate. MOVEPATH is a command to move a path of a robot, based upon the relative coordinate.
If inserting P or PTOP for <Interpolation Method>, a robot is moved on a point-to-point basis; and if inserting L or LINE for it, a robot is linearly moved to a target location. If it is omitted, P is set by default.
<Path Variable> is a path-move variable defined as DEFPATH or DEFPATHJ.
<Speed Option> can define the moving speed by the percentage, whose unit is %, of a basic moving speed.

③ Use Example

In the case the current coordinate is (0,0,0) for a Cartesian coordinate system and it is J(0,0,0,0,0) for a joint coordinate system, MOVEPATHTO command statement lets a robot move through a path (0,0,0) → (10,10,0) → (20,20,10) → (10,10,0), and MOVEPATH command statement lets a robot move through a path J(0,0,0,0,0) → J(10,10,10,10,10) → J(30,30,30,30,30) → J(40,40,40,40 ,40).

```
...
DEFPATH PATH1
```

```
DEFPATHJ JPATH1
POS1= (10,10,0)
JPOS1=J(10,10,10,10,10)
PATH1 = {POS1, (20,20,10),POS1}
JPATH1 = {JPOS1,J(20,20,20,20,20),JPOS1}
MOVEPATHTO PATH1
MOVEPATH JPATH1
...
```

3) APPROACHTO/APPROACH Statement

① Syntax

APPROACHTO|APPROACH [<Interpolation Method>,] <Position Variable>|<Position Constant>, <Approaching Distance> [, <Speed Option>]

<Interpolation Method> : P|PTOP|L|LINE
<Approaching Distance> : Distance which must be departed along z-axis.
<Speed Option> : S|SPEED = Integer

② Function Description

APPROACHTO is a command to move a robot to the location that is departed from the approaching distance along the z-axis from a target position, based upon the absolute coordinate. APPROACH is based upon the relative coordinate.

③ Use Example

Move to the coordinate (10,10,20), (10,10,30).

```
...
DEFPOS POS1
POS1= (10,10,10)
APPROACHTO POS1, 10
APPROACH L, POS1, 20, S=100
```

...

4) DEPART Statement

① Syntax

DEPART [<Interpolation Method>,] Approaching Distance [, <Speed Option>]

<Interpolation Method> : P|PTOP|L|LINE
<Speed Option> : S|SPEED = Integer

② Function Description

DEPART is a command to move a robot to the location that is departed from the approaching distance along the z-axis from the current location. Use of Interpolation Method and Speed Option are the same as those of MOVETO/MOVE.

③ Use Example

If the current coordinate is (10,10,10), a robot is moved to (10,10,20), (10,10,30).

```
...
DEPART 10
DEPART L, 10, S=100
...
```

5) HOME Statement

① Syntax

HOME|SETHOME <Joint Position Constant>|<Joint Position Variable>

② Function Description

HOME or SETHOME sets a particular joint position to a software-based home position of a robot. Set software-based home

position will be a position to be moved when using GOHOME statement.

③ **Use Example**

J(0,0,0,0,0) is set to a software-based home position.

```
...
JPOS1 = J(0,0,0,0,0)
HOME JPOS1
...
```

6) GOHOME Statement

① Syntax

GOHOME

② Function Description

Move a robot to a software-based home position.

③ **Use Example**

A robot is moved to a position J(0,0,0,0,0).

```
...
JPOS1 = J(0,0,0,0,0)
HOME JPOS1
...
GOHOME
...
```

7) MOVEEX Statement

① Syntax

MOVEEX START <Extra Motor Variable>
MOVEEX END <Extra Motor Variable>
MOVEEX <Extra Position Constant>|<Extra Position Variable>

<Extra Motor Variable> can uses M7 or M8.
<Extra Position Constant> = (Integer, Integer)
<Extra Position Constant> is configured to the size for moving M7 and M8.

② Function Description

Enable and disable an extra axis (a conveyor) except a robot. MOVEEX START enables the extra axis, and MOVEEX END disables the extra axis. EXTRA Motor Variable may use M7 or M8. EXTRA Position Constant can be used to execute MOVEEX in order to move two extra axes or to move it by a given amount.

③ Use Example

When a conveyor is connected to M7, if DI1 is ON even though a conveyor is being driven, it is stopped. The last executable statement moves a conveyor only by 100.

```
...
MOVEEX START M7
...
IF INSIGON DI1 THEN MOVEEX END M7
...
MOVEEX (100,0)
...
```

8) OPEN/CLOSE Statement

① Syntax

OPEN [<Gripper Interval>]

CLOSE [<Gripper Interval>]

② Function Description

A command relating to the operation of a robot gripper. OPEN is for gripper opening and CLOSE is for gripper closing. If the interval of a gripper is not specified, the gripper can be opened at its maximum extent, and if it is specified, the gripper can be opened up to the relevant extent. The unit of the interval is 'mm'.

③ Use Example

Open a gripper to its maximum extent and move it to the position JPOS1. Then, close the gripper with the distance being remained by 10 cm.

```
...
OPEN
MOVETO JPOS1
CLOSE 100
...
```

9) SET/RESET Statement

① Syntax

SET|RESET ALL|<DO Variable>[, <DO Variable> ...]

<DO Variable> may use DO1-DO8.

② Function Description

A command to control a digital output signal. SET sends a signal with 1 and RESET, with 0. If ALL is used, a signal is sent to all the digital outputs. If a particular DO variable is declared, a signal is sent to only the declared digital output. DO variables are available for DO1-DO8.

③ Use Example

Send OFF signals to all digital outputs. Move a gripper to JPOS1 and

close the gripper. Then send ON signal to DO1 and DO2.

```
...
OPEN
RESET ALL
MOVETO JPOS1
CLOSE 100
SET DO1,DO2
...
```

10) INDSIG Statement

① Syntax

INDSIG <Variable Name> = <DI Variable>

<DI Variable> may use DI1-DI8.

② Function Description

A command to take a digital input signal. The relevant digital input value is assigned to a specified variable. If the signal is ON, 1 is assigned; otherwise, 0 is assigned.

③ Use Example

If DI1 is ON, move to JPOS1.

```
...
INDSIG DIVAR1 = DI1
IF DIVAR1==1 THEN MOVETO JPOS1
...
```

11) OUTASIG Statement

① Syntax

OUTASIG <AO Variable Name> = <Integer>

<AO Variable> may use AO1 and AO2.

② Function Description

A command to provide an analog output with a specific integer value. Integer value is sent to an analog output relevant to the specified AO variable.

③ Use Example

If DI1 is ON, data 10 is sent to AO1.

```
...
INDSIG DIVAR1 = DI1
IF DIVAR1==1 THEN OUTASIG AO1 = 10
...
```

12) INASIG Statement

① Syntax

INASIG <Variable Name> = <AI Variable Name>

<AI Variable> may use AI1-AI4.

② Function Description

A command to take an analog input signal to a variable. The analog input value relevant to the specified AI variable is assigned to a variable.

③ Use Example

If AI1 value is greater than 10, move to JPOS1.

```
...
INASIG AIVAR1 = AI1
IF AI1>10 THEN MOVETO JPOS1
...
```

13) SPEED/ACCEL Statement

① Syntax

SPEED | SETSPEED <Integer>
ACCEL | SETACCEL <Integer>

② Function Description

SPEED is a command to set the basic operation speed of a robot. ACCEL is a command to set the basic operation of acceleration and deceleration speed of a robot.

③ Use Example

Basic speed value is set to 100. Basic acceleration and deceleration speed value is set to 100.

```
...
SETSPEED 100
SETACCEL 100
...
```

14) DELAY Statement

① Syntax

DELAY <Time>
<Time> is based upon a unit of 1/10 second.

② Function Description

Robot control progression is delayed by a set time.

③ Use Example

Move to JPOS1, do nothing for one seconds, and then move to JPOS2.

```
...
MOVEEX JPOS1
DELAY 10
```

```
MOVEEX JPOS2
...
```

15) WAITINSIG Statement

① Syntax

WAITINSIG ON|OFF ALL|<DI Variable>[,<DI Variable>...] [, TIMEOUT = <Time> [, <Variable Name>]]

<Time> is based upon a unit of 1/10 second.

② Function Description

Wait a robot control progression for a prescribed time, until specified DI variables are all ON or OFF.
If TIMEOUT option is not specified, it waits infinitely. When the variable is specified in TIMEOUT option, if WAIT is completed due to TIMEOUT, 1 is assigned; otherwise, 0 is assigned.

③ Use Example

Move to JPOS1, if DI1 and DI2 are all ON, or move to JPOS2 if it takes 5 seconds. Wait infinitely until the signals of DI variables are all OFF, and if all are OFF, move to JPOS1. Again, wait until DI1 is ON. If DI1 is ON, move to LABEL2, and if DI1 is not ON for 5 seconds, move to LABEL1.

```
...
MOVEEX JPOS1
WAITINSIG ON DI1,DI2, TIMEOUT = 50
MOVEEX JPOS2
WAITINSIG OFF ALL
MOVEEX JPOS1
WAITINSIG ON DI1, TIMEOUT=50, TVAR1
IF TVAR1==1 THEN GOTO LABEL1 ELSE GOTO LABEL2
...
```

4-4. Kinds of Functions and Operators

1) Kinds of Operators

① Relative Operators

< : smaller than
> : greater than
>= : equal to or greater than
<= : equal to or smaller than
== : equal to
!= : not equal to

② Arithmetic Operators

+ : addition
- : subtraction
* : multiplication
/ : division
MOD : remainder

③ Logical Operators

AND : logical product
OR : logical sum
XOR : exclusive logical sum

2) Kinds of Arithmetic Functions

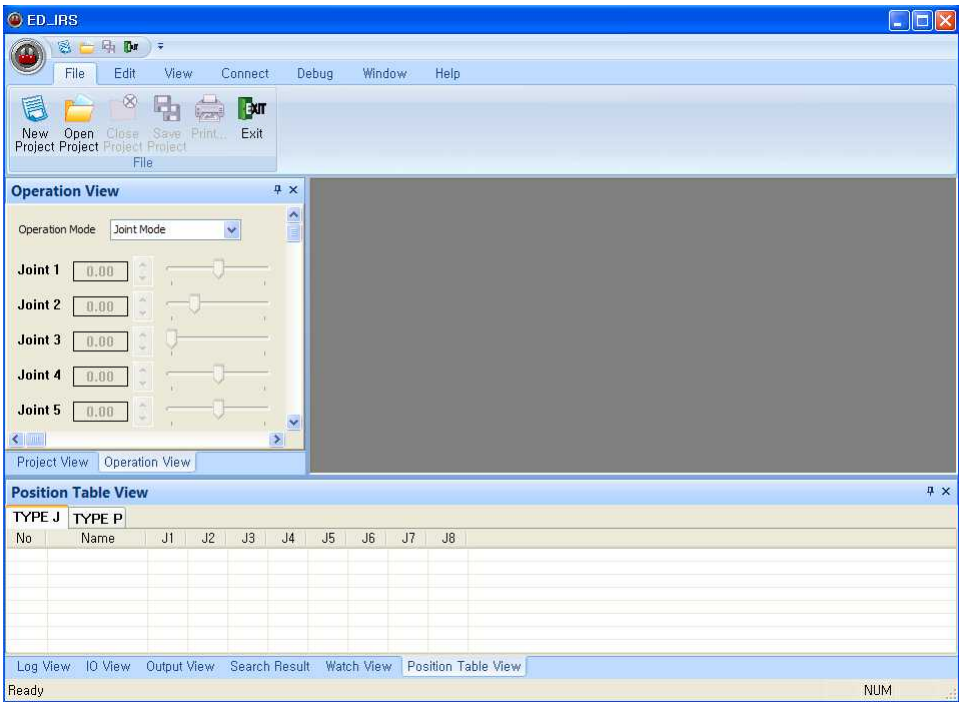
- ABS (parameter): Find the absolute value of the parameter
- MAX (parameter, parameter): Find the greater value of the two parameters
- MIN (parameter, parameter): Find the smaller value of the two parameters
- DEGRAD (parameter): Change the degree angle value parameter to radian angle value
- RADDEG (parameter): Change the radian angle value parameter to degree angle value

- SQR (parameter): Find the square value of the parameter
- SIN (parameter): Find the SINE value of the parameter
- COS (parameter): Find the COSINE value of the parameter
- TAN (parameter): Find the TANGENT value of the parameter
- ASIN (parameter): Find the ARC SINE value of the parameter
- ACOS (parameter): Find the ARC COSINE value of the parameter
- ATN (parameter): Find the ARC TANGENT value of the parameter
- ATN2 (parameter, parameter): Find the ARC TANGENT2 value of the parameter

PART 5

Host Program Example

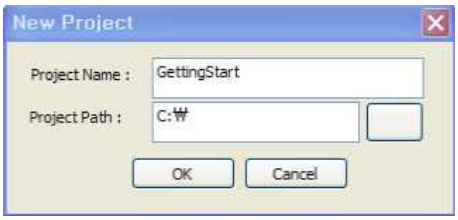
5-1. Starting

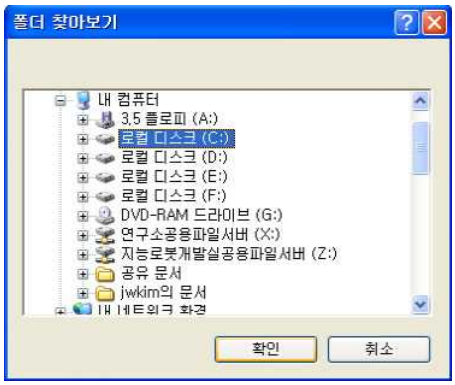


When a program is started first, ED-IRS is executed as the above drawing.

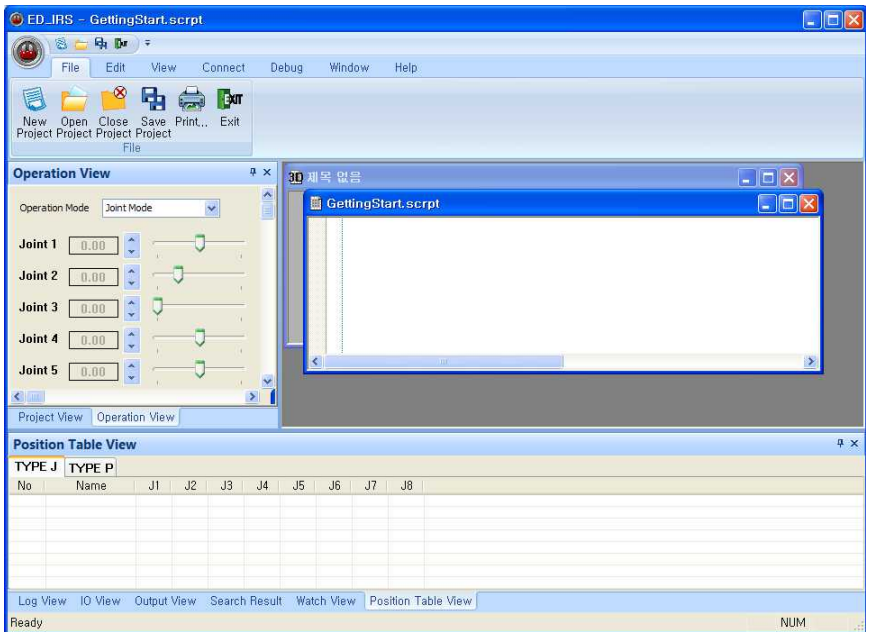
5-2. Creating Project

Click New Project( ) of File menu.



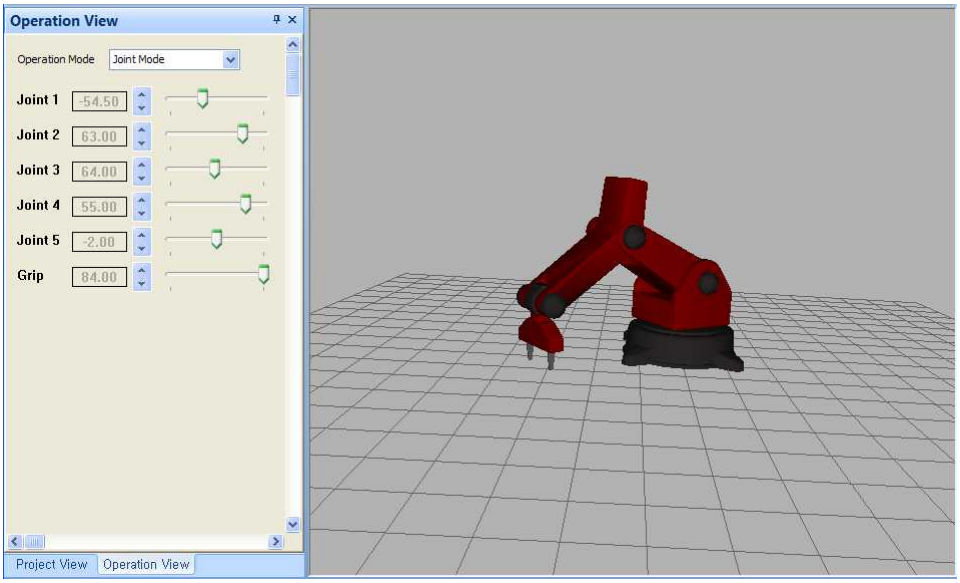


1 2 As shown in Figures above, type a project name to be created in ①. Here, we typed “GettingStart" as a project name. Press a button ② and specify in Folder Search window a project location or path where the project is to be saved. Here, we specified "C:\" as a project path. If pressing OK button, Arm View and Program View window is opened as follows, and a project is created.

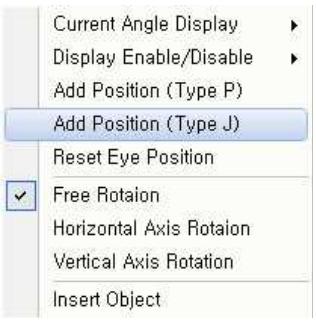


5-3. Teaching Robot

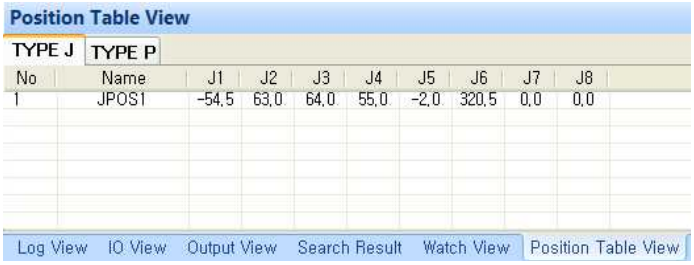
Adjust the window size such that a robot can be well expressed. Using Ctrl and Shift keys and a right and middle button of a mouse, adjust an eye position of a camera such that a robot can be well shown. Change the joint values of Operation View as shown in the following drawing such that a robot can pose as shown in the drawing.



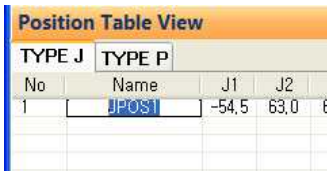
Put a mouse on a robot, and click a right button to call a popup menu.



In the popup menu, click 'Add Position (Type J)'. Then, the present joint space coordinate values of a robot is stored in 'Position Table View' as shown in the following drawing.

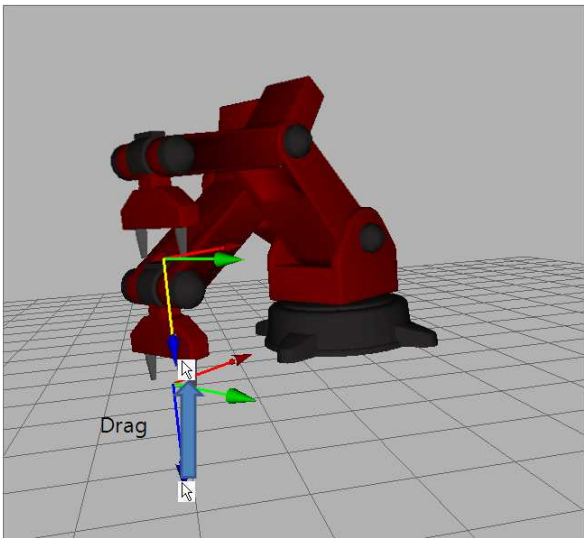


Double click a cell of "JPOS1" of "Name" in 'Position Table View'.

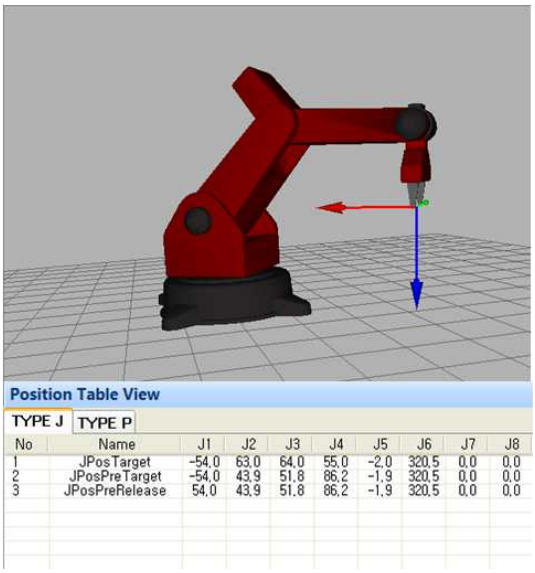


Then, the cell is changed to be edited as in the drawing. Name the position data as "JPosTarget".

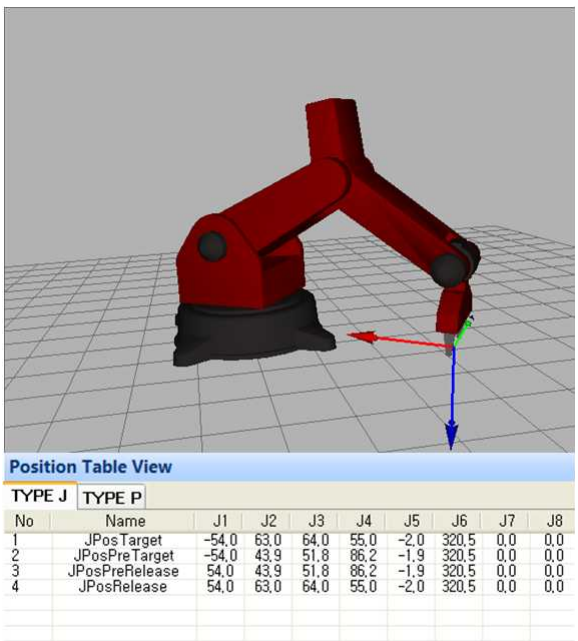
Dragging a blue colored axis (Z-axis) of an axis guide appearing when a robot is selected, move up the robot to a proper position. Save the position and name it as "JPosPreTarget".



Next, change Joint1 in “Operation View” and move the robot as in the drawing. Save the relevant position and name it as “JPosPreRelease”.



Again, dragging the blue colored axis (Z-axis) of the axis guide, move down the robot to a proper position. Save the position and name it as “JPosRelease”.



5-4. Writing Program

Let's write a program, using the position data obtained from teaching in B-3.

Click Program View( ) in View Menu, or search an open Program View window.

```
1  PROGRAM
2  END
3
```

Once writing as in the above drawing, save it by clicking Save Project( ) of File Menu or by using Ctrl+S shortcut key. If saved, the characters of program codes are expressed, as shown in the below drawing, by the color configured in Edit Menu의 Option( ), according to the syntax.

```
1  PROGRAM
2  END
3
```

※ Caution : A robot program must be started with "PROGRAM" and be ended with "END". Only when the lines are completed by pressing 'ENTER' key at the "END" statement, the error does not occur when in syntax or grammar checking.

1) Move to HOME Position

Write as follows to move a robot to a home position.

```
1  PROGRAM
2  ' Program Start
3  REM move home position
4  GOHOME
5  END
```

" ' " or "REM" explains that the corresponding line is a comment statement.

"GOHOME" command moves a robot to a home position.

2) Move to Position Able to Grip Object

Next, move a robot to the position 'JPosPreTarget' which was obtained by the teaching in B-3, and write a program for moving a robot to the standby position where it can grip an object. A command for moving a robot on the basis of absolute coordinate is "MOVETO".

```
1  PROGRAM
2  ' Program Start
3  REM move home position
4  GOHOME
5  REM move ready position to grip object
6  MOVETO JPosPreTarget
7  END
```

After writing up to now, perform grammar checking. Save the project and click Check Grammar( ) of Debug Menu. If the grammar checking is normally completed, "Checking Grammar Complete" message appears in Output View, as follows.

If the mis-typing or the error occurs, Output View displays the error message. If ENTER key was not performed after END statement, the following error message appears.

If double clicking the message, the relevant position is displayed in

Program View, as shown in the below drawing.

```
1  PROGRAM
2  ' Program Start
3  REM move home position
4  GOHOME
5  REM move ready position to grip object
6  MOVETO JPosPreTarget
7  END
```

3) Move to Object Position

Next, move a robot to the position 'JPosTarget' which was obtained by the teaching in B-3, and write as follows a program for moving a robot to the position for gripping the object.

```
1  PROGRAM
2  ' Program Start
3  REM move home position
4  GOHOME
5  REM move ready position to grip object
6  MOVETO JPosPreTarget
7  REM move target object position
8  MOVETO JPosTarget
```

4) Grip Object

Write a command for gripping the object by using a gripper of a robot. Commands for operating the gripper are "OPEN" and "CLOSE".

Move a robot to JPosPreTarget position, and fully open a gripper as a pre-gripping operation. After moving a robot to JPosTarget position, close a gripper by, for example, a command "CLOSE 50" if the object size is 30cm.

```
1  PROGRAM
2  ' Program Start
3  REM move home position
4  GOHOME
5  REM move ready position to grip object
6  MOVETO JPosPreTarget
7  REM open gripper
8  OPEN
9  REM move target object position
10 MOVETO JPosTarget
11 REM close gripper to grip object
12 CLOSE 50
13 END
```

5) Move to Object Released Position

Next, move a robot to the position where it releases the object.

As follows, move again a robot to JPosPreTarget position, and then move it to JPosPreRelease position where the robot releases the object.

```
1  PROGRAM
2  ' Program Start
3  REM move home position
4  GOHOME
5  REM move ready position to grip object
6  MOVETO JPosPreTarget
7  REM open gripper
8  OPEN
9  REM move target object position
10 MOVETO JPosTarget
11 REM close gripper to grip object
12 CLOSE 50
13 REM move ready position to grip object after grip object
14 MOVETO JPosPreTarget
15 REM move ready position to release object
16 MOVETO JPosPreRelease
17 REM move release position
18 MOVETO JPosRelease
19 END
```

6) Release Object

Now, let's have a robot release the object and move again to JPosPreRelease position. As follows, a gripper is fully opened and a robot is moved to JPosPreRelease position.

```
1  PROGRAM
2  ' Program Start
3  REM move home position
4  GOHOME
5  REM move ready position to grip object
6  MOVETO JPosPreTarget
7  REM open gripper
8  OPEN
9  REM move target object position
10 MOVETO JPosTarget
11 REM close gripper to grip object
12 CLOSE 50
13 REM move ready position to grip object after grip object
14 MOVETO JPosPreTarget
15 REM move ready position to release object
16 MOVETO JPosPreRelease
17 REM move release position
18 MOVETO JPosRelease
19 REM open gripper to release object
20 OPEN
21 REM move back read position
22 MOVETO JPosPreRelease
23 END
```

7) Repeat Operation

Use GOTO statement in writing a program, so that a robot is repeatedly operated.

As follows, add LABEL in the fourth line and write GOTO statement. Name LABEL as “L1”.

```
1  PROGRAM
2  ' Program Start
3  REM move home position
4  L1 : GOHOME
5  REM move ready position for grip object
6  MOVETO JPosPreTarget
7  REM open gripper
8  OPEN
9  REM move target object position
10 MOVETO JPosTarget
11 REM close gripper for grip object
12 CLOSE 50
13 REM move ready position for grip object after grip object
14 MOVETO JPosPreTarget
15 REM move ready position for release object
16 MOVETO JPosPreRelease
17 REM move release position
18 MOVETO JPosRelease
19 REM open gripper for release object
20 OPEN
21 REM move back read position
22 MOVETO JPosPreRelease
23 REM move process control to L1
24 GOTO L1
25 END
```

5-5. Robot Control Program Simulation

Let's have a simulation for a robot control program by using a virtual 3D robot, in order to examine whether the program works properly.

First, for grammar checking, save the project and click Check Grammar( ) of Debug Menu. Unless this process is normally completed, the simulation cannot be started.

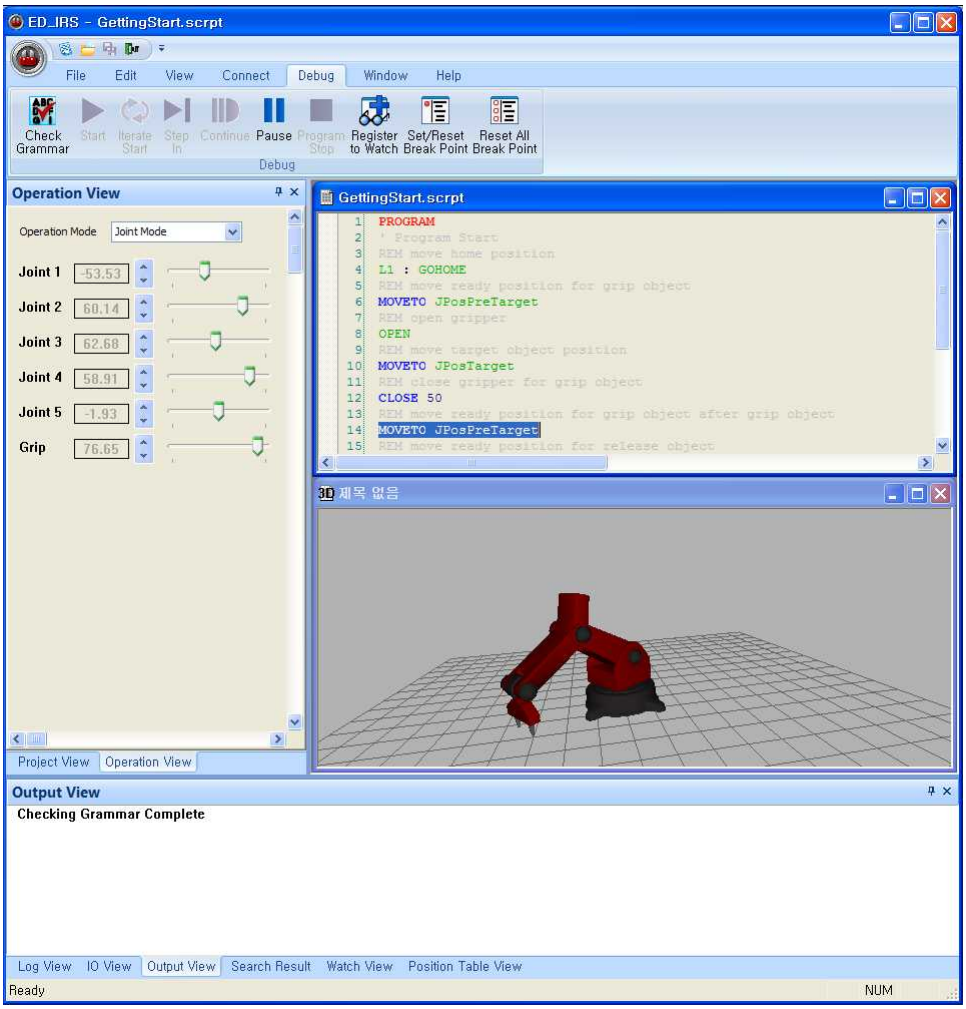
If the grammar checking is completed, Start( ) of Debug Menu is activated.

Rearrange windows in Window Menu, so that Program View and Arm

View can be well shown.

If clicking Start(▶) of Debug Menu, a virtual 3D robot operates according to the written program, as shown in the below drawing.

To stop the simulation, click Pause(❚❚) of Debug Menu, and then click Program Stop(■).



PART 6 Program Example

6-1. Program Example

PART 6

Program Example

6-1. Program Example

ED 7255 Textbook Scenario

- Study of Robot Commands -

1 .

Practice Task	Move a robot from a home position to JPOS1, by using robot commands.
Practice Purpose	Write robot commands such that a robot can move at a given target angle.

Practice Details

Using robot commands included in a simulator program, constitute robot commands for moving a robot from a home position to JPOS1 and operate the program in the simulator. Be sure to include the moving speed and absolute position.

Joint	joint 1	joint 2	joint 3	joint 4	joint 5	Gripper
Angle, cm	54.50	56.24	119.94	-85.64	0	50

Table 1 JPOS1

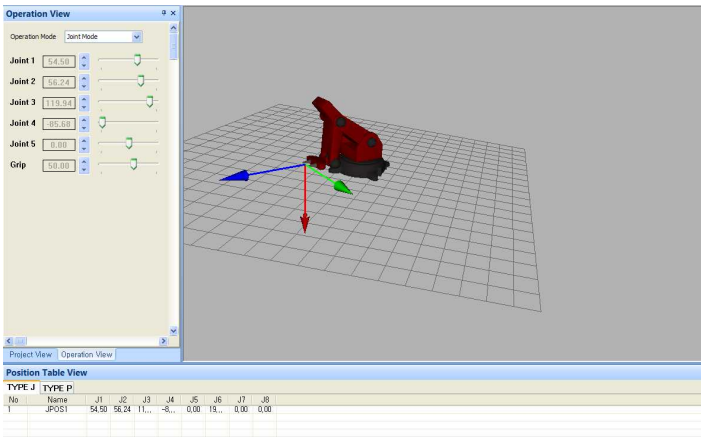


Figure 1. Teaching Image

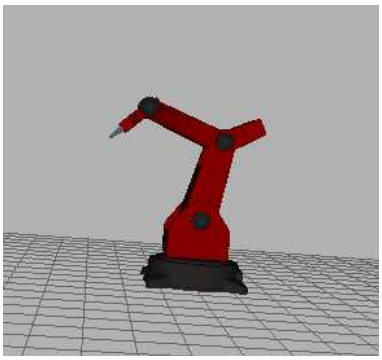


Figure 2. Software Home Coordinate

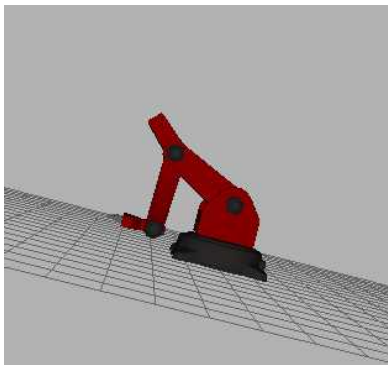


Figure 3. JPOS1 Coordinate

Writing Program	
E.g.) PROGRAM GOHOME MOVETO JPOS1, S=30 END	

Expansion

① Vary the coordinate of JPOS1.

② Vary the speed.

③ Test for a time command T=1000 so that the operation is

completed within a given arrival time.

2.

Practice Task	Continuously operate 5 teaching positions.
Practice Purpose	Have a robot perform a mission, by teaching many positions.

Practice Details

Teach arbitrary positions not more than 10 places, and operate (JPOS1 – JPOS5) in the simulator, wherein the 1st to 3rd are for absolute position moving and the fourth and fifth are for relative position moving. Enter the speed value for all cases.

E.g.)

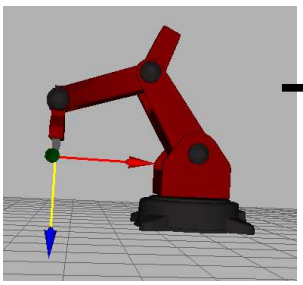


Figure 4 JPOS1

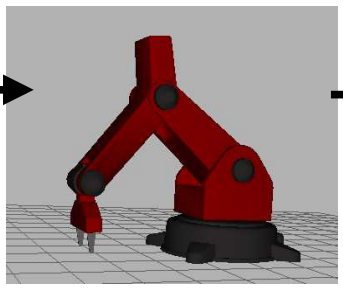


Figure 5 JPOS2

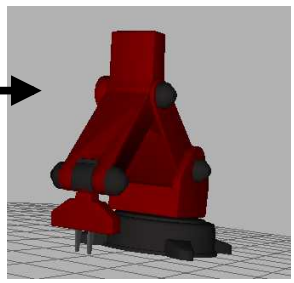


Figure 6 JPOS3

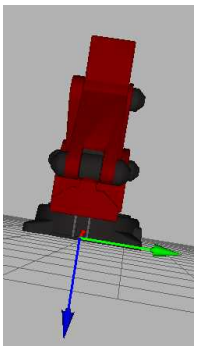


Figure 9
JPOS6

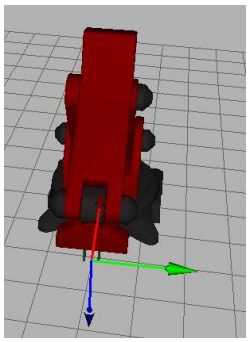


Figure 8 JPOS5

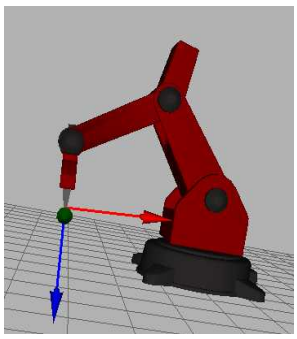


Figure 7 JPOS4

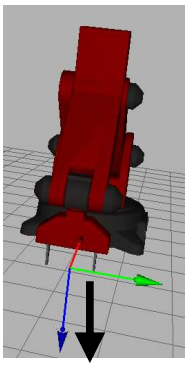


Figure10
JPOS7

Explanation of Drawings

① JPOS1: Take a basic posture of gripping a work.

② JPOS2: Move a gripper just over a work.

③ JPOS3: Grip a work.

④ JPOS4: Raise up a work (Z-axis) for easy moving to different position.

⑤ JPOS5: Take a work over the work released position by rotating only No. 1 joint.

⑥ JPOS6: Take an end-effector over the work placed position.

⑦ JPOS7: Open a gripper and safely place a work.

Writing Program	

Expansion

① Make virtual scenarios related to work moving, and write more robot commands.

② Assume the scenario for moving a work put on a wall to a floor.

③ Assume the scenario for stamping on a paper a stamp put on a floor.

3.

Practice Task	Perform a test by changing the speed variable into the time variable, in Example No. 2 above.
Practice Purpose	Understand the difference between the speed-based moving and time-based moving.

Practice Details

From Program Example No. 2, change the program so that the arrival time is entered when performing the fourth and fifth commands are executed.

Writing Program	

E.g.) PROGRAM
GOHOME
MOVETO JPOS1, S=30
MOVETO JPOS2, S=30
MOVETO JPOS3, S=30
MOVE JPOS4, T=2000
MOVE JPOS5, T=3000
END	

Tip

※ Speed = Distance / Time

1) Speed in a Cartesian space is based upon the distance that an end-effector is to go to and upon the arrival time.

2) Speed in a joint space is based upon the distance that all joints are vector-summed and upon the arrival time.

※ How to simultaneously control a plurality of axes

1) Cartesian Space: Apply automatic acceleration and deceleration for each X, Y, and Z-axis moving path relevant to a path that an end-effector moves to the target position.

2) Joint Space: (fabs(Target Angle - Current Angle) / Time) * DSP sampling time

Divide the distance, that each joint is to be moved, by time, and make the speed of each axis different to one another so that all joints can complete operation in a given time.

4.

Practice Task	Move linearly from A position to B position.
Practice Purpose	Understand the Cartesian space-based moving.

Practice Details

Constitute robot commands for moving from A to B (see the below drawings) and operate a simulator. However, the end-effector moving must be linear.

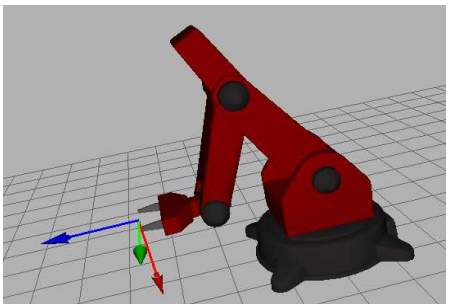


Figure A

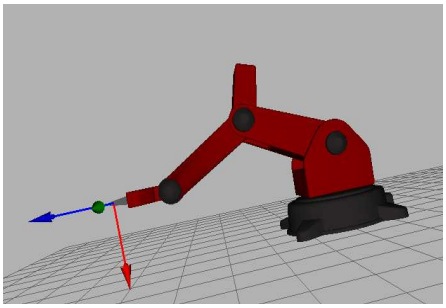


Figure B

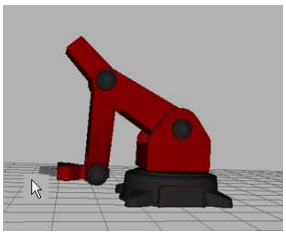
Joint	joint 1	joint 2	joint 3	joint 4	joint 5	Gripper
Angle, Cm	91.98	61.03	112.24	-84.79	0	80

Table 2 JPOS1

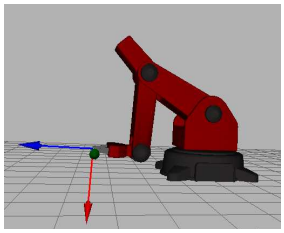
Writing Program	
E.g.) PROGRAM GOHOME MOVETO JPOS1, S=30 MOVETO L, JPOS2, S=30 END	

Tip

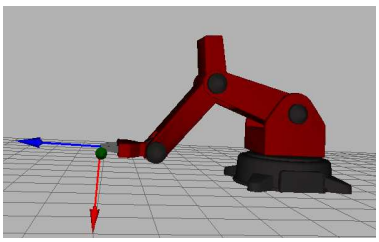
※ How to move linearly the end-effector of a robot in a simulator



Step 1



Step 2



Step 3

Step 1: Secure a work space that an end-effector can move linearly, by moving the joint. Then click a left key of a mouse on the end-effector.
Step 2: If clicking the end-effector, a coordinate arrow, that Z-axis is the direction viewing from the end-effector, is activated.
Step 3: By moving the Z-axis (blue colored) left by a mouse, the end-effector of a robot moves.

Expansion

① Perform a test by raising the end-effector vertically.
② Perform a test by forwarding the end-effector obliquely.
③ Study about kinematics.
④ Understand the difference between PtoP moving and CP moving, i.e., between a motion not in the least considering the halfway motion when moving to the target position, and a motion controlling the halfway motion.

5 .

Practice Task	Move a robot to a region where linear moving is not allowed.
Practice Purpose	Study about the result of linear moving when a robot is placed in singularity point.

Practice Details

Find that linear moving by a robot command performs the desired linear moving when what joint value is input, and does not perform the desired motion when what joint value is input.

Tip

In Figure (a) and (b), it is all impossible to move in the direction of a dark arrow.

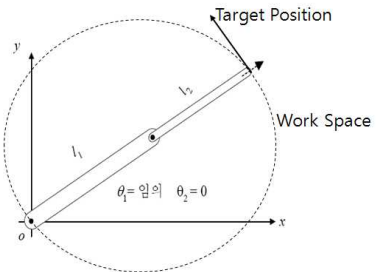


Figure (a)

Θ_1=임의: Θ_1=arbitrary

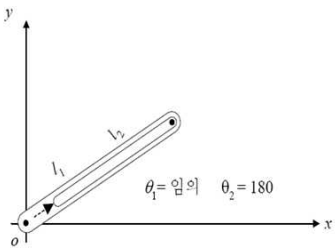


Figure (b)

Θ_1=임의: Θ_1=arbitrary

Explanation of Drawings

Figure (a): In the case that all joints are extended linearly, the linear moving is impossible in the direction excepting -z-axis. Even for -z-axis, DLS is to be taken into consideration when calculating Jacobian Transpose.

Figure (b): When the joint is retracted, the motion cannot be performed in the direction shown in the drawing.

6.

Practice Task	Teach the robot coordinates by X, Y, Z, Rx, Ry, and Rz inputs of an end effector, rather than by a joint input.
Practice Purpose	Kinematics and inverse kinematics are the operation using the relationship between a joint space and a Cartesian space. Teach the robot coordinates by using inputs based upon a Cartesian space intuitive to a human.

Practice Details

From the above Example 4, drive a simulator by changing the input value such that the input value for Step C can be calculated in a Cartesian space not in a joint space, when performing Step A to Step C.

POS1: X= 320.3, Y= 0.01, Z = 23.3 RX= 90.2 RY=0.01, RZ= -74.2

Writing Program	
E.g.) PROGRAM GOHOME MOVETO JPOS1, S=30 MOVETO L, POS1, S=30 END	

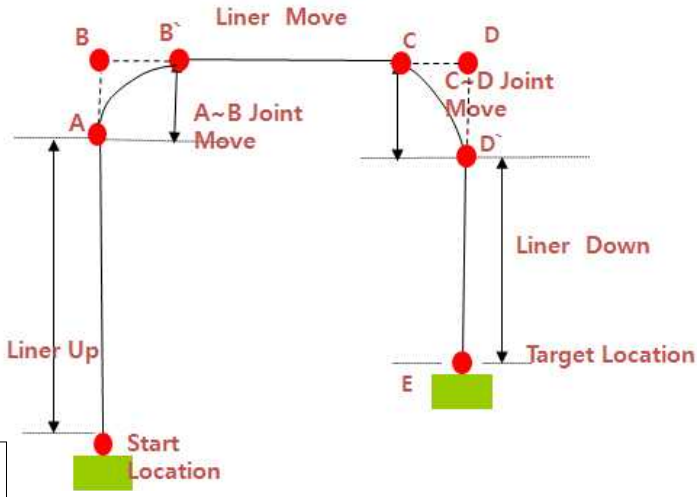
Expansion

① Using DH table, study the inverse kinematics along with the practice.

② Try to use MOVETO L and JPOS2 for linear moving.

③ Observe the difference between PtoP moving and CP moving.

Perform the following practice with regard to the drawing below.



번역자 주:
Liner Move, Liner Up, Liner Down에서 'Liner'는 'Linear'의 오자로 판단됨

Figure 11

For smooth moving, do not drop in B and C position, and do not check the inposition.

7.

Practice Task	Move a work at A position to B position only by using a joint space in reference to a given path.
Practice Purpose	Promote the understanding of an industrial robot by performing a given mission.

Practice Details

Figure 11 schematically shows the operation for moving a work at the start location to the target location. Drive a simulator so that **all the position moving must be performed based on a joint space.** However, speed and absolute/relative coordinates can be used freely.

Writing Program	

8.

Practice Task	Perform a mission by properly mixing PtoP moving and CP moving.
Practice Purpose	Experience moving a real industrial robot through the combination of linear moving and PtoP moving.

Practice Details

From the drawing, write a program and drive a simulator, so that a robot can linearly move during a start location to A position, B to C position, and D to E position, and can move on a joint space basis during A to B and C to D position.
However, speed and absolute/relative coordinates can be used freely.

- From now on, confirm performance of a given mission through a real robot (ED-7255).

9.

Practice Task	For a given mission, drive ED-7255 by using a teaching pendant.
Practice Purpose	Operate a real robot based upon the operation confirmed from the simulator.

Practice Details

Complete the operation of Figure 11 by using a teaching mode of a teaching pendant. However, speed and absolute/relative coordinates can be used freely.

Writing Program	

Tip

① Before writing commands, study how to use a teaching pendant in advance.

② Write commands of the pendant, based on the operation confirmed from the simulator.

Matters requiring safety and attention	1. When executing the robot commands written, perform it step by step through a step mode. 2. If the error or unexpected operation occurs during operation, stop a robot by using a stop button or an emergency button. 3. A stop button is used for an unexpected operation, while an emergency button, which is located in a main controller, is used for reckless run due to the system error or any emergency situation. 4. At least two persons are required to attend the practice. One person other than an operator should confirm the operation, getting ready to press the emergency button.

10.

Practice Task	Upload to a main controller a robot command written in a simulator, and execute it through the pendant.
Practice Purpose	Study the organic connection of a simulator and a robot.

Practice Details

Download to ED-7255 the robot command written in Example 8, confirm the operation through a real robot.

Operation Procedure

① Connect a cross cable to a main controller and PC, and power on.
② In a simulator, open a project to be uploaded and connect it.
③ Press an upload button to upload to a main controller.
④ Open the uploaded project and drive a robot, using the pendant.
⑤ If the unexpected operation is performed, correct the command in a simulator and confirm the operation through a 3D model. Thereafter upload it again to the main controller.

11.

Practice Task	As an extension of Example 8, drive ED-7255 by using digital I/O, waiting function, etc.
Practice Purpose	Study about connection to external devices.

Practice Details

Add the followings to the robot command written in Example 8.

- After arriving at A position, wait until the digital input signal of No. 1 goes 'Hi호' and, if the input signal is received, continue the next operation.
- After arriving at D position, wait for 2 seconds and perform the next operation.

Writing Program	
E.g.) PROGRAM GOHOME MOVETO STARTPOS,S=30 OPEN 30 MOVETO JPOS1, S=30 WAIT INSIG ON D1	MOVETO JPOS2, S=30 MOVETO JPOS3, S=30 MOVETO JPOS4, T=2000 DELAY 2000 MOVETO JPOS5, T=3000 OPEN 80 END

- Study of Teaching Pendant -

12.

Practice Task	Manually operate a robot by using a jog of a teaching pendant.
Practice Purpose	Study about the jog moving used for practical teaching in industrial robots.

Practice Details

Perform each joint's moving and X, Y, and Z moving of a jog, by using a jog mode of a teaching pendant.

13.

Practice Task	Write a robot command by using a jog.
Practice Purpose	Using the pendant, write a robot command based upon the position received from a jog.

Practice Details

Using a teaching editor mode, constitute robot commands by teaching each position using a jog mode, and operate the robot.

-Coupled operation with automatic robot control (FMS)-

14.

Practice Task	Using FMS model that a robot is added, safely place 6 works all.
Practice Purpose	Experience an industrial robot through a robot FMS system, in which knowledge about industrial robots, robot commands, etc. can be tested.

Practice Details

Grip a work, that physical distribution inspection has been completed through a robot FMS system, and safely place it at the designated positions A, B, C, D, E, and F.

Operation Procedure

① Wait the input signal at jpos1 where a work can be gripped.
② If a work passing a test is arrived, the signal No. 1 is provided from FMS.
③ Grip the work.
④ Move the gripped work to 7cm in Z-axis above A position.

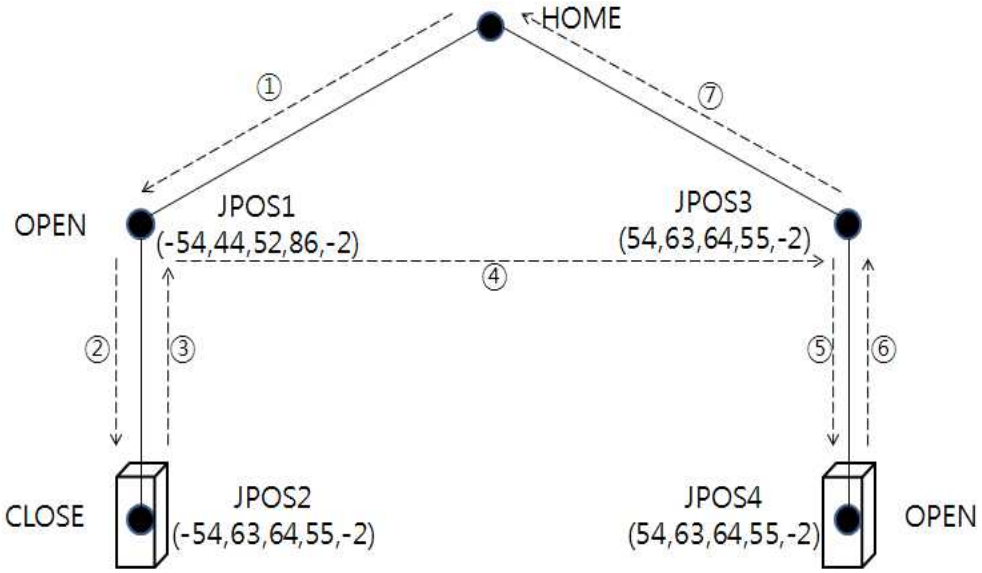
⑤ Place the work at A position through the linear moving.
⑥ Open the gripper.
⑦ Send the output signal OUTSIG OFF DO2 and OUTSIG ON DO2 to enable a test of the next work.
⑧ Locate a robot at jpos1 and wait the next work.
⑨ Repeat the procedures from No. 1 up to F position.

Writing Program	

Example

This is a program example to move a work located at JPosTarget to JPosRelease. Names and moving sequence for each position are as follows.

HOME → JPOS1 → OPEN → JPOS2 → CLOSE → JPOS1 → JPOS3 → JPOS4 → OPEN → JPOS3 → HOME



PROGRAM	: Program start
X1 : GOHOME	: Return to Home & Set a label X1
MOVETO JPOS1	: Move to JPOS1
OPEN	: Open a gripper
MOVETO JPOS2	: Move to JPOS2
OPEN 15	: Grip an object
MOVETO JPOS1	: Return to JPOS1
MOVETO JPOS3	: Move to JPOS3
MOVETO JPOS4	: Move to JPOS4
OPEN	: Place an object
MOVETO JPOS3	: Return to JPOS3
GOTO X1	: Move control to X1
END	: End of Program

PART 7
Installation of Robot Control Library

7-1. VMware

7-1-1. Installation of VMware

7-1-2. Starting VMware Program

ED Corporation

PART 7

Linux Installation

7-1. VMware

VMware is a powerful virtual machine software on a desktop. Using VMware, a virtual computer having components, such as a memory, hard disk, CD-ROM, and network, which are identical to those of a common PC components, can be realized. Through this virtual computer, various operations including formatting, bios control, OS installation, booting, software installation, etc. can be performed.

In other words, perfect networking can be supported on a single PC, and a plurality of operating systems including Microsoft Windows, Linux, Novell Net Ware, etc. can be simultaneously used through the virtual machine of portability.

You can get, and use for only 30 days, this virtual machine software through Internet at no cost, or buy it and get a license.

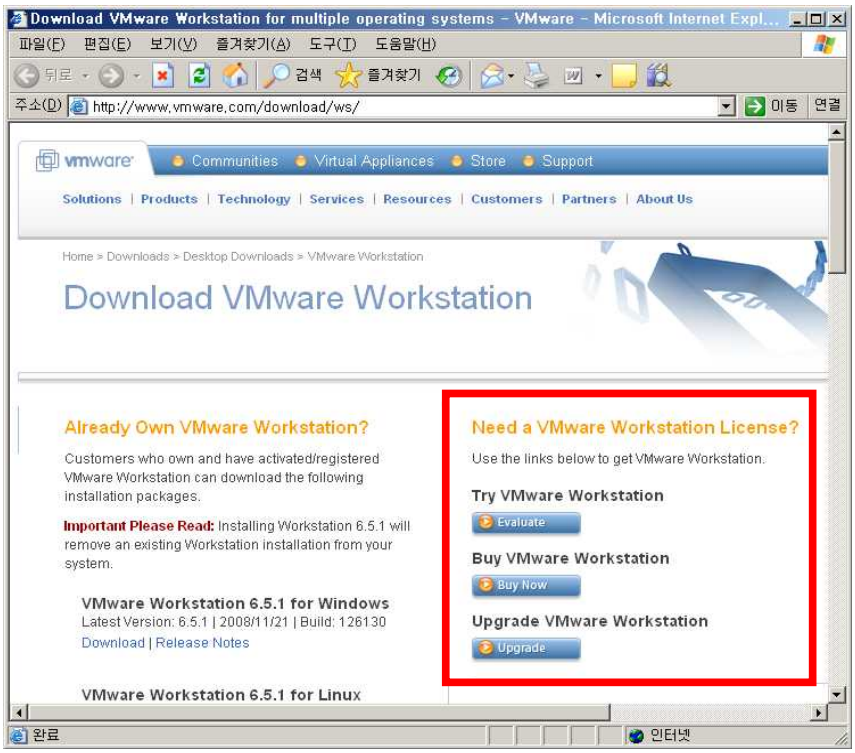


Figure7-1. VMware Homepage (http://www.vmware.com/download/ws/)

7-1-1. VMware Installation

① Download or buy a VMware-workstation installation file and get a license. Execute an installation file to start installation.

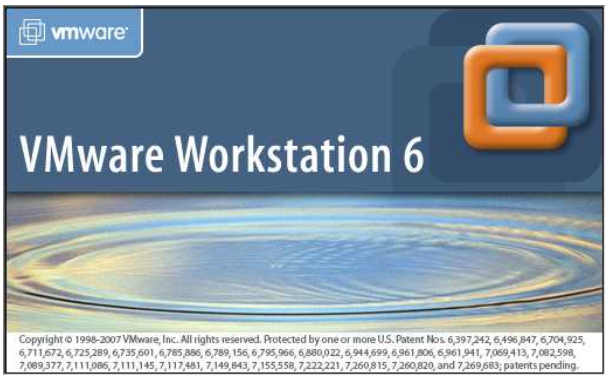


Figure7-2. Starting Screen for Installation

② This is a confirmation message for installation of VMware. Press Next button.

Figure7-3. Confirmation for Installation Progress or Not

③ Setup Type selection screen. Select Typical and press Next.

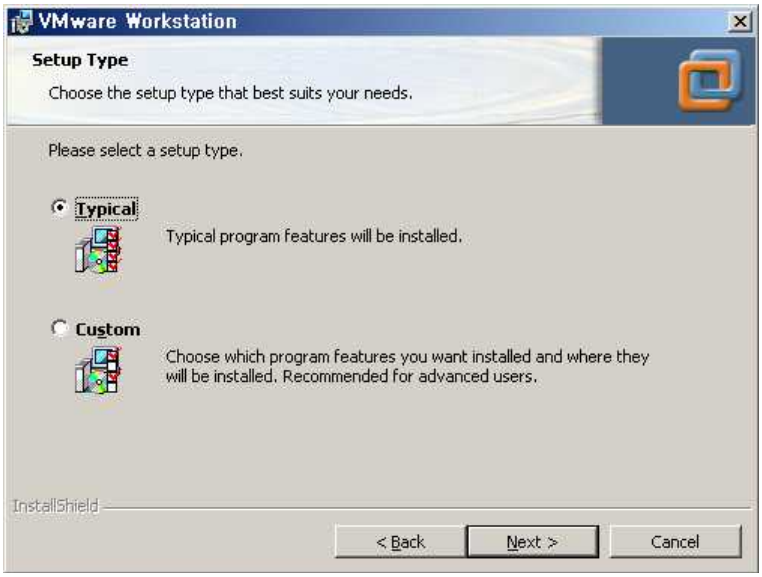


Figure7-4. Setup Type Selection (Set Typical)

④ Set an installation directory path. Press Next button without any change.

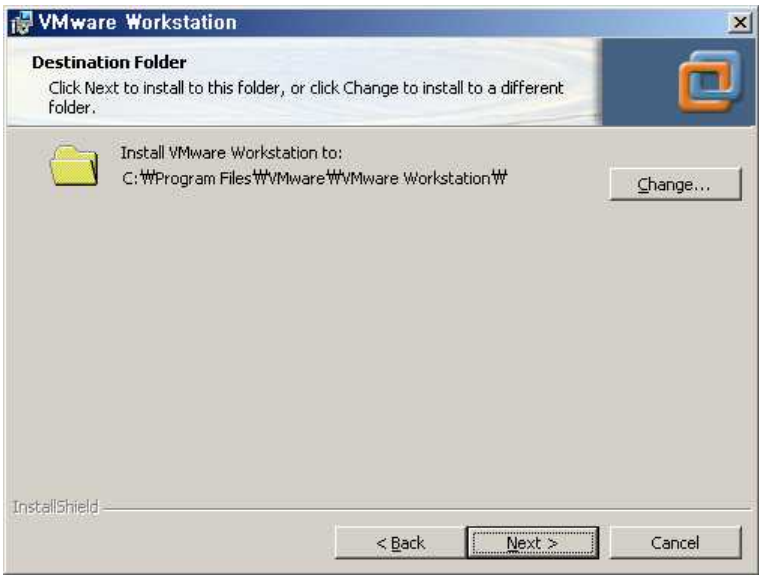


Figure7-5. Installation Path

⑤ Create a program shortcut icon. Check the boxes to make a desktop icon and program shortcut icon in Start Menu.

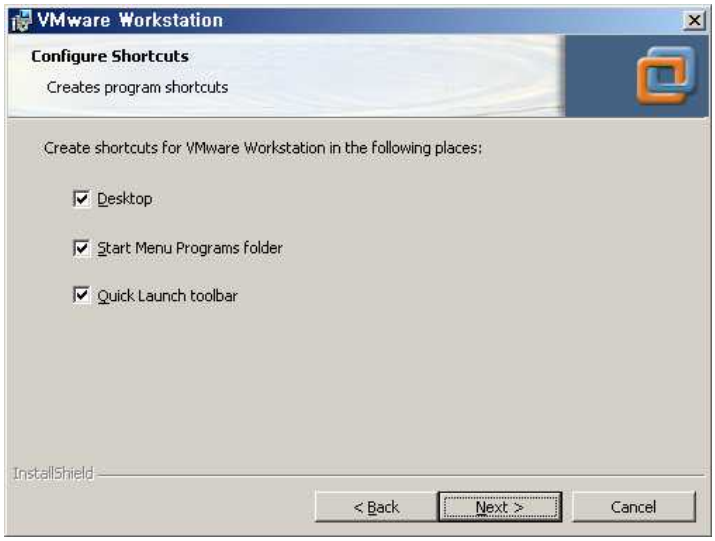


Figure7-6. Creating Start Menu

⑥ If the environment has been all set up, press Install button to start the program installation.

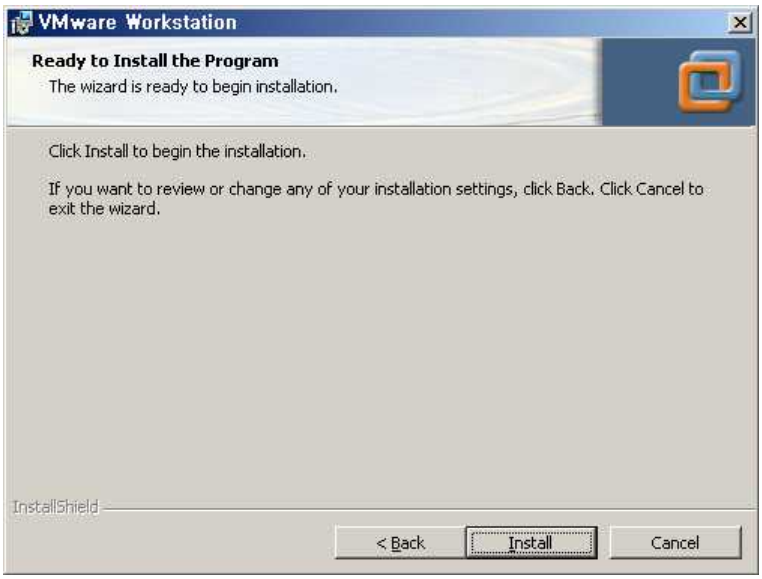


Figure7-7. Creating Start Menu

⑦ Program installation starts.

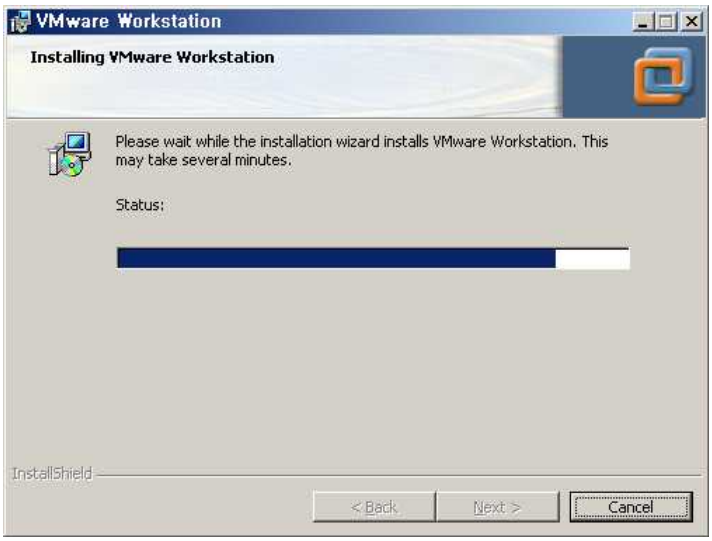


Figure7-8. Installing Program

⑧ If program installation is completed, write a license and press Enter button.

⑨ Press Finish button to finish the installation.

7-1-2. Starting VMware Program

Double click VMware( ) on a wallpaper to execute VMware Workstation. If the following messaging screen asks the license agreement, click 'Yes, ...' radio button and then OK button.

Figure7-9. License Agreement

Main screen of VMware Workstation. Uncheck "Show tips startup" checkbox, and press Close button.

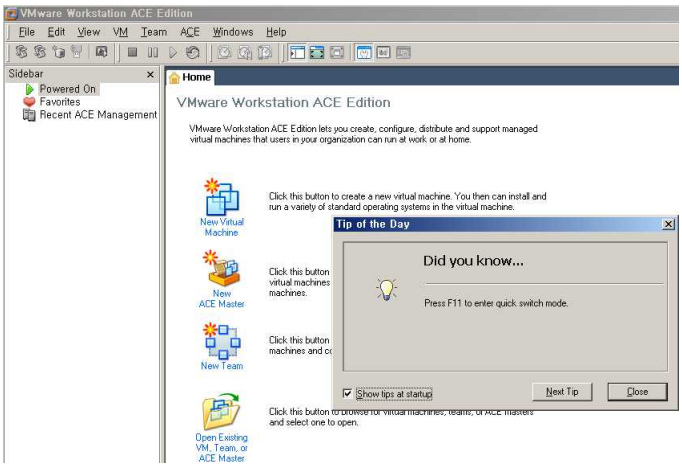


Figure7-10. VMware Workstation

New Virtual Machine provides a function to install a different OS by using a virtual machine.

To install Linux in Windows environment of the current PC, VMware will be first installed to make a virtual machine and then Linux will be installed.

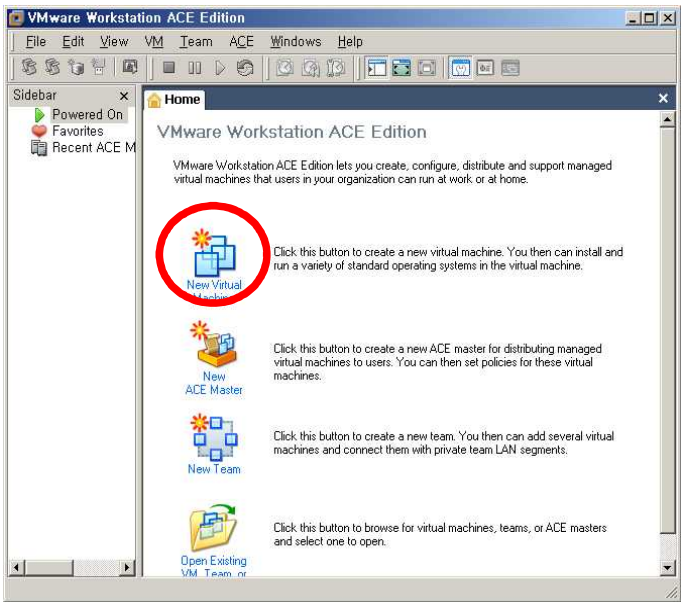


Figure7-11. VMware Workstation

If clicking New Virtual Machine icon, a virtual machine creating screen appears. Press Next button.

Figure7-12. New Virtual Machine

Setup screen for the environment of a virtual machine. For standard installation, select Typical and then Next.

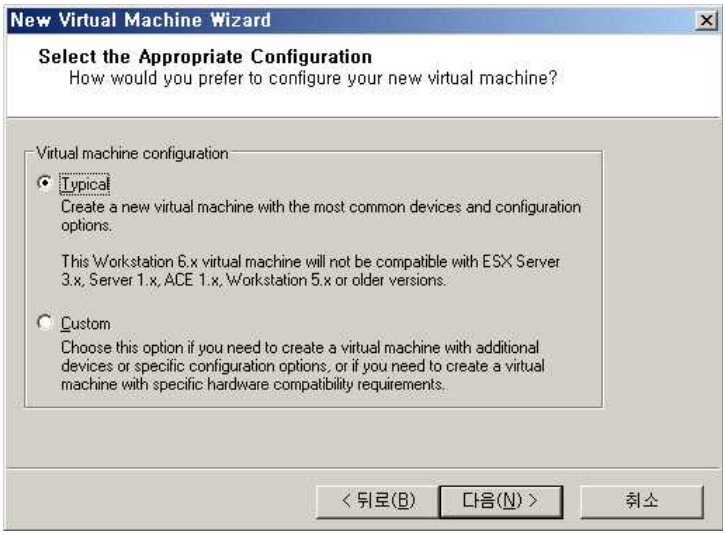


Figure7-13. Typical Installation of Virtual Machine

OS selection screen. Select Linux radio button and then Next.

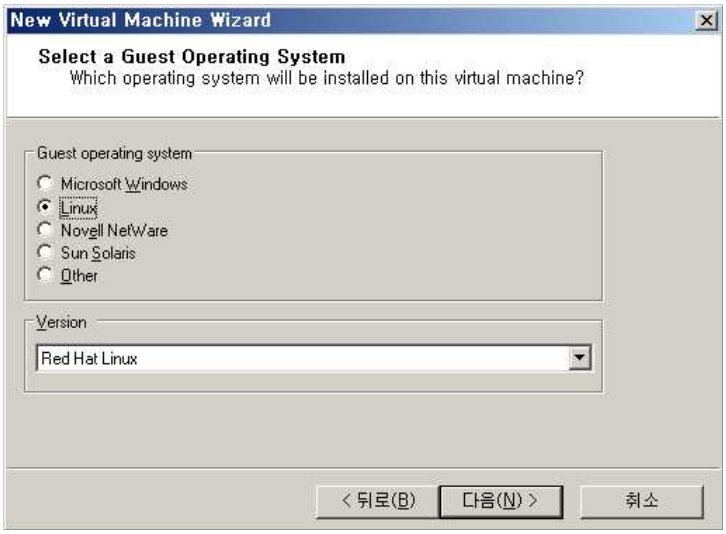


Figure7-14. Selection of OS

Specify a virtual machine name of OS to be installed in a virtual machine and OS installation path.

Figure7-15. Virtual Machine Name

Select “Use bridged networking" radio button and then Next. "Use bridged networking" can connect a host PC and other PC with Internet, by primarily using IP address of a network band identical to the host PC.

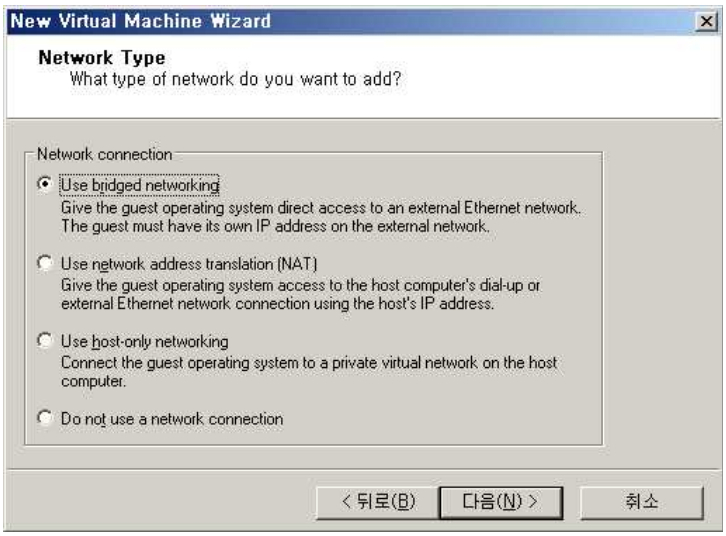


Figure7-16. Selection of Network Type

Specify the disk capacity to be assigned to a virtual machine. 10GByte is commonly used.

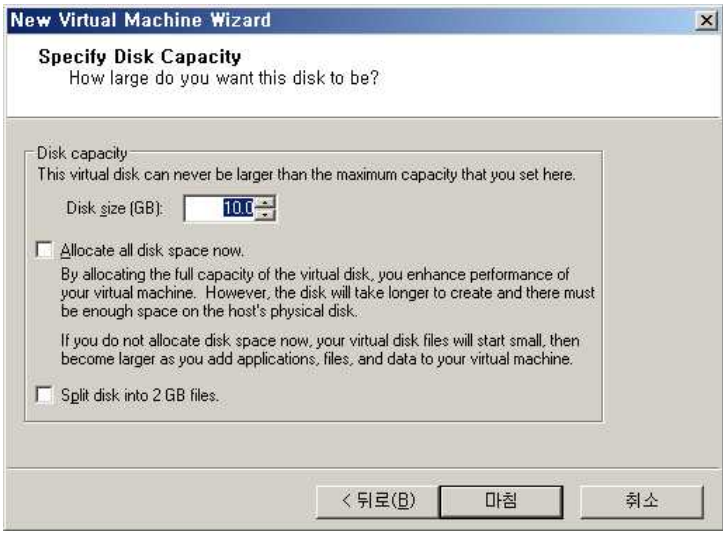


Figure7-17. Setting HDD Capacity (for 10GByte)

The environment for Linux installation has been completely set up. Uncheck "Do not show this page again" checkbox and press Close button.

Figure7-18. New Virtual Machine Created

From VMware main screen, it can be seen that a virtual machine tab named as Fedora has been created.

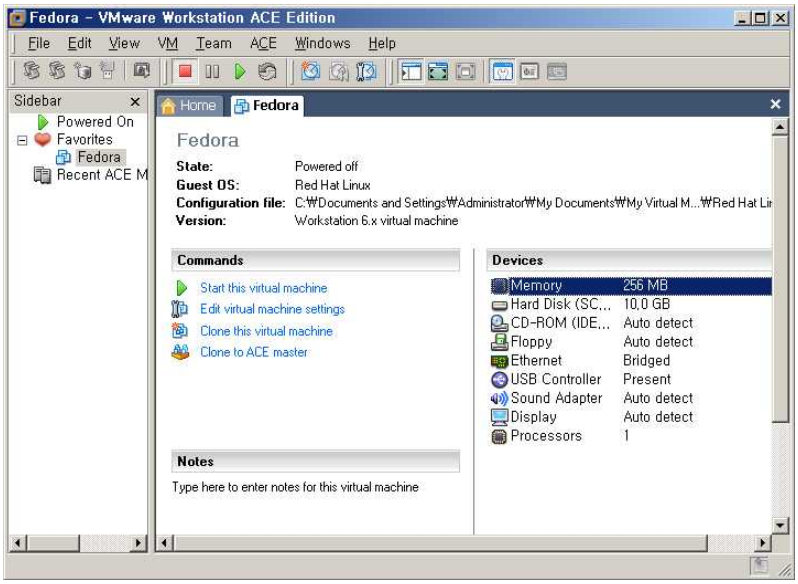


Figure7-19. New Virtual Machine Created

PART 8
Starting Embedded Linux Control

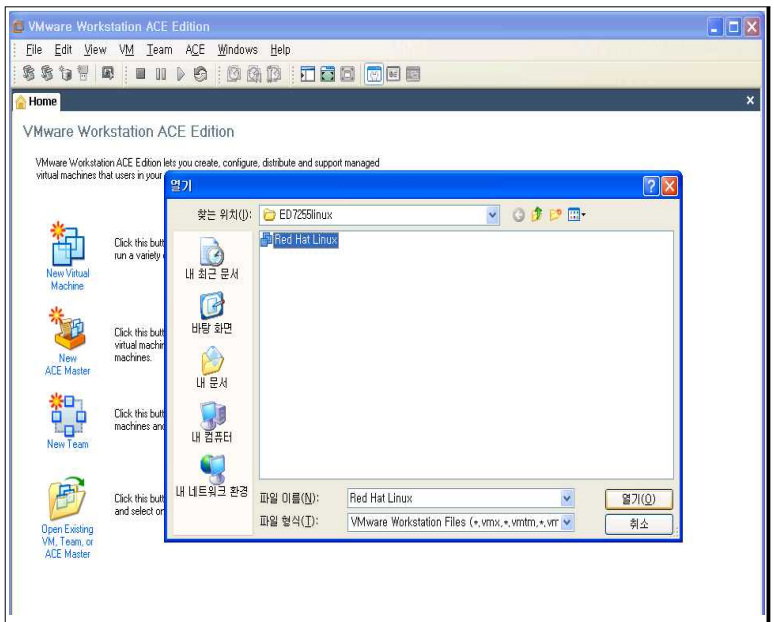
ED Corporation

PART 8

Use of ED-7255 ARM Library

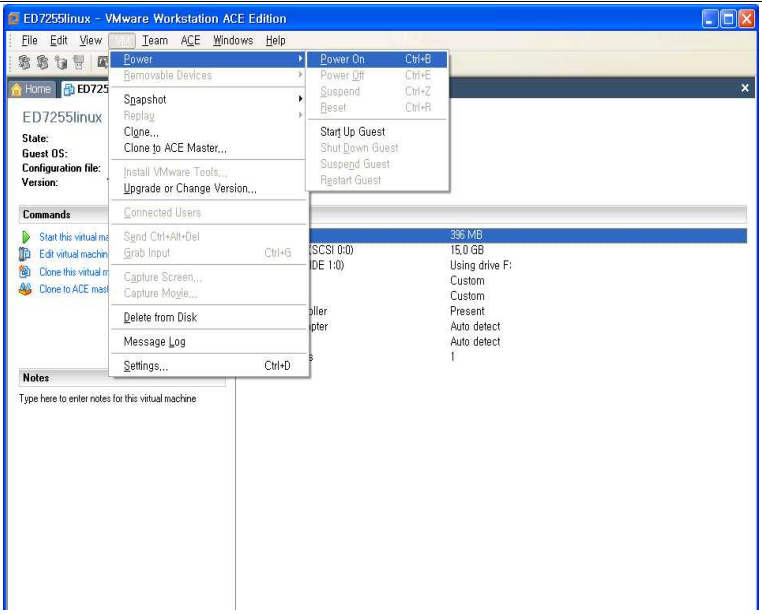
※ ARM image used in PART 8 stands for ED-7255 Linux ARM Image provided by ED Corp. A user wishing to get it can inquire the pertinent business office.

1. Directly install Linux in VMware state, or call an image. Since the direct installation is in detail described in ED-7255, it will be omitted. Here, we will describe installation by calling the image.
   - Click 'Open' in 'File' tab of VMware top menu to call the following image file.



<Linux installation image to be called>

2. Click as follows Top Menu 'VM' → 'Power' → 'Power On', and then Linux booting is performed.



<Execute Linux image>

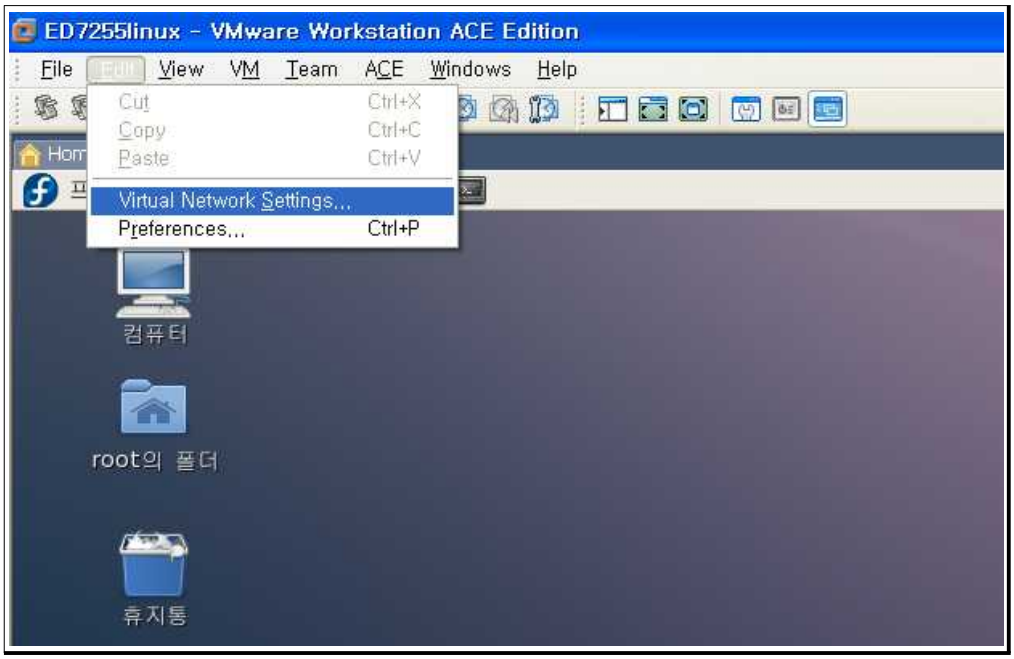
Link Setting of ED-7255

※ VMware network setting method is described. This is a PC environment in which two LAN cards are installed, one of which is a card for communication with the equipment, the other of which is a card that Linux and Windows share files.

1. ID and password of Fedora Linux provided by ED Corp. are as follows.

ID	root
Password	edrobot

2. To set up and confirm the status of a network currently connected with ED equipment, perform as follows.
   - VMware Window → Edit → Virtual Network Setting



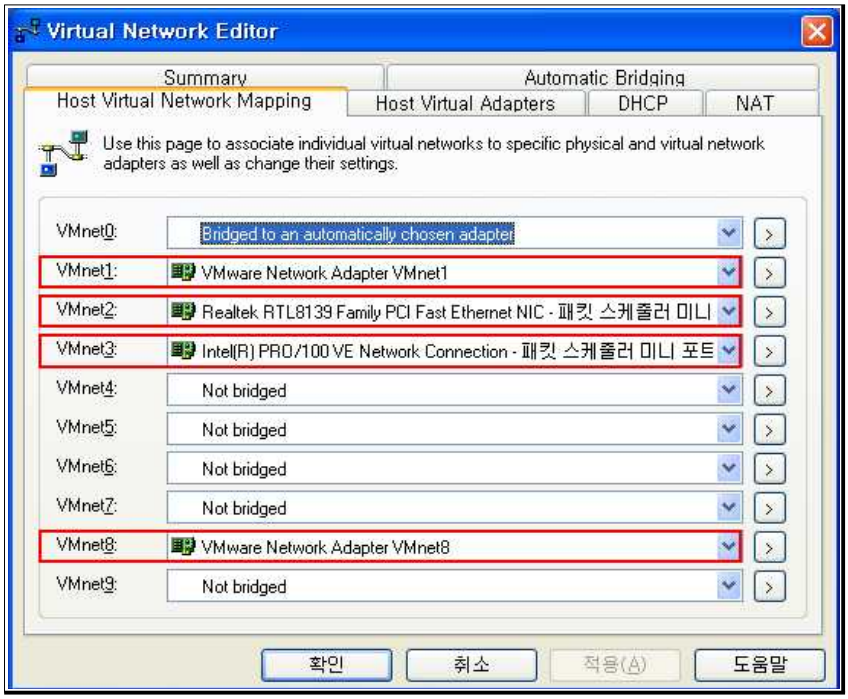
3. In Virtual Network Setting, click 'Host Virtual Network Mapping' tab to check VMnet1.2.3.8. VMnet No. 1 and 8 are created automatically. Since

VMnet No. 2 and 3 are used for the following purposes, a user is required to personally set up them.

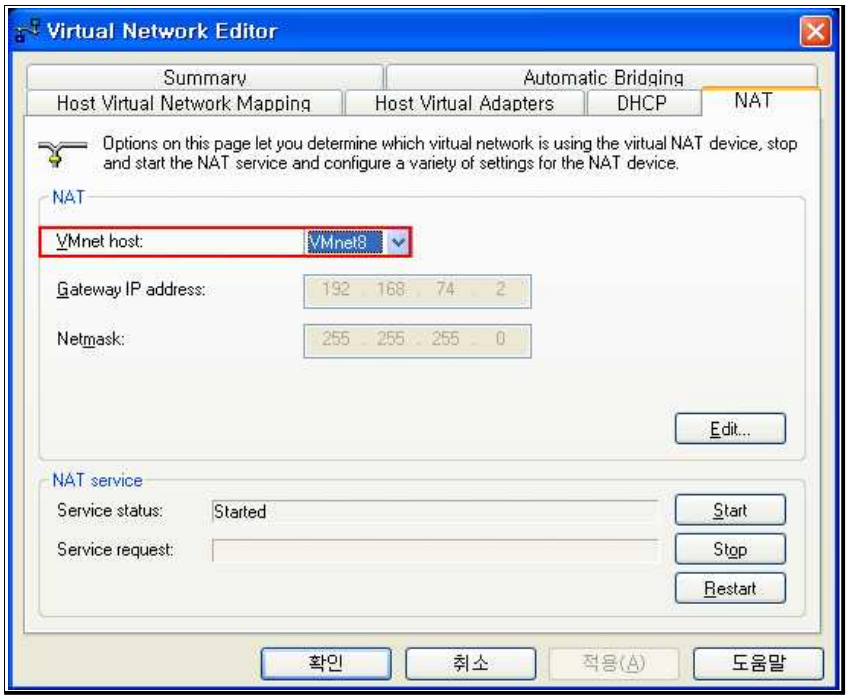
VMnet 2	LAN connected with the equipment (ED-7255)
VMnet 3	LAN connected with the outside (Internet)

※ Tip for LAN Setting

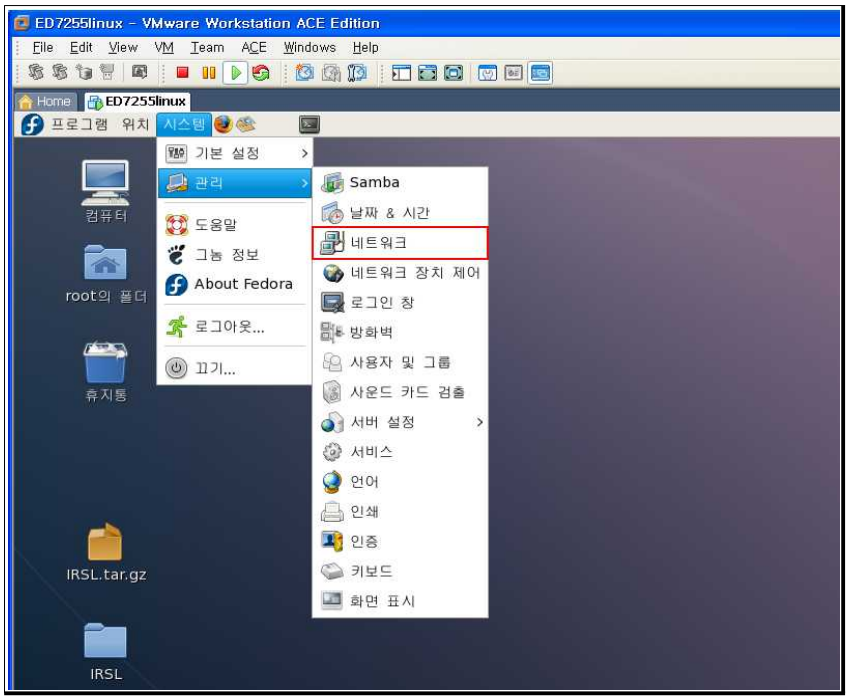
- It may be difficult to distinguish the equipment connected LAN from the outside connected LAN. It is convenient to distinguish LAN cables by confirming that a release mark is shown at the bottom of a window, when connecting and again disconnecting the LAN cable.



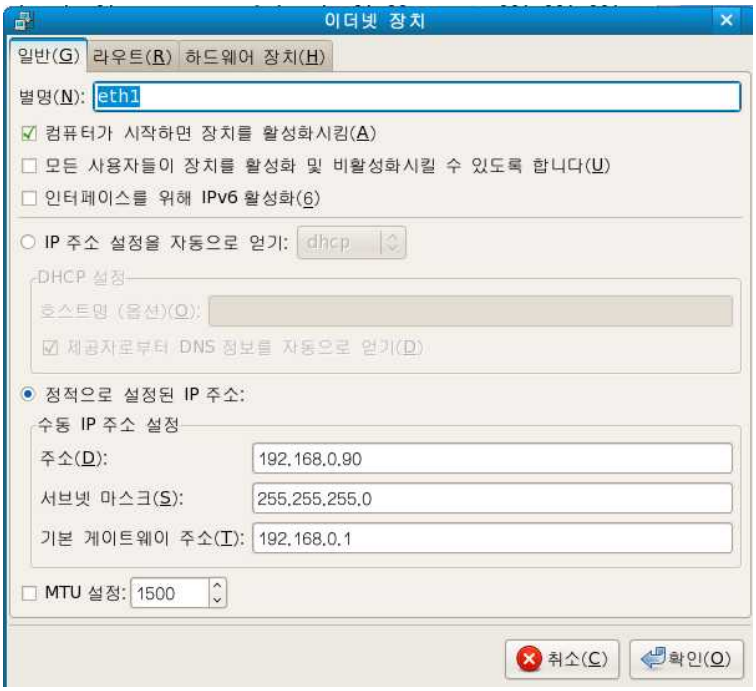
4. Click 'NAT' table in Virtual Network Setting, and confirm that VMnet host is set to 'VMnet8'.


Virtual Network Editor
Summary
Automatic Bridging
Host Virtual Network Mapping
Host Virtual Adapters
DHCP
NAT
Options on this page let you determine which virtual network is using the virtual NAT device, stop and start the NAT service and configure a variety of settings for the NAT device.
NAT
VMnet host:
VMnet8
Gateway IP address:
192 . 168 . 74 . 2
Netmask:
255 . 255 . 255 . 0
Edit...
NAT service
Service status:
Started
Service request:
Start
Stop
Restart
확인
취소
적용(A)
도움말

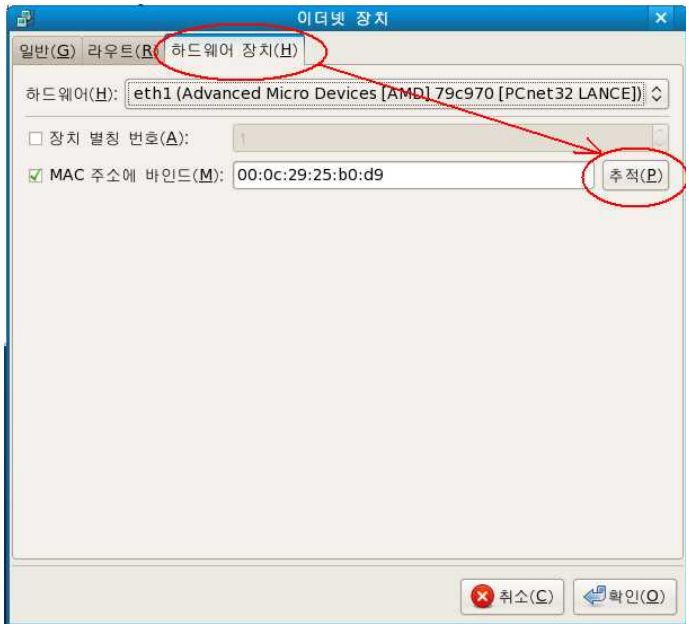
5. If clicking System → Manage → Network in the top menu of Fedora, a network setup window is created. Set as follows in 'Device' tab.



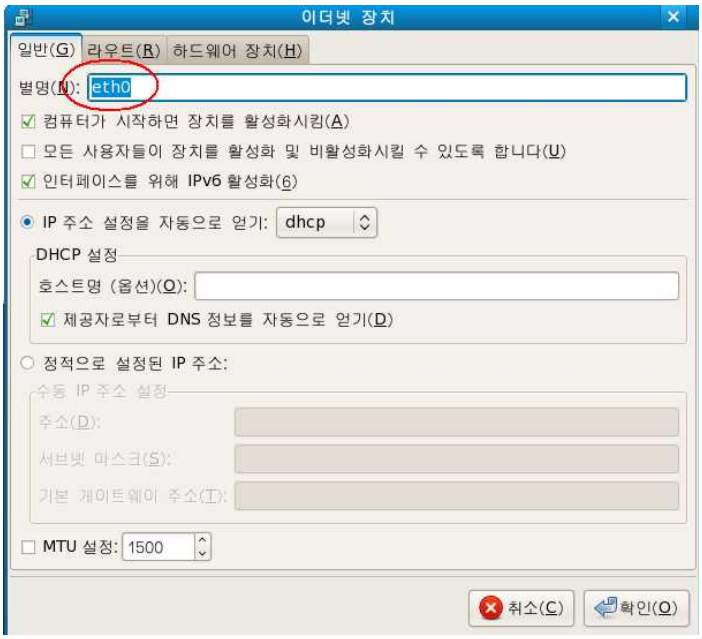
If double clicking eth1, the following window appears.



If clicking 'Pursuit' of MAC address, the network setting is completed.

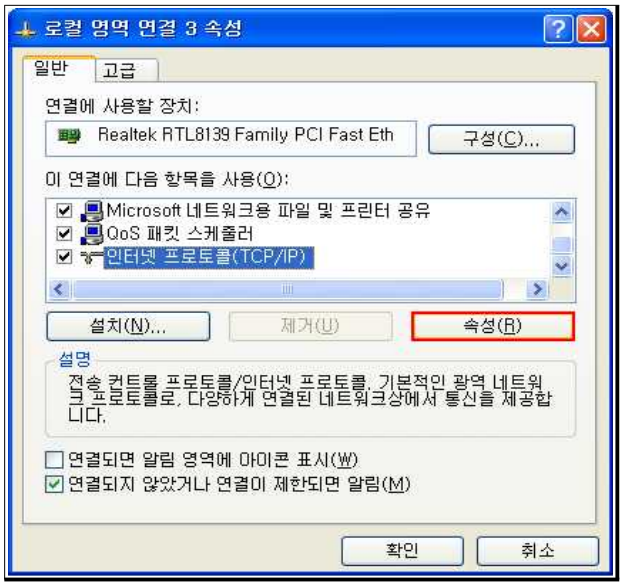


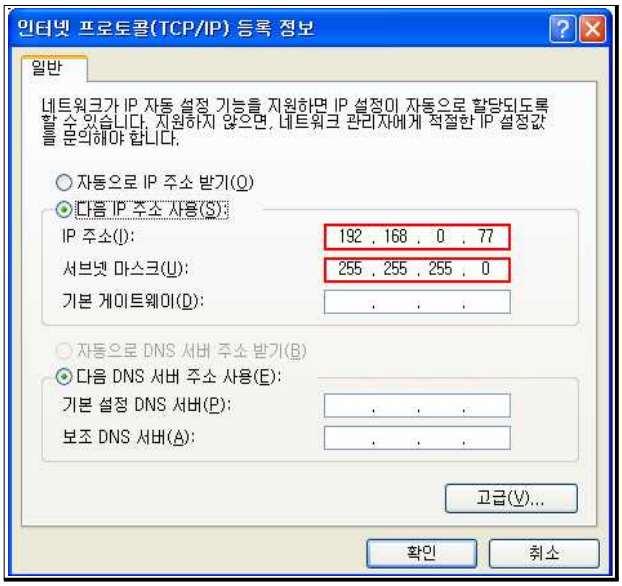
'eth0' can be also processed by the above procedure.



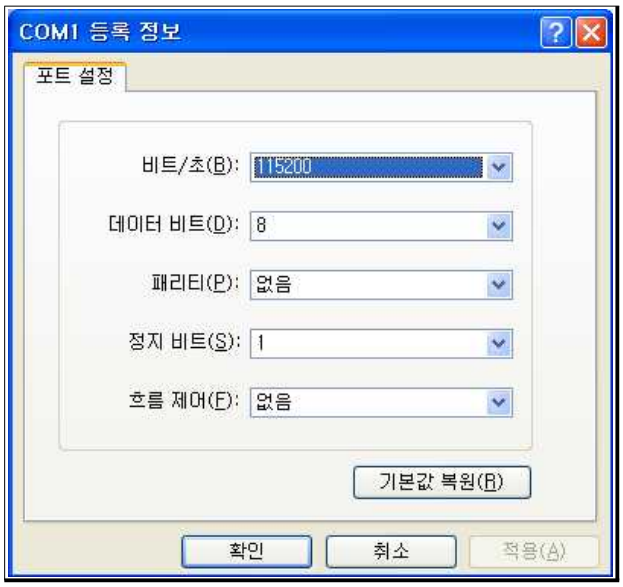
6. When the step 5 is completed, the setting is saved, and 'network setting' window is closed, a system asks system rebooting. Reboot the system.

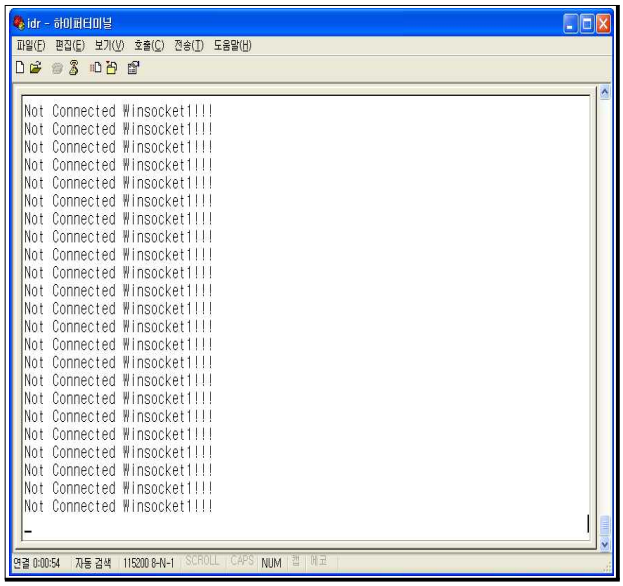
7. It is necessary to change the setting related to ED-7255 in 'My Network Environment'. If clicking Attributes button in 'My Network Environment', the following window is created. Click 'Internet Protocol (TCP/IP)', select 'Attributes', and enter the followings in 'Internet Protocol (TCP/IP) Registration Information' window.





8. To confirm whether or not the connection with ED-7255 is successful, create a hyperterminal window. The connection protocol is as follows. If the following message continues in the hyperterminal when powering on a main controller of ED-7255, connection with the ED-7255 was well performed.

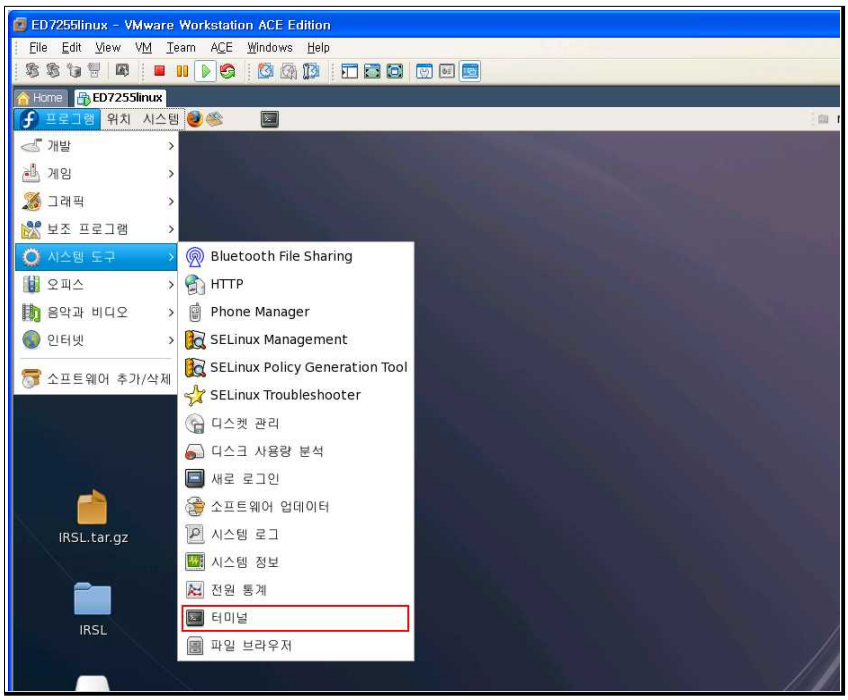



idr - 하이퍼터미널
파일(F) 편집(E) 보기(V) 호출(C) 전송(T) 도움말(H)
Not Connected Winsocket1!!!
Not Connected Winsocket1!!!
Not Connected Winsocket1!!!
Not Connected Winsocket1!!!
Not Connected Winsocket1!!!
Not Connected Winsocket1!!!
Not Connected Winsocket1!!!
Not Connected Winsocket1!!!
Not Connected Winsocket1!!!
Not Connected Winsocket1!!!
Not Connected Winsocket1!!!
Not Connected Winsocket1!!!
Not Connected Winsocket1!!!
Not Connected Winsocket1!!!
Not Connected Winsocket1!!!
Not Connected Winsocket1!!!
Not Connected Winsocket1!!!
Not Connected Winsocket1!!!
Not Connected Winsocket1!!!
Not Connected Winsocket1!!!
Not Connected Winsocket1!!!
Not Connected Winsocket1!!!
연결 0:00:54 자동 검색 115200 8-N-1 SCROLL CAPS NUM 캡 메모

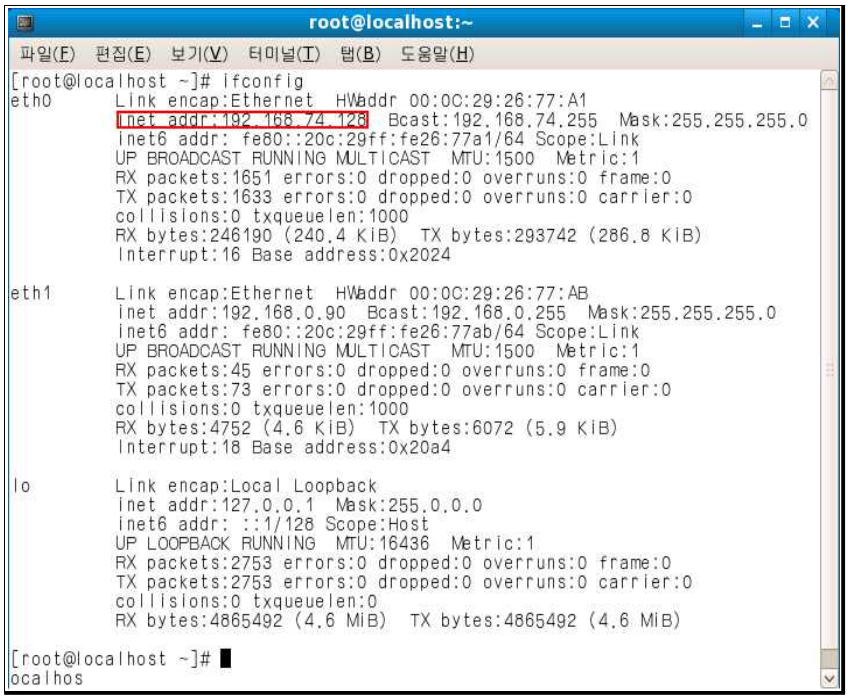
Windows & Linux Shared Folder

※ Linux and Windows are different OS and so File System is also different. To also edit the source in Windows which supports many development environment, file sharing should be used between Linux and Windows operated in WMware.

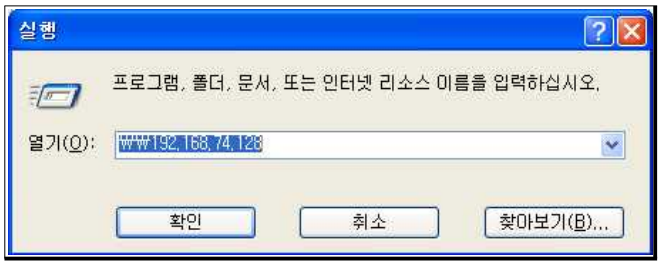
1. In VMware, click Program → System Tool → Terminal.



a. To confirm the current IP of Linux, type 'ifconfig in a prompt window of Fedora terminal. The following screen will appear. Confirm 'inet addr : 192.168.74.128'.

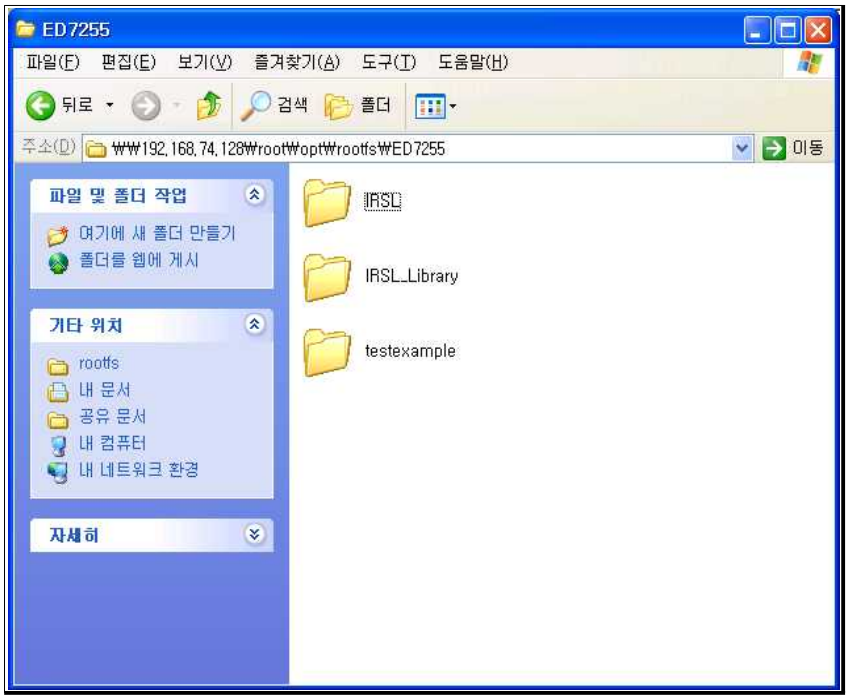


b. Activate 'Execute' window of Start Menu, and enter the confirmed IP. Then a folder shared to Linux will be opened.



2. Enter the open shared folder using the following address, the following 3 folders exist there.

Address	\\192.168.74.128\root\opt\rootfs\ED-7255



※ Tip

- IRSL_Library Folder : A folder in which driving algorithms exist. (Simple example file : Library_main.cpp[1])

1) Separately attached

Setting NFS (Network File System) Mode (Cross Compile Mode)

1. In VMware, a driving mode for ED-7255 can be directly set. There are two driving modes.

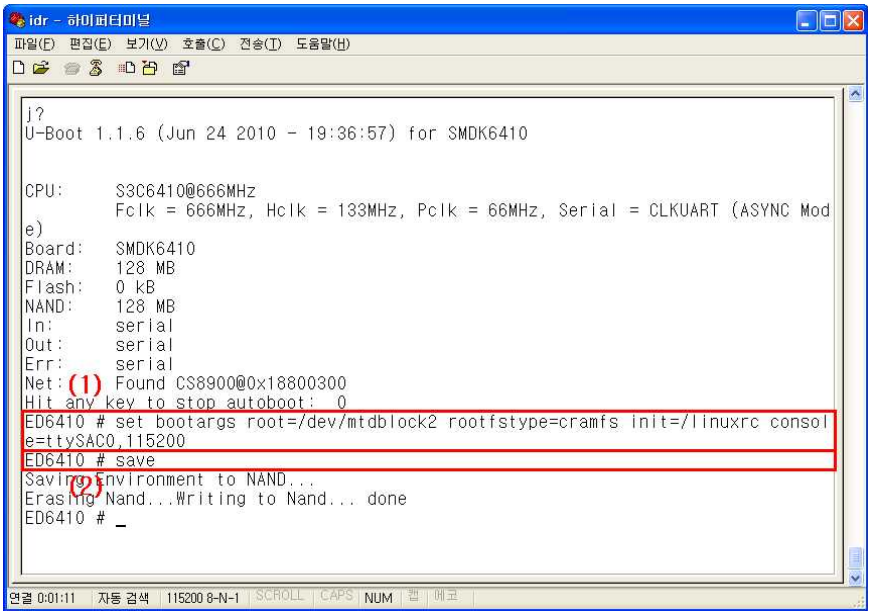
NAND Flash Mode	- A mode for driving ED-7255 based on a simulator
VMware Associative Mode	- A driving mode by using coding based on VMware

- The method of entering from a hyperterminal to 'ED6410 #' prompt mode is as follows.
  1. Activate a hyperterminal window.
  2. Power on a main controller of ED-7255.
  3. When a count is displayed in the hyperterminal window, provide Enter before the count becomes '0'.
  4. 'ED6410 #' prompt as above appears.

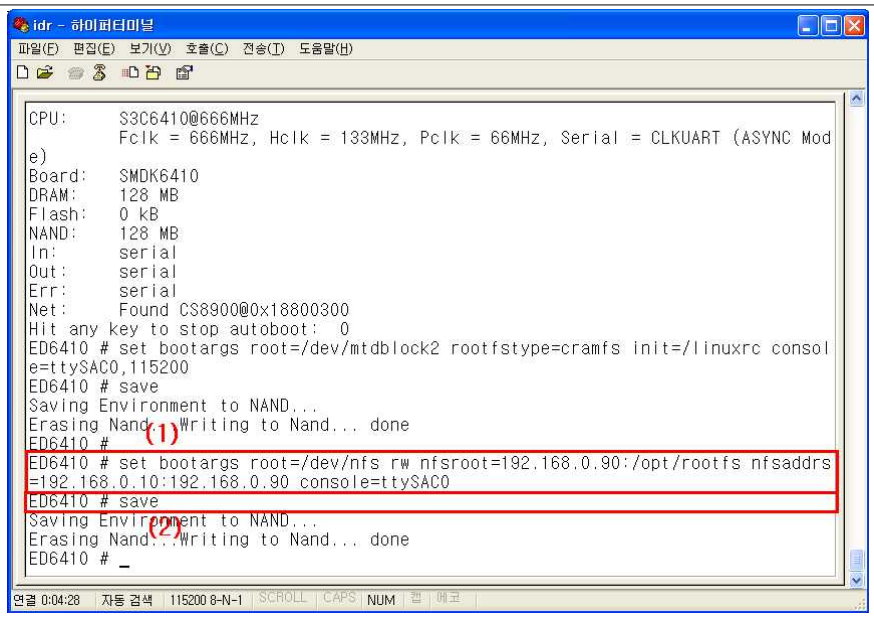
- The following contents including Linux_cmd.txt[2] file is for changing the mode setting.

※ RAMFS booting (Built-in flash) (1) set bootargs root=/dev/mtdblock2 rootfstype=cramfs init=/linuxrc console=ttySAC0,115200 (2) save
※ NFS booting (Associative to VMware) (1) set bootargs root=/dev/nfs rw nfsroot=92.168.0.90:/opt/rootfs nfsaddrs=192.168.0.10:192.168.0.90 console=ttySAC0 (2) save

2) Separately attached



<Example of a built-in flash booting setup in a hyperterminal>



<Example of a VMware associative booting setup in a hyperterminal>

※ Tip

- In VMware, double click ‘Computer’ → ‘File System’ in a wallpaper and enter the following address. There is ′profile′ file. Here, select an automatic execution mode or a manual mode (prompt standby) in a hyperterminal.

Address	root\opt\rootfs\etc

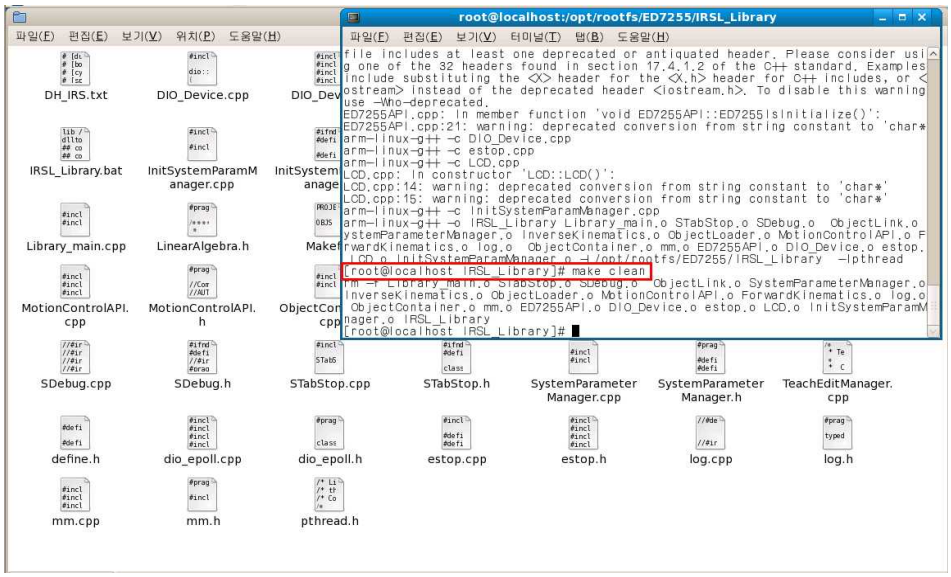
- If displaying ′##′ at the bottom of the profile, the automatic execution is not performed. ′##′ command in Linux has the effect the same as commenting.

```
/bin/echo "Start ED-7255..."
##      qpe&

    ### Some aliases
##    alias ps2='ps facux '
##    alias ps1='ps faxo "%U %t %p %a" '
##    alias af='ps af'
##    alias cls='clear'
##    alias df='df -h'
##    alias indent='indent -bad -bap -bbo -nbc -br -brs -c33 -cd33 -ncdb -ce -ci4 -cli0 -cp33 -cs
lp -npcs -npsl -nsc -nsob -nss -ts4 '
    #alias bc='bc -l'
##    alias minicom='minicom -c on'
##    alias calc='calc -Cd '
##    alias bc='calc -Cd '
fi;

/bin/echo "Device driver loading..."
insmod /dev/cmodule/dpram.ko
insmod /dev/cmodule/lcd.ko
insmod /dev/cmodule/relay.ko
insmod /dev/cmodule/iodev.ko
insmod /dev/cmodule/dac.ko
insmod /dev/cmodule/adc.ko
insmod /dev/cmodule/estop.ko
/bin/echo "Complete!!!"
/bin/echo ""

##/ED7255/testexample/lcd_app/lcd_test

cd /Qtopia/Project
##/ED7255/IRSL/IRSL
```

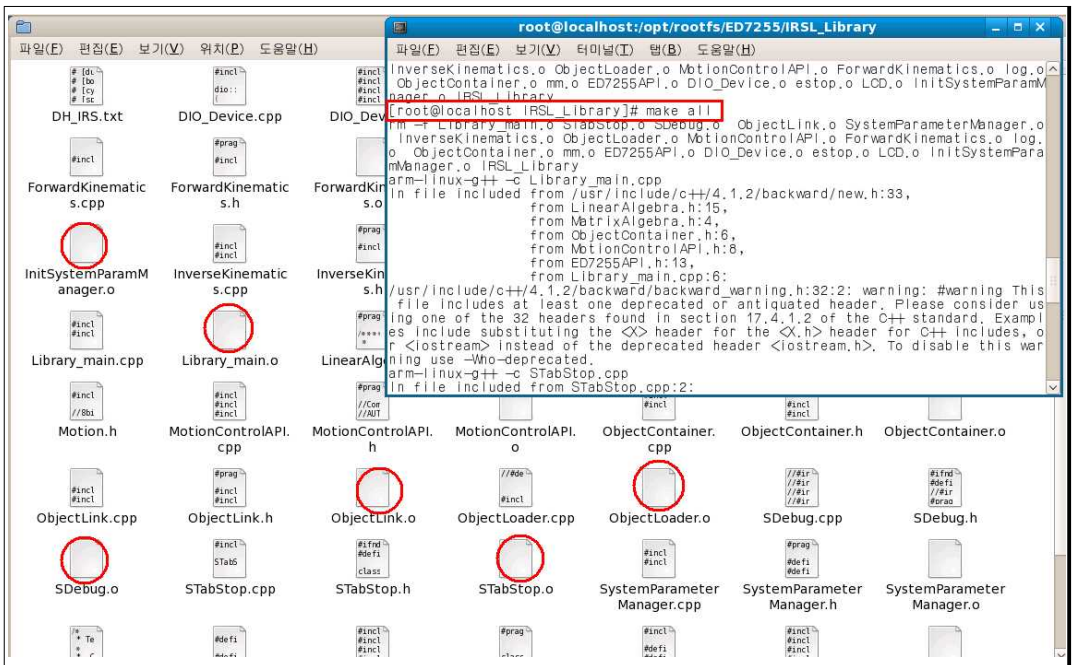
<Cancelling automatic execution mode in profile>

2. If library functions of VMware were changed, object files must be updated to be adapted to the contents being used from now. To do so, delete the old object files or overwrite it as follows. The following commands are used in VMware Fedora(Linux) terminal.

Delete Object File	make clean
Make Object File	make all



<Object files deleted by 'make clean' command>

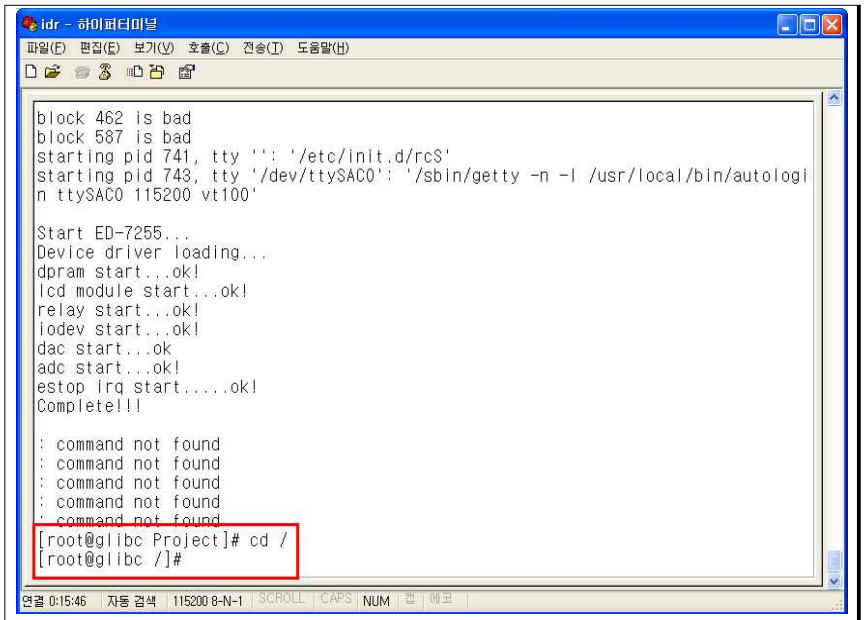


<Object files created by 'make all' command>

3. As above, if the library was directly changed, a system may be driven by two methods. First, as to an automatic mode, boot a system with 'VMware associative' mode, remove the commenting('##') in a profile, and operate the system in a pendant mode. Second, with regard to a manual mode, perform the commenting('##') in a profile and operate the system manually. The manual operation method is as follows.

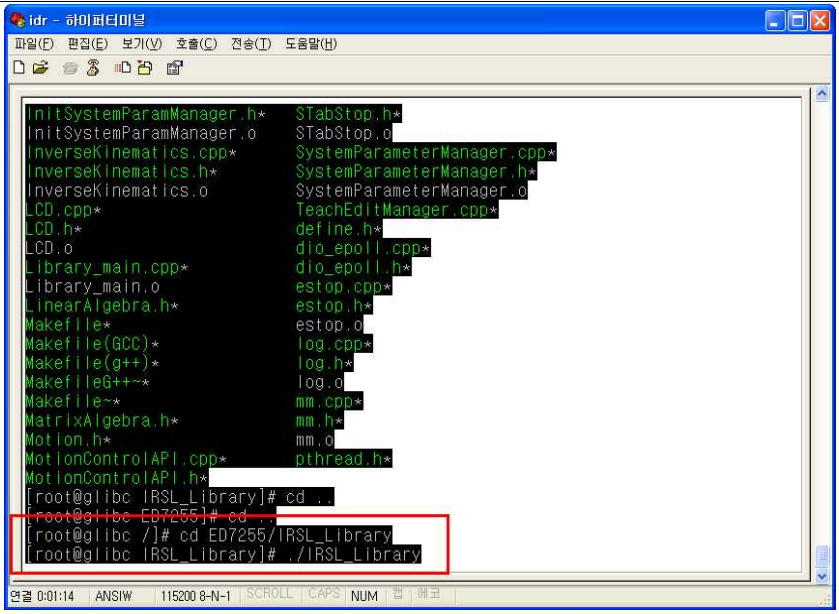
(※ Caution : In a manual mode, commenting of a profile (cd/Qtopia/Project ##/ED-7255/IRSL/IRSL) must be performed before operation.)

- Activate a hyperterminal window, and power on a main controller of ED-7255. If '[root@glibc Project]#' prompt appears, type as follows.



<Change prompt after powering on ED-7255 main controller>

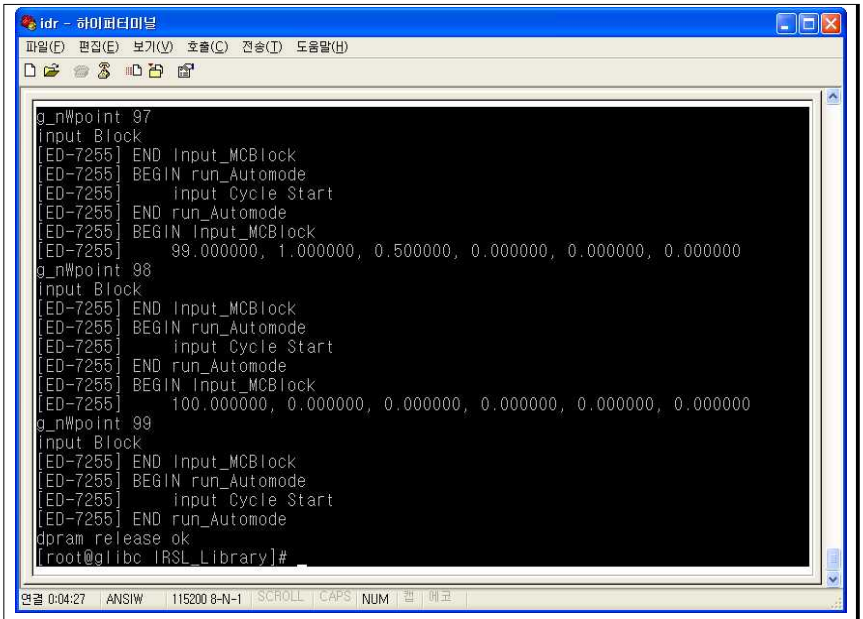
- Move to 'ED-7255/IRSL_Library' directory, and execute './IRSL_Library' command.



<Change prompt after powering on ED-7255 main controller>

- The already written program is executed, and the following prompt is created.

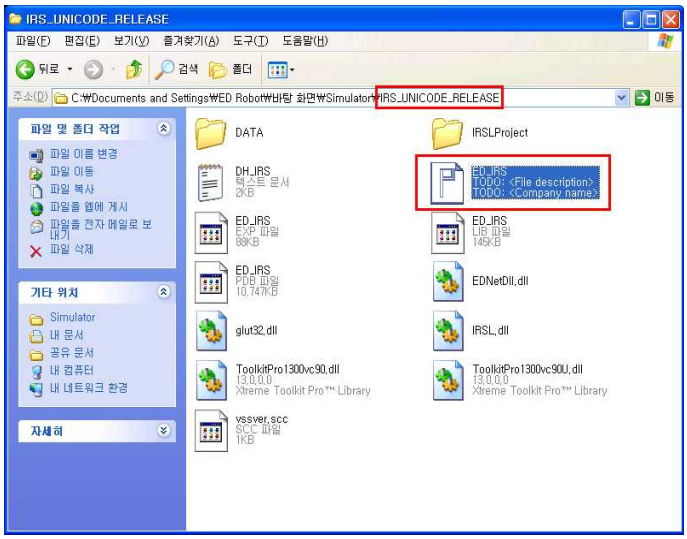
(※ Data value for each joint and other information are displayed.)



<Change prompt after powering on ED-7255 main controller>

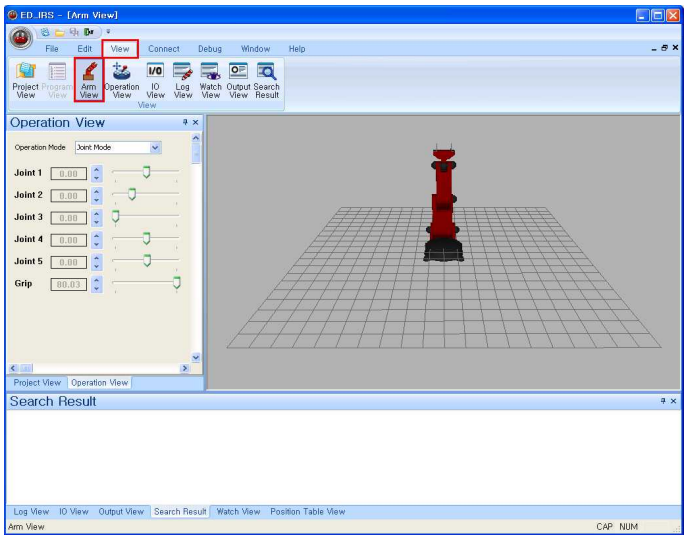
Simulator Based Operation

1. In the above, we described the operation based on library change. From now on, we will describe the operation based on a simulator by using a teaching pendant. Execute 'ED_IRS' file in the following folder.
   (※ A simulator must be operated without the commenting of a profile (cd /Qtopia/Project/ED-7255/IRSL/IRSL).)



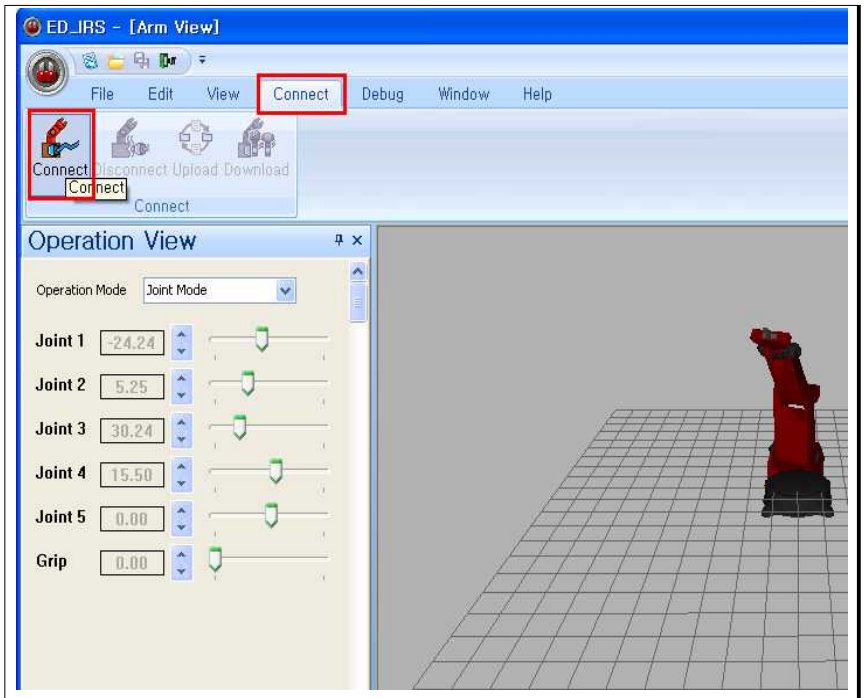
<Select ED_IRS file in IRS_UNICODE_RELEASE folder>

2. From Main Menu, select 'View' tab and click 'Arm view', and then a robot arm is shown as follows. Power on a main controller of ED-7255.



<Screen appearing by selecting Arm view>

3. From Main Menu, select 'Connect' tab and click 'Connect', and then a popup window for linking a robot with a simulator appears. If the linkage is successful, the popup window disappears.



<Connect >

4. If the linkage or connection is completed, select No. 3 (operation mode) of a teaching pendant. Select AXIS1 to 5 or GRIP button and move a red up/down button. You can see that an arm in Arm View window moves together.

5. Besides, there is a method using a simple operation language. However, this is fully described in ED-Robot book, its further description will be omitted.

Library_Main.cpp

```
#include <stdio.h>
#include <stdlib.h>

#include "ED-7255API.h"

#include <signal.h>
#include <time.h>
#include <malloc.h>
#include <string.h>
#include <sys/mman.h>

#include "pthread.h"
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <unistd.h>
#include <dirent.h>

ED-7255API* pAPI;
int main(int argc, char * argv[])
{
        int i=1;
        pAPI = new ED-7255API;
        pAPI->ED-7255IsInitialize();
        //Target setter values for joint No. 1 to 6
        float j[6] = {0,0,0,0,0,0};
        //Operation speed
        float v = 50;
        for(int i=0;i<100;i++){
                //Increase a joint No. 1 by 1 degree.
                j[0] += 1;
                if(i<49){
                        j[1] += 1;
                        j[2] += 0.5f;
                }
                if(i>50){
                        j[1] -= 1;
                        j[2] -= 0.5f;
                }pAPI->ED-7255_v_move_J_as(j,v);
```

```
        }
        while(i){
                sleep(10);
                i=0;
        }
        delete pAPI;
        pAPI = NULL;
        return 0;
}
```

Linux_cmd.txt

```
==================================
u-boot download  //Checking a network by 'print' is required first.
==================================
nand scrub
nand erase 0 40000
tftp c0000000 u-boot.bin
nand write c0000000 0 40000
==================================
==================================
zImage download - kernel
==================================
tftp c0000000 zImage
nand erase 40000 1c0000
nand write c0000000 40000 1c0000
----------------------------------
// create kernal zImage
make menuconfig
make clean
make oldconfig
make
----------------------------------
==================================
==================================
rootfs download - filesystem
==================================
tftp c0000000 rootfs.cramfs
nand erase 200000 3000000
nand write c0000000 200000 3000000
----------------------------------
// create ramfs
cd /opt
mkfs.cramfs rootfs rootfs.cramfs
cp rootfs.cramfs /tftpboot
----------------------------------
==================================
// ramfs bootting (Built-in flash)
set bootargs root=/dev/mtdblock2 rootfstype=cramfs init=/linuxrc console=ttySAC0,115200
save

// nfs bootting (VMware associative)
set bootargs root=/dev/nfs rw nfsroot=192.168.0.90:/opt/rootfs nfsaddrs=192.168.0.10:192.168.0.90 console=ttySAC0
save
```

```
========================================
U-Boot 1.1.6 (Jul  3 2009 - 17:03:06) for SMDK6410


CPU:       S3C6410@666MHz
           Fclk = 666MHz, Hclk = 133MHz, Pclk = 66MHz, Serial =
CLKUART (ASYNC Mode)
Board:     SMDK6410
DRAM:      128 MB
Flash:     0 kB
NAND:      128 MB
In:        serial
Out:       serial
Err:       serial
Net:       Found CS8900@0x18800300
Hit any key to stop autoboot:  0
ED-7255 # print
bootcmd=nand read c0008000 40000 1c0000;bootm c0008000
bootdelay=3
baudrate=115200
ethaddr=00:40:5c:26:0a:5b
filesize=34000
fileaddr=C0000000
gatewayip=192.168.0.1
netmask=255.255.255.0
ipaddr=192.168.0.10
serverip=192.168.0.90
bootargs=root=/dev/nfs rw nfsroot=192.168.0.90:/opt/rootfs
nfsaddrs=192.168.0.10:192.168.0.90 console=ttySAC0
stdin=serial
stdout=serial
stderr=serial

Environment size: 379/16380 bytes

========================================

=============================================
vaware setting
=============================================
< Host Virtual Network Mapping >
VMnet0: Bridged to an automatically chosen adapter
VMnet1: VMware Network Adapter VMnet1
VMnet2: Local net 2 -- LAN connected with the equipment
VMnet3: Local net 1 -- LAN connected with an external LAN wire
VMnet8: VMware Network Adapter VMnet8
```

< NAT >
VMnet host: VMnet8
See Gateway IP address.

==========
linux setting
==========
< eth0 >
Set to DHCP
Primary DNS : 192.168.93.2(Gateway IP Address value for NAT)

< eth1 >
Set to manual ip
192.168.0.90
255.255.255.0
Primary DNS : 192.168.93.2(Gateway IP Address value for NAT)

PART 9
Robot Control Library

9-1. ED-7255IsInitialize

9-2. ED-7255StopService()

9-3. ED-7255set_inposition_enable

9-4. ED-7255get_inposition_enable

9-5. ED-7255get_inposition_state

9-6. ED-7255set_dcmotor_enable

9-7. ED-7255get_dcmotor_enable

9-8. ED-7255set_dcmotorPower_OnOff

9-9. ED-7255get_dcmotorPower_OnOff

9-10. ED-7255enable_MotorPower_all

9-11. ED-7255set_soft_limit_sel

9-12. ED-7255get_soft_limit_sel

9-13. ED-7255set_soft_plimit_position

9-14. ED-7255get_soft_plimit_position

9-15. ED-7255set_soft_nlimit_position

9-16. ED-7255get_soft_nlimit_position

9-17. ED-7255setOverride

9-18. ED-7255getOverride

9-19. ED-7255get_remain_angle

ED Corporation

9-20. ED-7255get_remain_angle
9-21. ED-7255set_P_gain
9-22. ED-7255set_I_gain
9-23. ED-7255set_D_gain
9-24. ED-7255get_P_gain
9-25. ED-7255get_I_gain
9-26. ED-7255get_D_gain
9-27. ED-7255set_EmergencyStop
9-28. ED-7255set_ReleaseEmergencyStop
9-29. ED-7255setDigitalOutput_ON
9-30. ED-7255setDigitalOutput_OFF
9-31. ED-7255getDigitalInput
9-32. ED-7255setLED_on
9-33. ED-7255setLED_on
9-34. ED-7255_v_move_J_as
9-35. ED-7255_v_move_J_ra
9-36. ED-7255_t_move_J_as
9-37. ED-7255_t_move_J_ra
9-38. ED-7255_v_move_CL_as
9-39. ED-7255_v_move_CL_ra
9-40. ED-7255_t_move_CL_as
9-41. ED-7255_t_move_CL_ra
9-42. ED-7255_v_move_JL_as
9-43. ED-7255_v_move_JL_ra
9-44. ED-7255_t_move_JL_as
9-45. ED-7255_t_move_JL_ra

PART 9

Robot Control Library

9-1. ED-7255IsInitialize

1) Purpose

Initialize a library for controlling ED 7255 dedicated board.

2) Format

int ED-7255IsInitialize()

3) Input

None

4) Output

TRUE : Library initialization sucess
FALSE : Library initialization fail

5) Description

Initialize a main board and robot information for robot control.

6) Example

```
if(ED-7255IsInitialize() == 1){
  cout<<"Initial. success"<<endl ;
}
else if(ED-7255IsInitialize() == 0){
  cout<<"Initial. fail"<<endl;
}
```

7) See Also

void ED-7255StopService();

9-2. ED-7255StopService()

1) Purpose

Clear a memory assigned by initialization of a dedicated library.

2) Format

void ED-7255StopService();

3) Input

None

4) Output

None

5) Description

Memories assigned when a library is stopped are cleared.

This is different from a function of stopping a robot while in operation.

6) Example

```
//Clear a memory when a destructor of C++ or a
    program is completed.
ED-7255StopService();
```

7) See Also

int ED-7255IsInitialize()

9-3. ED-7255set_inposition_enable

1) Purpose

Enable performing inposition in an instructed target angle.

2) Format

void ED-7255set_inposition_enable(int ip);

3) Input

ip: 1 - Check Inposition
0 - Not check Inposition

4) Output

None

5) Description

Inposition means that a system first confirms whether a joint arrives in a target angle, and then performs the next instruction.

Inposition is set when exact behavior is required, but disturbs smooth behavior. Especially, it is not necessary to continuously perform A action and B action.

6) Example

```
//Call when inposition is required.
int inposition = 1;
ED-7255set_inposition_enable(inposition);
```

7) See Also

int ED-7255get_inposition_enable(), short* ED-7255get_inposition_state()

9-4. ED-7255get_inposition_enable

1) Purpose

Used to confirm whether a command being performed checks the inposition.

2) Format

int ED-7255get_inposition_enable();

3) Input

None

4) Output

TRUE: Inposition check is now being performed.
FALSE: Inposition check is not performed now.

5) Description

6) Example

```
//Confirm inposition check
if( ED-7255get_inposition_enable() == 1)
{
  cout<<"Inposition check"<<endl;
}
else if(ED-7255get_inposition_enable() == 0)
{
  cout<<"Clear inposition check"<<endl;
}
```

7) See Also

int ED-7255set_inposition_enable(), short* ED-7255get_inposition_state()

9-5. ED-7255get_inposition_state

1) Purpose

After performing ED-7255set_inposition_enable(1), it can be known that inposition has been performed for a target angle through a real motion control.

2) Format

short* ED-7255get_inposition_state();

3) Input

None

4) Output

short ip[8]; Return the inposition state value for 8 encoders.

5) Description

ED 7255 has 6 motors and can operate two redundancy motors. It can perform 8 returns. The values of an array No. 0 to 5 may be referred to, for inposition state for a robot.

6) Example

```
//Confirm whether inposition has been performed.
short ip[8];
ip = ED-7255get_inposition_state();

for(int j=0; j<6 ; j++)
    cout<< j <<" joint - Inposition state: "<<ip[j] <<endl;
```

7) See Also

int ED-7255set_inposition_enable(), void ED-7255set_inposition_enable(int ip)

9-6. ED-7255set_dcmotor_enable

1) Purpose

Set a motor to be used and a motor not to be used during motion control.

2) Format

void ED-7255set_dcmotor_enable(short* disable);

3) Input

short dc[8]; Enter enable or disable for 8 motors.

1 - enable

0 - disable

4) Output

None

5) Description

Select a motor to be included in motion control. If Disable is selected, the pertinent motor does not operate even though a target is given. However, this is different from Motor Off.

6) Example

```
//Use or non-use of a motor (DC
    Motor)
short dcable[8] = {1,1,1,1,1,0,0,0};
ED-7255set_dcmotor_enable(dcable);
```

7) See Also

short* ED-7255get_dcmotor_enable()

9-7. ED-7255get_dcmotor_enable

1) Purpose

Find enable or disable state of a motor now performing the command.

2) Format

short* ED-7255get_dcmotor_enable();

3) Input

None

4) Output

short dcenable[8];
Return short-type state value for 8 motors.

5) Description

Among the motors now operating, a motor whose status is 'enable' or a motor whose status is 'disable' can be found.

6) Example

```
//Find use or non-use of a motor (DC
    Motor).
#define DISABLE 0
#define ENABLE 1

short dcable[8];
dcable = ED-7255get_dcmotor_enable();
if(dcable[2] == DISABLE )
{
   dcable[2] = ENABLE;
   ED-7255set_dcmotor_enable(dcable);
}
```

7) See Also

short* ED-7255set_dcmotor_enable()

9-8. ED-7255set_dcmotorPower_OnOff

1) Purpose

Power on/off a motor.

2) Format

void ED-7255set_dcmotorPower_OnOff(short *dcPower);

3) Input

dcPower[8] The short-type on/off value for motor No. 1 to 8.

4) Output

None

5) Description

Power source to a motor can be controlled. Motor on for '1', and motor off for '0'.

6) Example

```
//Set the power source to a motor (DC Motor).

short dcpower[8]= {1,1,0,1,1,0,0,0};
ED-7255set_dcmotorPower_OnOff(dcpower);
```

7) See Also

short* ED-7255get_dcmotorPower_OnOff(), void ED-7255enable_MotorPower_all()

9-9. ED-7255get_dcmotorPower_OnOff

1) Purpose

Find power-on or off status of DC motor.

2) Format

short* ED-7255get_dcmotorPower_OnOff();

3) Input

None

4) Output

short dcpower[8];
Return the power status for 8 motors.

5) Description

Motor powered status is found. Motor On for '1', and Motor Off for '2'.

6) Example

```
//Find power-on/off of a motor (DC Motor).
#define ON 1
#define OFF 0

short dcpower[8];
dcpower = ED-7255get_dcmotorPower_OnOff();
if(dcpower[2] == OFF )
{
   dcpower[2] = ON;
   ED-7255set_dcmotorPower_OnOff(dcpower);
}
```

7) See Also

short* ED-7255set_dcmotor_enable(), void ED-7255enable_MotorPower_all();

9-10. ED-7255enable_MotorPower_all

1) Purpose

Power on and enable all the motors.

2) Format

void ED-7255enable_MotorPower_all();

3) Input

None

4) Output

None

5) Description

All of the motors are powered on and use of all of the motors are enabled. 6) Example

```
//Enable a motor (DC Motor).
ED-7255enable_MotorPower_all();
```

7) See Also

ED-7255set_dcmotorPower_OnOff(short*dcPower), ED-7255get_dcmotorPower_OnOff();

9-11. ED-7255set_soft_limit_sel

1) Purpose

Set use or non-use of a soft-limit.

2) Format

void ED-7255set_soft_limit_sel(short* Softchk);

3) Input

short* Softchk 8 short-type arrays

4) Output

None

5) Description

For High and Low values of a joint limit established for each joint,

which joint is to be checked a soft-limit can be set.

6) Example

```
//Set a joint to be checked a soft-limit.
short softlimi[8] = {1,1,1,1,1,0,0,0}
ED-7255set_soft_limit_sel(softlimi);
```

7) See Also

short* ED-7255get_soft_limit_sel();

void ED-7255set_soft_plimit_position(float* SoftH);

float* ED-7255get_soft_plimit_position();

void ED-7255set_soft_nlimit_position(float* SoftL);

float* ED-7255get_soft_nlimit_position();

9-12. ED-7255get_soft_limit_sel

1) Purpose

Investigate which joint is checked a soft-limit.

2) Format

short* ED-7255get_soft_limit_sel();

3) Input

None

4) Output

Returns a joint number, which is being checked a joint limit, from Joint No. 1 to 8.

1 - Joint limit is being checked, 0 - Joint limit is not checked.

5) Description

A joint number now being checked a joint limit can be found.

6) Example

```
//Set a joint to be checked a soft-limit.
short softlimi[8];
softlimi = ED-7255get_soft_limit_sel();

for(int j=0;j<8;j++)
    cout<<j+1<<"State of checking n'th joint limit"<< softlimi[j]
     <<endl;
```

7) See Also

void ED-7255set_soft_limit_sel(short* Softchk);

void ED-7255set_soft_plimit_position(float* SoftH);

float* ED-7255get_soft_plimit_position();

void ED-7255set_soft_nlimit_position(float* SoftL);

float* ED-7255get_soft_nlimit_position();

9-13. ED-7255set_soft_plimit_position

1) Purpose

Set the upper limit value(positive direction) of a soft-limit.

2) Format

void ED-7255set_soft_plimit_position(float* SoftH);

3) Input

SoftH[8] positive limit value for each joint

4) Output

None

5) Description

Set the soft-limit value in the positive direction.

6) Example

```
//Set a joint to be checked a soft-limit.
float softlimiH[8]={175.0f, 80.0f, 75.0f, 170.0f, 170.0f, 325.0f, 0.0f, 0.0f};
ED-7255set_soft_plimit_position(softlimiH);
```

7) See Also

void ED-7255set_soft_limit_sel(short* Softchk);

short* ED-7255get_soft_limit_sel();

void ED-7255set_soft_limit_sel(short* Softchk);

float* ED-7255get_soft_plimit_position();

float* ED-7255get_soft_nlimit_position();

9-14. ED-7255get_soft_plimit_position

1) Purpose

Receive the positive direction soft-limit value now being set.

2) Format

float* ED-7255get_soft_plimit_position();

3) Input

None

4) Output

float plimi[8];

5) Description

Receive a soft-limit value of positive direction.

6) Example

```
//Set a joint to be checked a soft-limit.
float softlimiH[8];
softlimiH = ED-7255get_soft_plimit_position();
```

7) See Also

short* ED-7255get_soft_limit_sel();

void ED-7255set_soft_limit_sel(short* Softchk);

void ED-7255set_soft_plimit_position(float* SoftH);

void ED-7255set_soft_nlimit_position(float* SoftL);

float* ED-7255get_soft_nlimit_position();

9-15. ED-7255set_soft_nlimit_position

1) Purpose

Set the lower limit value(negative direction) of a soft-limit.

2) Format

void ED-7255set_soft_nlimit_position(float* SoftL);

3) Input

SoftL[8] Negative limit value for each joint

4) Output

None

5) Description

Set the soft-limit value in the negative direction.

6) Example

```
//Set a joint to be checked a soft-limit.
float softlimiL[8]={-175.0f,-20.0f,0.0f,-170.0f,-170.0f, 0.0f, 0.0f, 0.0f};
ED-7255set_soft_nlimit_position(softlimiL);
```

7) See Also

short* ED-7255get_soft_limit_sel();

void ED-7255set_soft_limit_sel(short* Softchk);

float* ED-7255get_soft_plimit_position();

void ED-7255set_soft_plimit_position(float* SoftH);

float* ED-7255get_soft_nlimit_position();

9-16. ED-7255get_soft_nlimit_position

1) Purpose

Receive the negative direction soft-limit value now being set.

2) Format

float* ED-7255get_soft_nlimit_position();

3) Input

None

4) Output

float nlimi[8];

5)Description

Receive the negative soft-limit value.

6) Example

```
//Set a joint to be checked a soft-limit.
float softlimiL[8];
softlimiL = ED-7255get_soft_nlimit_position();
```

7) See Also

short* ED-7255get_soft_limit_sel();

void ED-7255set_soft_limit_sel(short* Softchk);

void ED-7255set_soft_plimit_position(float* SoftH);

float* ED-7255get_soft_plimit_position();

void ED-7255set_soft_nlimit_position(float* SoftL);

9-17. ED-7255setOverride

1) Purpose

Override performance for rapid confirmation of operation.

2) Format

void ED-7255setOverride(float ovride);

3) Input

override speed *= set 1.0f for a fundamental override

4) Output

None

5) Description

Increase motion performance speed by multiplying the motion speed now being performed by the input value.

6) Example

```
//Increase 1.5 times the
    fundamental speed.
float override = 1.5f;
ED-7255setOverride(override);
```

7) See Also

float ED-7255get_velocity_override();

9-18. ED-7255getOverride

1) Purpose

Find the scaling factor of override now being adapted to.

2) Format

float ED-7255getOverride();

3) Input

None

4) Output

Motion speed of float-type override

5) Description

Find the scaling factor of override now being performed.

6) Example

```
//Increase 1.5 times the fundamental speed.
float fRide=0.0f;
fRide  = ED-7255getOverride();
cout<<"Scaling factor of override for the current motion is"<<fRide<<"."<<endl;
```

7) See Also

float ED-7255set_velocity_override();

9-19. ED-7255get_remain_angle

1) Purpose

In automatic mode, find the remaining angle value up to completion of a command.

2) Format

float* ED-7255get_remain_angle();

3) Input

None

4) Output

float remainAng[8]; The remaining angle value to a target for each joint.

5) Description

Receive the remaining angle value to complete the command now being executed.

6) Example

```
//Increase 1.5 times the fundamental speed.
float remainAng[8] ;
remainAng = ED-7255get_remain_angle();

for(int j=0;j<8;j++)
 cout<<j<<"Up to joint arrival," <<remainAng[j] <<"remains."<<endl;
```

7) See Also

9-20. ED-7255get_remain_angle

1) Purpose

Read the angle value read out of an encoder of a robot.

2) Format

float* ED-7255get_current_angle();

3) Input

None

4) Output

float currentAng[8];

5) Description

Read the current angle value of a robot.

6) Example

```
//Increase 1.5 times the fundamental speed.
float currentAng[8] ;
currentAng = ED-7255get_current_angle();

for(int j=0;j<8;j++)
 cout<<j<<"Current angle of a joint is"<<currentAng[j] <<"."<<endl;
```

7) See Also

9-21. ED-7255set_P_gain

1) Purpose

Set P Gain values in PID control.

2) Format

void ED-7255set_P_gain(float *Pgain);

3) Input

float Pgain[8];

4) Output

None

5) Description

Set P Gain value for each joint.

6) Example

```
//P gain setting
float PGain[8] = {15.0f, 15.0f, 6.0f, 5.0f, 5.0f, 10.0f, 0.0f, 0.0f};
ED-7255set_P_gain(PGain);
```

7) See Also

void ED-7255set_I_gain(float *Igain);

void ED-7255set_D_gain(float *Dgain);

float* ED-7255get_P_gain();

float* ED-7255get_I_gain();

float* ED-7255get_D_gain();

9-22. ED-7255set_I_gain

1) Purpose

Set I Gain values in PID control.

2) Format

void ED-7255set_I_gain(float *Igain);

3) Input

float Igain[8];

4) Output

None

5) Description

Set I Gain value for each joint.

6) Example

```
//I gain setting
float IGain[8] = {0.1f, 0.1f, 0.1f, 0.1f, 0.1f, 0.1f, 0.0f, 0.0f};
ED-7255set_I_gain(IGain);
```

7) See Also

void ED-7255set_P_gain(float *Pgain);

void ED-7255set_D_gain(float *Dgain);

float* ED-7255get_P_gain();

float* ED-7255get_I_gain();

float* ED-7255get_D_gain();

9-23. ED-7255set_D_gain

1) Purpose

Set D Gain values in PID control.

2) Format

void ED-7255set_D_gain(float *Dgain);

3) Input

float Dgain[8];

4) Output

None

5) Description

Set D Gain value for each joint.

6) Example

```
//I gain setting
float DGain[8] = {6.0f, 6.0f, 6.0f, 1.0f, 1.0f, 1.0f, 0.0f, 0.0f};
ED-7255set_D_gain(DGain);
```

7) See Also

void ED-7255set_P_gain(float *Pgain);

void ED-7255set_I_gain(float *Igain);

float* ED-7255get_P_gain();

float* ED-7255get_I_gain();

float* ED-7255get_D_gain();

9-24. ED-7255get_P_gain

1) Purpose

Read P gain values from PID control values for a robot.

2) Format

float* ED-7255get_P_gain();

3) Input

None

4) Output

float Pgain[8];

5) Description

Read P Gain value for each joint.

6) Example

```
//P gain read
float PGain[8] ;
PGain = ED-7255get_P_gain();
for(int j=0;j<8;j++)
  cout<<j<<"N'th P Gain value is"<<PGain[j]<<"."<<endl;
```

7) See Also

void ED-7255set_P_gain(float *Pgain);

void ED-7255set_I_gain(float *Igain);

void ED-7255set_D_gain(float *Dgain);

float* ED-7255get_P_gain();

float* ED-7255get_I_gain();

float* ED-7255get_D_gain();

9-25. ED-7255get_I_gain

1) Purpose

Read I gain values from PID control values for a robot.

2) Format

float* ED-7255get_I_gain();

3) Input

None

4) Output

float Igain[8];

5) Description

Read I Gain value for each joint.

6) Example

```
//I gain read
float IGain[8] ;
IGain = ED-7255get_I_gain();
for(int j=0;j<8;j++)
  cout<<j<<"N'th I Gain value is"<<IGain[j]<<"."<<endl;
```

7) See Also

void ED-7255set_P_gain(float *Pgain);

void ED-7255set_I_gain(float *Igain);

void ED-7255set_D_gain(float *Dgain);

float* ED-7255get_P_gain();

float* ED-7255get_I_gain();

float* ED-7255get_D_gain();

9-26. ED-7255get_D_gain

1) Purpose

Read D gain values from PID control values for a robot.

2) Format

float* ED-7255get_D_gain();

3) Input

None

4) Output

float Dgain[8];

5) Description

Read D Gain value for each joint.

6) Example

```
//P gain read
float DGain[8] ;
DGain = ED-7255get_P_gain();
for(int j=0;j<8;j++)
  cout<<j<<"N'th D Gain value is"<<DGain[j]<<"."<<endl;
```

7) See Also

void ED-7255set_P_gain(float *Pgain);

void ED-7255set_I_gain(float *Igain);

void ED-7255set_D_gain(float *Dgain);

float* ED-7255get_P_gain();

float* ED-7255get_I_gain();

float* ED-7255get_D_gain();

9-27. ED-7255set_EmergencyStop

1) Purpose

Enter the EM Stop mode.

2) Format

void ED-7255set_EmergencyStop();

3) Input

None

4) Output

None

5) Description

This is called at the moment emergency stop is required. If clearing the emergency stop, a command must be re-entered since the previously entered commands are all cleared.

6) Example

```
//Emergency Stop Call
ED-7255set_EmergencyStop();
```

7) See Also

void ED-7255set_ReleaseEmergencyStop();

9-28. ED-7255set_ReleaseEmergencyStop

1) Purpose

Clear EM Stop mode. After clearing the mode, the current position is remained.

2) Format

void ED-7255set_ReleaseEmergencyStop();

3) Input

None

4) Output

None

5) Description

Clear EM Stop mode.

6) Example

```
//Release Emergency Stop
ED-7255set_ReleaseEmergencyStop()
    ;
```

7) See Also

void ED-7255set_EmergencyStop()

9-29. ED-7255setDigitalOutput_ON

1) Purpose

Change a digital output port to 'high' state.

2) Format

void ED-7255setDigitalOutput_ON(int nSelectNumber);

3) Input

Output port numbers from 0 to 7

4) Output

None

5) Description

Receive the port number and change the signal of the relevant port to 'high'.

6) Example

```
//Change a signal of the relevant port
    to 'high'.
for(int j=0;j<4;j++)
   ED-7255setDigitalOutput_ON(j);
```

7) See Also

void ED-7255setDigitalOutput_OFF(int nSelectNumber)
char* ED-7255getDigitalInput()

9-30. ED-7255setDigitalOutput_OFF

1) Purpose

Change a digital output port to 'low' state.

2) Format

void ED-7255setDigitalOutput_OFF(int nSelectNumber);

3) Input

Output port numbers from 0 to 7

4) Output

None

5) Description

Receive the port number and change the signal of the relevant port to 'low'.

6) Example

```
//Change a signal of the relevant port
    to 'low'.
for(int j=0;j<4;j++)
    ED-7255setDigitalOutput_Off(j);
```

7) See Also

void ED-7255setDigitalOutput_On(int nSelectNumber) char* ED-7255getDigitalInput()

9-31. ED-7255getDigitalInput

1) Purpose

Read the signal status of a digital input port.

2) Format

char* ED-7255getDigitalInput();

3) Input

None

4) Output

The status value of input ports 0 to 7

5) Description

Read the signal status incoming from outside.

6) Example

```
char input[8];
input =    ED-7255getDigitalInput();
for(int j=0;j<8;j++)
 cout<<j<<"번 input state"<<input[j]<<endl;
```

7) See Also

void ED-7255setDigitalOutput_On(int nSelectNumber) ,
void ED-7255setDigitalOutput_Off(int nSelectNumber)

9-32. ED-7255setLED_on

1) Purpose

Control LED of a main control box.

2) Format

void ED-7255setLED_on(int nNumber);

3) Input

Port No. 0 to 2 to turn on the LED.

4) Output

None

5) Description

Control an external LED of a main control box.

6) Example

```
ED-7255setLED_on(0);
ED-7255setLED_on(1);
```

7) See Also

void ED-7255setLED_off(int nNumber);

9-33. ED-7255setLED_on

1) Purpose

Control LED of a main control box.

2) Format

void ED-7255setLED_off(int nNumber);

3) Input

Port No. 0 to 2 to turn off the LED.

4) Output

None

5) Description

Control an external LED of a main control box.

6) Example

```
ED-7255setLED_off(0)
    ;
ED-7255setLED_off(1)
    ;
```

7) See Also

void ED-7255setLED_on(int nNumber);

9-34. ED-7255_v_move_J_as

1) Purpose

Move a joint to a target and with a speed received based on the absolute position.

2) Format

int ED-7255_v_move_J_as(float *target, float Velocity);

3) Input

float target[8] Target angle in absolute value.
Velocity Motion speed

4) Output

1 - Buffer input success
0 - Buffer input fail

5) Description

Receive the target angle in absolute value from a reference coordinate, and operate a robot with the received speed.

6) Example

```
float joint[6]={0,30,41,30,0,0};
float v = 50;
ED-7255_v_move_J_as(joint,v);
```

7) See Also

int ED-7255_v_move_J_ra(float *target,float Velocity);
int ED-7255_t_move_J_as(float *target,float time);

int ED-7255_t_move_J_ra(float *target,float time);
int ED-7255_v_move_CL_as(float *vec,float Velocity);
int ED-7255_v_move_CL_ra(float *vec,float Velocity);
int ED-7255_t_move_CL_as(float *vec,float time);
int ED-7255_t_move_CL_ra(float *vec,float time);

9-35. ED-7255_v_move_J_ra

1) Purpose

Move a joint to a target and with a speed received based on the relative position.

2) Format

int ED-7255_v_move_J_ra(float *target, float Velocity);

3) Input

float target[8]	Target angle in relative value
Velocity	Motion speed

4) Output

1 - Buffer input success
0 - Buffer input fail

5) Description

Receive the target angle in relative value with reference to the present joint angle value, and operate a robot with the received speed.

6) Example

```
//Move a joint No. 3 by 10 degree from the present
    position.
float joint[6]={0,0,10,0,0,0};
float v = 50;
ED-7255_v_move_J_ra(joint,v);
```

7) See Also

int ED-7255_v_move_J_as(float *target, float Velocity);

int ED-7255_t_move_J_as(float *target,float time);

int ED-7255_t_move_J_ra(float *target,float time);

int ED-7255_v_move_CL_as(float *vec,float Velocity);

int ED-7255_v_move_CL_ra(float *vec,float Velocity);

int ED-7255_t_move_CL_as(float *vec,float time);

int ED-7255_t_move_CL_ra(float *vec,float time);

9-36. ED-7255_t_move_J_as

1) Purpose

Move to a target for a time received based on the absolute value.

2) Format

int ED-7255_t_move_J_as(float *target, float time)

3) Input

float target[8] Target angle in absolute value

time Arrival time

4) Output

1 - Buffer input success

0 - Buffer input fail

5) Description

Receive the target angle in absolute value from a absolute coordinate, and operate a robot with the angle for a given time.

6) Example

```
//Move a joint No. 3 by 10 degree from the present
    position.
float joint[6]={0,30,41,30,0,0};
float time = 3;
ED-7255_t_move_J_as(joint,t);
```

7) See Also

int ED-7255_v_move_J_as(float *target, float Velocity);
int ED-7255_t_move_J_as(float *target,float time);
int ED-7255_v_move_CL_as(float *vec,float Velocity);
int ED-7255_v_move_CL_ra(float *vec,float Velocity);
int ED-7255_t_move_CL_as(float *vec,float time);
int ED-7255_t_move_CL_ra(float *vec,float time);

9-37. ED-7255_t_move_J_ra

1) Purpose

Move to a target for a time received based on the relative value.

2) Format

int ED-7255_t_move_J_ra(float *target,float time);

3) Input

float target[8] Target angle in relative value

time Arrival time

4) Output

1 - Buffer input success

0 - Buffer input fail

5) Description

Receive the target angle in relative value with reference to the present joint angle value, and operate a robot with the angle for a given time.

6) Example

```
//Move a joint No. 3 by 10 degree from the present
    position.
float joint[6]={0,0,10,0,0,0};
float t= 6;
ED-7255_t_move_J_ra(joint,t);
```

7) See Also

int ED-7255_v_move_J_as(float *target, float Velocity);

int ED-7255_t_move_J_as(float *target,float time);

int ED-7255_v_move_CL_as(float *vec,float Velocity);

int ED-7255_v_move_CL_ra(float *vec,float Velocity);

int ED-7255_t_move_CL_as(float *vec,float time);

int ED-7255_t_move_CL_ra(float *vec,float time);

9-38. ED-7255_v_move_CL_as

1) Purpose

Perform linearly motion to a target for a time received based on the absolute value.

2) Format

int ED-7255_v_move_CL_as(float *vec, float Velocity);

3) Input

float target[3] Target cartesian position in absolute value
0: X coordinate value, 1: Y coord. value, 2: Z coord. value
Velocity Motion speed

4) Output

1 - Buffer input success
0 - Buffer input fail

5) Description

Receive the target position in absolute value from a reference coordinate and extract RPY value from the present joint angle, and perform the linear motion with the received speed.

6) Example

```
float joint[6]={270,0,40};
float v = 50;
ED-7255_v_move_CL_as(joint,v);
```

7) See Also

int ED-7255_v_move_J_ra(float *target,float Velocity);
int ED-7255_t_move_J_as(float *target,float time);
int ED-7255_t_move_J_ra(float *target,float time);
int ED-7255_v_move_CL_ra(float *vec,float Velocity);
int ED-7255_t_move_CL_as(float *vec,float time);
int ED-7255_t_move_CL_ra(float *vec,float time);

9-39. ED-7255_v_move_CL_ra

1) Purpose

Perform linear motion to a target for a time received based on the relative value.

2) Format

int ED-7255_v_move_CL_ra(float *vec,float Velocity);

3) Input

float target[3] Target cartesian position in relative value
0: X coordinate value, 1: Y coord. value, 2: Z coord. value
Velocity Motion speed

4) Output

1 - Buffer input success
0 - Buffer input fail

5) Description

Receive the target position in relative value from a reference coordinate, and perform the linear motion with the received speed.

6) Example

```
float joint[6]={0,0,20};
float v = 50;
ED-7255_v_move_CL_as(joint,v);
```

7) See Also

int ED-7255_v_move_J_ra(float *target,float Velocity);
int ED-7255_t_move_J_as(float *target,float time);

int ED-7255_t_move_J_ra(float *target,float time);
int ED-7255_v_move_CL_as(float *vec,float Velocity);
int ED-7255_t_move_CL_as(float *vec,float time);
int ED-7255_t_move_CL_ra(float *vec,float time);

9-40. ED-7255_t_move_CL_as

1) Purpose

Perform linear motion using the target and the arrival time received based on the absolute value.

2) Format

int ED-7255_t_move_CL_as(float *vec, float time);

3) Input

float target[3] Target Cartesian position in absolute value
0: X coordinate value, 1: Y coord. value, 2: Z coord. value
time Arrival time

4) Output

1 - Buffer input success
0 - Buffer input fail

5) Description

Receive the target position in relative value from a reference coordinate, and perform the linear motion with the received arrival time.

6) Example

```
float joint[3]={270,0,40};
float t = 5;
ED-7255_t_move_CL_as(joint , t);
```

7) See Also

int ED-7255_v_move_J_ra(float *target,float Velocity);

int ED-7255_t_move_J_as(float *target,float time);

int ED-7255_t_move_J_ra(float *target,float time);

int ED-7255_v_move_CL_as(float *vec,float Velocity);

int ED-7255_v_move_CL_as(float *vec,float Velocity);

int ED-7255_t_move_CL_ra(float *vec,float time);

9-41. ED-7255_t_move_CL_ra

1) Purpose

Receive the target based on the relative position and perform the linear motion with the arrival time.

2) Format

int ED-7255_t_move_CL_ra(float *vec, float time);

3) Input

float target[3] Target Cartesian position in relative value

0: X coordinate value, 1: Y coord. value, 2: Z coord. value

time Arrival time

4) Output

1 - Buffer input success

0 - Buffer input fail

5) Description

Receive the target position in relative value from a reference coordinate, and perform the linear motion with the received arrival time.

6) Example

```
float joint[3]={0,0,20};
float t = 5;
ED-7255_t_move_CL_ra(joint , t);
```

7) See Also

int ED-7255_v_move_J_ra(float *target, float Velocity);
int ED-7255_t_move_J_as(float *target, float time);
int ED-7255_t_move_J_ra(float *target, float time);
int ED-7255_v_move_CL_as(float *vec, float Velocity);
int ED-7255_v_move_CL_as(float *vec, float Velocity);
int ED-7255_t_move_CL_as(float *vec, float time);

9-42. ED-7255_v_move_JL_as

1) Purpose

Perform linear motion using the target and the speed received based on the absolute value.

2) Format

int ED-7255_v_move_JL_as(float *target, float Velocity);

3) Input

float target[8] Target angle in absolute value

Velocity Motion speed

4) Output

1 - Buffer input success

0 - Buffer input fail

5) Description

Convert a joint angle value of the absolute position into Cartesian position x, y, and z values by using forward kinematics, and perform linear motion based on the received speed.

6) Example

```
//Move a joint No. 3 by 10 degree from the present position.
float joint[6]={0,40,30,40,0,0};
float v = 50;
ED-7255_v_move_J_ra(joint,v);
```

7) See Also

int ED-7255_v_move_J_as(float *target, float Velocity);

int ED-7255_t_move_J_as(float *target,float time);

int ED-7255_t_move_J_ra(float *target,float time);

int ED-7255_v_move_CL_as(float *vec,float Velocity);

int ED-7255_v_move_CL_ra(float *vec,float Velocity);

int ED-7255_t_move_CL_as(float *vec,float time);

int ED-7255_t_move_CL_ra(float *vec,float time);

9-43. ED-7255_v_move_JL_ra

1) Purpose

Perform linear motion using the target and the speed received based

on the relative value.

2) Format

int ED-7255_v_move_JL_ra(float *target, float Velocity);

3) Input

float target[8] Target angle in relative value
Velocity Motion speed

4) Output

1 - Buffer input success
0 - Buffer input fail

5) Description

Convert a joint angle value of the relative position, received with reference to the present joint angle value of a robot, into Cartesian position x, y, and z values by using forward kinematics, and perform linear motion based on the received speed.

6) Example

```
//Move a joint No. 3 by 10 degree from the present position.
float joint[6]={0,0,10,0,0,0};
float v = 50;
ED-7255_v_move_J_ra(joint,v);
```

7) See Also

int ED-7255_v_move_J_as(float *target, float Velocity);
int ED-7255_t_move_J_as(float *target,float time);
int ED-7255_t_move_J_ra(float *target,float time);
int ED-7255_v_move_CL_as(float *vec,float Velocity);
int ED-7255_v_move_CL_ra(float *vec,float Velocity);
int ED-7255_t_move_CL_as(float *vec,float time);

int ED-7255_t_move_CL_ra(float *vec,float time);

9-44. ED-7255_t_move_JL_as

1) Purpose

Perform linear motion using the target and the arrival time received based on the absolute value.

2) Format

int ED-7255_t_move_JL_as(float *target, float time);

3) Input

float target[8] Target angle in relative value
time Arrival time

4) Output

1 - Buffer input success
0 - Buffer input fail

5) Description

Convert a joint angle value of the absolute position into Cartesian position x, y, and z values by using forward kinematics, and perform linear motion for a received time.

6) Example

```
//Move a joint No. 3 by 10 degree from the present position.
float joint[6]={0,0,10,0,0,0};
float t = 5;
ED-7255_t_move_JL_as(joint,t);
```

7) See Also

```
int ED-7255_v_move_J_as(float *target, float Velocity);
int ED-7255_t_move_J_as(float *target,float time);
int ED-7255_t_move_J_ra(float *target,float time);
int ED-7255_v_move_CL_as(float *vec,float Velocity);
int ED-7255_v_move_CL_ra(float *vec,float Velocity);
int ED-7255_t_move_CL_as(float *vec,float time);
int ED-7255_t_move_CL_ra(float *vec,float time);
```

9-45. ED-7255_t_move_JL_ra

1) Purpose

Perform linear motion using the target and the arrival time received based on the relative value.

2) Format

```
int ED-7255_t_move_JL_ra(float *target, float time);
```

3) Input

float target[8]	Target angle in relative value
time	Arrival time

4) Output

1 - Buffer input success
0 - Buffer input fail

5) Description

Convert a joint angle value of the relative position, received with reference to the present joint angle value of a robot, into Cartesian position x, y, and z values by using forward kinematics, and perform linear motion based on the received performance time.

6) Example

```
//Move a joint No. 3 by 10 degree from the present position.
float joint[6]={0,0,10,0,0,0};
float t = 5;
ED-7255_t_move_JL_ra(joint,v);
```

7) See Also

int ED-7255_v_move_J_as(float *target, float Velocity);

int ED-7255_t_move_J_as(float *target,float time);

int ED-7255_t_move_J_ra(float *target,float time);

int ED-7255_v_move_CL_as(float *vec,float Velocity);

int ED-7255_v_move_CL_ra(float *vec,float Velocity);

int ED-7255_t_move_CL_as(float *vec,float time);

int ED-7255_t_move_CL_ra(float *vec,float time);

PART 10 Maintenance

10-1. Troubleshooting Procedure

10-2. Tension Adjustment

10-3. Encoder Assembly

10-4. Motor Port

10-5. ERROR Code

PART 10

Maintenance

10-1. Troubleshooting Procedure

Failure of trouble has two classifications:

1) Complete system failure. A system does not operate at all.
2) Partially confined failure. Parts of system operate.

Generally, it is more difficult to solve the complete system failure because the origin of the failure cannot be found easily. Rather, it is easier to solve the partial failure because the failed part can be isolated from the operable part of a system. If it is not easy to repair the failed part, the failed part is often replaced.

The most difficult thing is to isolate the failed part. To find the failed part, generally, they replace the failed part with the same one and then monitor whether the problem disappears. On the other hand, they remove a part and then monitor that the problem disappears or other different reaction arises. If the reaction of a system does not become different even after removing a part, the removed part is likely to be failed part. Replacing or removing a part seems to be somewhat radical, but through this, the failed parts can be found rapidly and efficiently.

The first step of troubleshooting is to examine the most basic things as follows.

1) Examine the visible damage.
2) Is the power supplied to a system?
3) Is a system turned on?
4) Is there any missing part?
5) Is a fuse broken?
6) Is the heat excessively generated?
7) Are all contacts completely connected?

If a system does not operate after correcting or modifying the system, it is preferable that you should do the correction or modification in the reverse direction and monitor whether or not the system works again. Moving a system to other place is also available. In the case that a system is physically shocked, a chip may depart from a socket or part's connecting portions may be loosened. The departed or loosened parts may be settled by slightly pressing the chips mounted to the sockets after removing an upper cover of a controller. In addition, monitor whether connector cables are firmly fastened. Watch out whether the ambient condition has been probably changed.

Possibility of operator's mistake should not be excluded. Many problems result from the operator's mistake which may include mis-understanding the setup procedure or mis-understanding what the use of commands or their response is.

10-1-1. Power Problem

1) Power switch does not light up

If a power switch does not light up and a system is not supplied with electric power, look into a power cable and a fuse. A fuse can be found in a power module. Remove a power cable before removing or replacing the fuse. A new fuse to be replaced must have the same spec. as the old one. Be sure to find the cause to break the fuse before replacing the fuse.

2) Problem in switching power supply

If the voltage does not exist at the output terminals, in whole or partly, of a switching power supply, it is to be determined that this results from the switching power supply or a main board of a controller.

After confirming the system power is shut down, detach the switching power supply from a main board. Apply the power to the system, and check again the output voltage of the switching power supply. If the voltage does not exist at the output terminals, in whole or partly, get in touch with a vendor. (Never try to repair the switching power by yourselves.)

If the voltage is normally output, there may exist the short-circuit either in main board, in an I/O board, or in a teaching pendant. By separating the three boards one after another, you can find the short-circuited portion from them. If the short-circuited portion is found in a teaching pendant, check a cable plug. If it is found in a main board or I/O board, check that any metal body is put on the board.

3) Problem in motor power

Distinguish first where did the problem result, from a transformer or a main board. All remove a power cable connected from a transformer to a main board, a motor cable and an encoder cable of a front panel of a controller. Supply a system with the power. If the voltage across a center tap of the transformer and a (+) terminal of a bridge (located in front of the transformer) is about 24VDC, there is no problem in the transformer. If the voltage is not 24VDC, the transformer is not normal, and thus contacting a vendor is required.

If the problem is not found in the transformer, it is possible that any short-circuit exists in the main board. Check that any metal body exists on the board.

10-1-2. Motor Problem

1) Motor does not stop and continues up to limit point

In general, a motor deviates from a regular position because an encoder does not normally read values. Such situation is caused by

various reasons: the case that a connection cable between a robot and a controller is damaged, an encoder in itself is damaged, or an encoder recognition circuit is broken. Direct solution to these cases is to isolate the problem. If the problem is not solved by replacing the motor with a new one normally working, a controller may be regarded as the problem source.

If a motor continues moving to the limit point, the problem will be in a cable or an encoder for the relevant motor. If the motor erroneously operates even though the cable is replaced, the problem might have resulted from the encoder. A small crack can be found by visually examining a chip.

If it is determined that the encoder is damaged, generally, something outside the encoder might have broken the encoder. It is the most frequent case that the short-circuit occurs inside the encoder due to the cable damage. Check whether the cable is good.

2) Motor oscillates

Motor oscillation is caused by two reasons as follows.

① Unstable feedback loop, and
② Phase difference between the motor signal and the encoder signal

The second case may arise when a motor lead wire or an encoder connection cable is reversed when replacing the motor or encoder. To solve the problem, simply correct the motor lead wire to its regular polarity. If the motor oscillation disappears but the motor rotates in the wrong direction, correct the signal lines of both the motor and the encoder.

If the problem occurred due to the unstable feedback loop, it is necessary to correct the gain value. Referring to How to Use Teaching Pendant in PART 3, recover the gain value to the factory setting or

reset the system.

3) Encoder count does not work

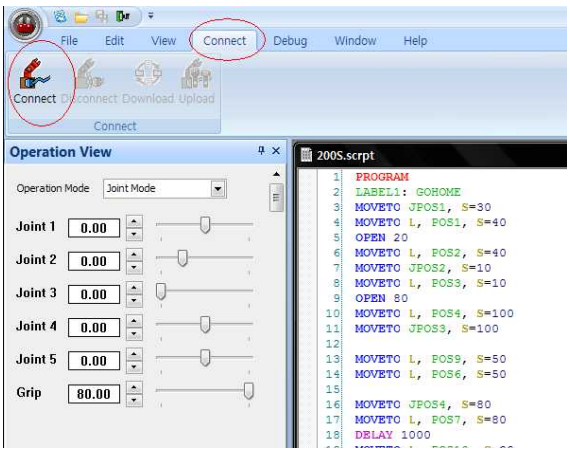
Generally, this problem occurs when an encoder is burnt by electric shock and an encoder line is short-circuited. If the encoder is burnt, the hot temperature may be felt by a hand. If you cannot feel hot from the encoder, replace the short-circuited encoder.

4) Motor cannot be selected or driven

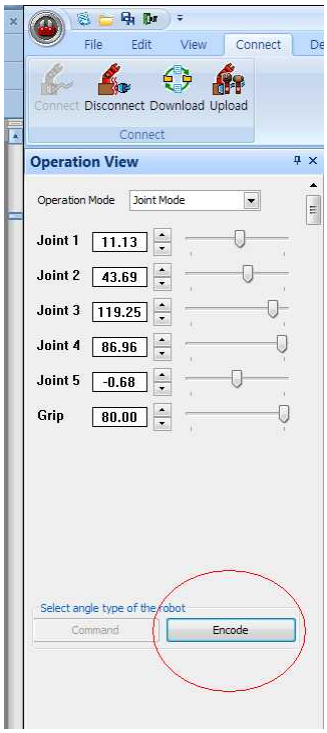
This symptom is generally not a mechanical or electrical failure. This problem means that a behavior requested to a motor is not consistent with a motor driving mode. For example, when a motor is in an idle mode, no effect will arise even though a motor selection key is pressed many times. Refer to How to Use Teaching Pendant in PART 3

5) How to check encoder failure by using simulator

After connecting a cross cable, make a robot be in the on-line condition by using 'Connect' of 'Connect Tab' in a simulator.



If it is turned to the on-line condition, the following 'Encode' button is activated, which means that the value of an encoder inserted in a robot is displayed on the simulator. Through this button, the status of the encoder can be checked.



10-1-3. Communication Problem

1) Problems in host communication

Check whether or not a cable is exactly connected to COM port being used by PC program. In addition, confirm, using a resistance meter, that the wiring of a 9-pin cable connected to COM port is assembled on a pin-to-pin basis.

If the status value is not displayed by PC program even after checking, try to exit the PC terminal program and reboot it.

2) Problem in teaching pendant

If there appears no display on a teaching pendant at the moment of powering on, it was not supplied with the power. This problem may arise from lack of system power or when a pendant cable is not connected. If it is certain that the power is supplied to a main board of a controller, remove a cable plug and check whether the connection condition is good or not.

If a starting message "ED-7255 Pendant" is being displayed even though plenty of time passed, communication with a controller is not performed even though the pendant is supplied with the power. If "*1.AUTO_RUN_MODE" is not shown, there are two possibilities of such failure. One is a bad cable connection and the other is a failed chip for transmission and reception, inside a controller. If there is no problem for a host computer to communicate, the chip for transmission and reception is not regarded as to be failed. Check whether or not the cable plug of the pendant is correctly connected.

10-1-4. I/O Point Problem

1) I/O point does not work

If an output point does not work even though the output point is turned ON by using the pendant, or if the input point is not recognized to be ON even though 24V is applied to the input point, an IC relay mounted at the output point or a photo-couple mounted at the input point is likely to be failed.

Select 'Connect' of 'Connect tab' in a simulator first, and then directly enter the values into I/O through 'Input and Output' of 'IO View' at the bottom of the simulator. I/O can be checked in this manner.

If all I/O points are failed, check that an I/O board is well connected

with a main board.

2) I/O point LED does not light up

If only one LED does not light up, it is to be regarded that the LED was burnt. This can be solved by replacing the LED. However, if all LEDs do not light up, the problem lies in a driver chip. Replace the driver chip.

10-2. Tension Adjustment

— A timing belt and a gear play an important role of activating each joint by receiving the dynamic force from a motor. If the tension is too strong or too weak, the error may arise in a motor or it may be hard on the motor.

1. When pressing the timing belt, it would be ideal that there is about 2-mm clearance.
2. How to adjust the tension of a timing belt used for pitching and rolling
   - Loosen both screws fastened to a link between joints No. 3 and 4.
   - Fasten again the screws while pressing a shaft which provides the tension.
3. The tension of an elbow can be adjusted in the same manner as above.
4. Check other screws and nuts, and firmly fasten them.
5. Be sure to check as above before use.

Appendix 1
Mechanical Analysis Theory

1. Introduction

1-2. Robot as a Machine

1-2-1. Work Space

1-3. Forward Kinematics

1-3-1. Algebraic Solution

1-3-2. Geometric Solution

1-3-3. Matrix-Based Representation of Robot Kinematics

1-3-4. Homogeneous Transformation

1-4. Denavit-Hartenberg

1-4-1. Denavit-Hartenberg Representation for Robot

1-4-2. Denavit-Hartenberg Representation

1-4-3. Denavit-Hartenberg Representation for Homogeneous Transformation Matrix

Appendix 1

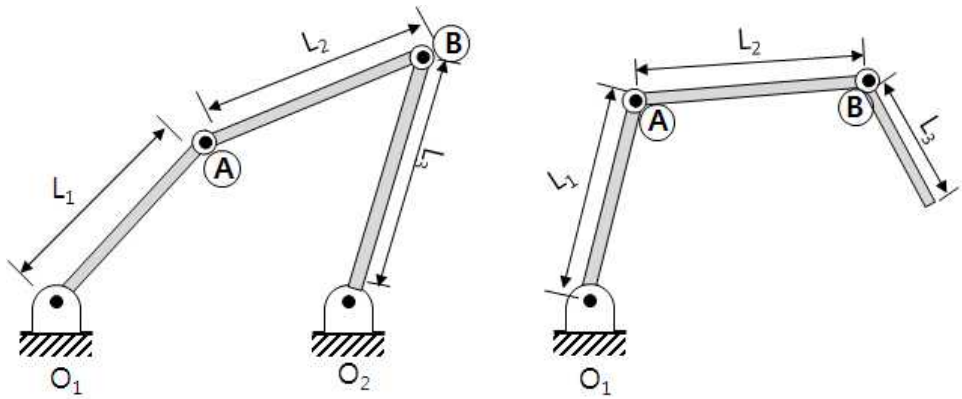
Kinematics : Position Analysis

1-1. Introduction

In this part, we will discuss about kinematics and inverse kinematics. A robot can determine the robot's end-effector, i.e., a robot hand by using kinematics, only if it knows all the variables of joints. In addition, to locate the robot's end point to a specified position and an orientation angle, we can calculate all of the joint variables by using inverse kinematics. First, we will introduce a method of representing objects, positions, the orientation angle, and the motion by using the algebraic solution and the geometric solution. Second, we will study about kinematics and inverse kinematics for various robot shapes in a Cartesian coordinate system. Finally, we will derive formulae for kinematics and inverse kinematics for all possible shapes of a robot, by using Denavit-Hartenberg representation.

1-2. Robot as a Machine

A manipulator of multiple DOF (Degree of Freedom) has a mechanism of 3-dimensional open loop, which is connected as a chain, and a mechanism of 3-dimensional closed loop, in which the one end is not connected.



(a) 1-DOF Closed Loop 4-Joint Mechanism

(b) 3-DOF Open Loop 3-Joint Mechanism

Figure 1-1. Closed Loop 4-Joint Mechanism & Open Loop 3-Joint Mechanism

As shown in Figure 6-1 (a), if a link AB is deformed, this affects a link O_2B, but this deformation will move all the continuous composing mechanism connected without feedback for an open loop system like a robot. As a result, in an open loop system, parameters for all joints and links must be continually measured, and the end point of a system must be always known. Thus, they must be monitored for mechanical position. Let's compare the differences through a vector equation.

$$\overline{O_1A} + \overline{AB} = \overline{O_1O_2} + \overline{O_1B}$$ Figure 7-1 (a) Vector Equation

$$\overline{O_1A} + \overline{AB} + \overline{BC} = \overline{O_1C}$$ Figure 7-1 (b) Vector Equation

In Figure 6-1 (a), if a link AB has the deformation, it will move along a link O_2B. However, in Figure 7-1 (b), although AB deformed, the deformation cannot be known unless O_1C is measured by other method.

To solve such problem in an open loop robot, the position of a hand must be continually measured by e.g., a camera.

1-2-1. Work Space

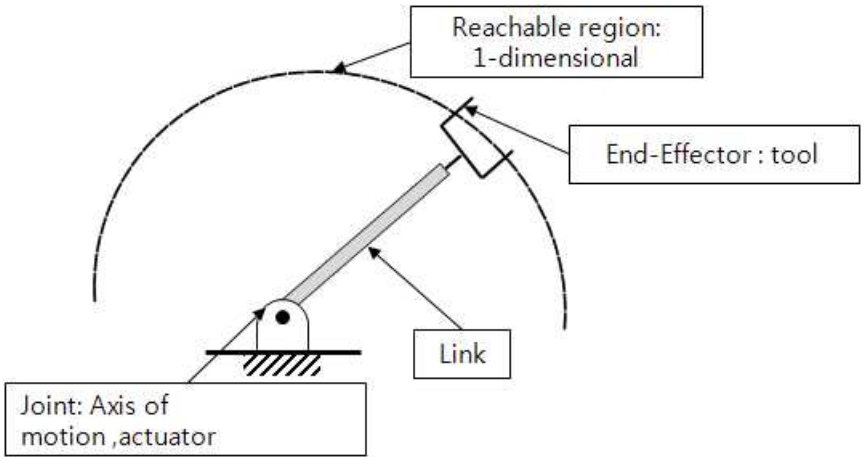


Figure 1-2. 1-DOF Robot

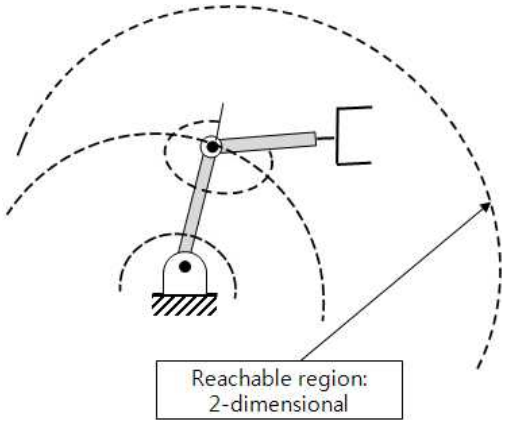


Figure 1-3. 2-DOF Robot

1-3. Forward Kinematics

The purpose of the forward kinematics is to express the position and orientation of an end-effector as the joint variables and the link length in a Cartesian space.

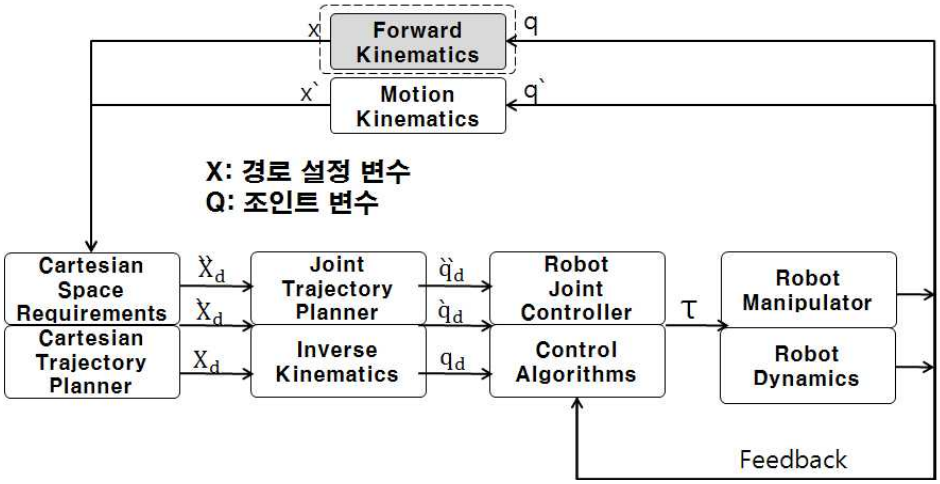


Figure1-4. Whole block diagram of robot for explanation of forward kinematics, inverse kinematics, and dynamics

X: Path Setting Variable
Q: Joint Variable

Figure1-4 shows a role of forward kinematics in the whole robot block diagram. Mechanical information of a robot, referred to by a robot joint controller, and X, Y, and Z coordinate values, calculated by the forward kinematics, are sent to Cartesian Space Requirements to calculate Trajectory Plan. The trajectory plan will be dealt with in Inverse Kinematics Chapter.

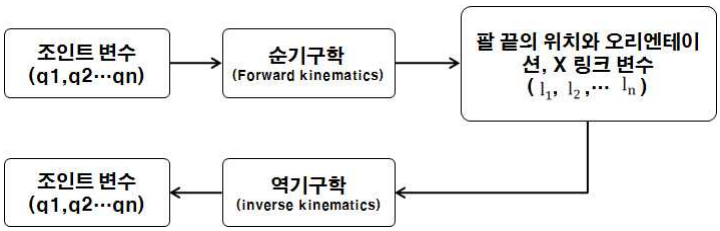


Figure 1-5. Relationship between forward kinematics and inverse kinematics

조인트변수: Joint Variable
팔 끝의 위치와 오리엔테이션:
Position and Orientation of End-Effector
X 링크 변수: X Link Variable

Figure 1-5 shows the forward and inverse kinematics which are divided by the variables to be found. The forward kinematics represents the end-effector position(X) in a Cartesian space, when the angle(q) is given in a joint space. The inverse kinematics is for calculation of the angle in a joint space, when the end-effector position is given in a Cartesian space.

If each joint of a robot forms a certain angle(q), the end-effector position(X_p) can be represented with a function of the angle and the link length(l),

$$X_p = F(q,l)$$

The forward kinematics is a function of each joint and the link length of a robot. Solutions to the forward kinematics roughly includes algebraic solution and geometric solution.

1-3-1. Algebraic Solution

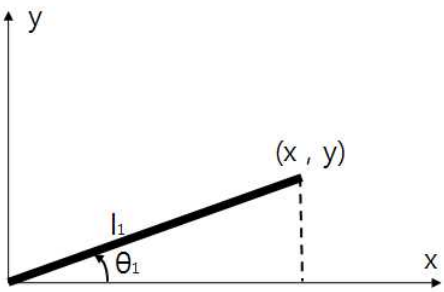


Figure 1-6. Simple one-link manipulator case

Figure 1-6 shows a question of finding the position (x,y) in a Cartesian space, provided that a base joint θ_1 and a link length l_1 are given in 2-dimensional space.

It can be simply solved using a trigonometric function as follows.

$$x = f_x(\theta_1) = l_1\cos\theta_1$$

$$y = f_y(\theta_1) = l_1\sin\theta_1$$

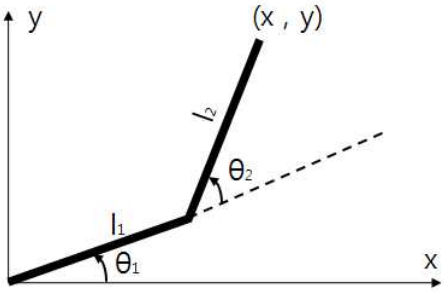


Figure 7-7. Simple two-link manipulator case

Figure 1-7 shows a question of finding the position (x,y) in a Cartesian space, provided that link lengths l_1 and l_2 and base joints θ_1 and θ_2 at the end of l_1 are given in 2-dimensional space.

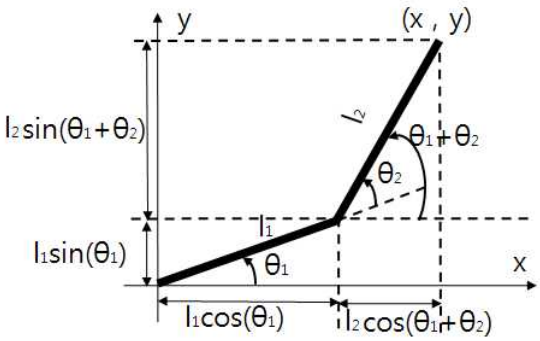


Figure 1-8 Analysis diagram of Figure 7-7

It can be solved as follows, based upon the analysis in Figure 1-8.

$$x = f_x(\theta_1,\theta_2) = l_1\cos\theta_1 + l_2\cos(\theta_1+\theta_2)$$

$$y = f_y(\theta_1,\theta_2) = l_1\sin\theta_1 + l_2\sin(\theta_1+\theta_2)$$

Using such algebraic approach, the coordinate of an end-effector can be found.

However, generally, DOF of a humanoid robot is 13 DOF minimum

and DOF of an industrial manipulator is basically 6 DOF. Of course, DOF could not be any more different depending on robot models.

Figure1-9. Android type humanoid robot Albert HUBO

Figure 1-9 shows a humanoid type Albert Hubo by KAIST.

The following table lists various DOFs of Hubo.

Head	Torso	Arm	Hand	Leg	Total
28/Face 3/ Neck	1/Torso (Yaw)	3/Shoulder 1/Elbow	5/Hand 2/Wrist	3/Hip 1/Knee 2/Ankle	
31 DOF	1 DOF	8 DOF	14 DOF	12DOF	66 DOF

The Robot Hubo has as many as 66 joint DOFs, and thus, representing it algebraically is beyond our capacity. Accordingly, generally in robot mechanics, the robot coordinate (x,y,z) in 3-dimensional space is solved by the geometric approach.

1-3-2. Geometric Solution

The matrix is used to represent points, vectors, coordinate systems, transitional rotation, and transformation, as well as mechanical elements

and objects in a certain coordinate system.

1) Representation of point in space

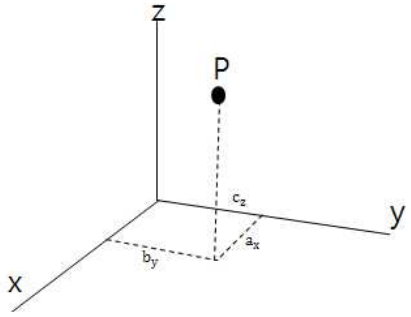


Figure 1-10 Representation of a point in space

In Figure 1-10, a point P in a space can be represented with three coordinates for the reference coordinate system.

$$P = a_x\hat{i} + b_y\hat{j} + c_z\hat{k}$$

Here, a_x, b_y, and c_z are three coordinates of a point represented in the reference coordinate system.

2) Representation of vector in space

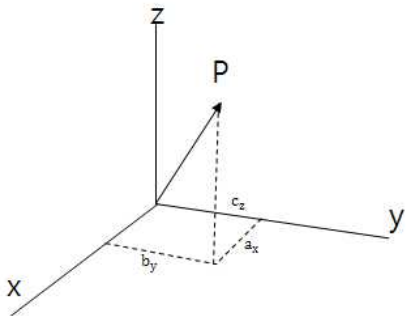


Figure 1-11. Representation of a point in space

A vector can be represented with three coordinates at its start and end point. If a vector starts at point A and ends at point B, it can be represented as:

$$\overline{P}_{ab} = (B_x - A_x)\hat{i} + (B_y - A_y)\hat{j} + (B_z - A_z)\hat{k}$$

Especially, if the vector starts at the origin as in Figure 7-11,

$$\mathrm{P} = a_x\hat{i} + b_y\hat{j} + c_z\hat{k} \ .$$

Here, a_x, b_y, and c_z are three elements of a vector in the reference coordinate system. If fact, the point P in the previous section can be represented as a vector connected to P and be represented with the three elements of a vector.

The three elements of a vector can be expressed by the matrix as the following expression. This will be used over this textbook to represent all the kinematical elements.

$$\overline{P} = \begin{bmatrix} a_x \\ b_y \\ c_z \end{bmatrix}$$

This representation can be slightly modified to the form including w, if x, y, and z can be divided by w, the size factor, and thereby a_x, b_y, and c_z can be found. Therefore, the following vector representation becomes possible.

$$\overline{P} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}, \text{ wherein } a_x = \frac{x}{w},\ b_y = \frac{y}{w}, \text{ etc.}$$

Variable w may be a certain value and as it varies, the total size, or magnitude, of a vector may vary. This is similar to magnifying a picture in computer graphics. Since the value w varies, the vector size also varies. If w is greater than a unit vector, all vector elements become greater, and if w is smaller than a unit vector, all vector elements become smaller. This is used in computer graphics in which a picture size is to be varied.

If w is a unit vector, size of all elements does not vary, but in the case of w=0, a_x, b_y, and c_z have infinity value. In this case, x, y, and z are also represented as a vector of infinite length, as a_x, b_y, and c_z are. However, a vector involves the directional representation, which means that a directional vector can be represented by the size element, w=0. Here, the size is not very important, but the direction is represent by the vector elements.

Exercise 1.1

Assuming that a vector is represented as $\overline{P} = 3\hat{i} + 5\hat{j} + 2\hat{k}$, represent this vector in the matrix form.

Ⓐ Size factor is 2.

Ⓑ Represent the direction by a unit vector.

☞ [Solution] Since the vector size factor is 2, it can be represented by the following matrix.

$$\overline{P} = \begin{bmatrix} 6 \\ 10 \\ 4 \\ 2 \end{bmatrix}$$

Because of condition Ⓑ, if dividing the size factor by each element to represent a directional vector, w=0 is satisfied.

That is, $\overline{P} = \begin{bmatrix} 3 \\ 5 \\ 2 \\ 0 \end{bmatrix}$.

Therefore, a unit vector will be obtained by finding the total size and equating the new size. For this, each vector element must be divided by the square root of the sum of the square of each the three elements

$$\lambda = \sqrt{p_x^2 + p_y^2 + p_z^2} = 6.16$$

Thus, $p_x = \frac{3}{6.16} = 0.487$, $p_y = \frac{5}{6.16} = 0.811$, and $p_z = \frac{2}{6.16} = 0.324$.

It is representing by the matrix as $\overline{p_{unit}} = \begin{bmatrix} 0.487 \\ 0.811 \\ 0.324 \\ 0 \end{bmatrix}$.

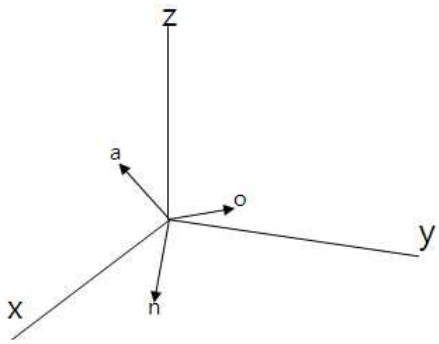


Figure 1-12. Coordinate representation at origin of reference coordinate system

3) Representation of a coordinate system at origin of a fixed reference coordinate system

Figure 7-12 shows that a coordinate system centering the origin of a reference coordinate system can be represented with three vectors. These vectors are named as a unit vector $\overline{n}$, $\overline{o}$, and $\overline{a}$ for normal, orientation, and approach vector, respectively. Each of the unit vectors is represented with three elements of a reference coordinate system. F coordinated system is represented three vectors in the form of matrix, as follows.

$$F = \begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix}$$

4) Representation of a coordinate system in a fixed reference coordinate system

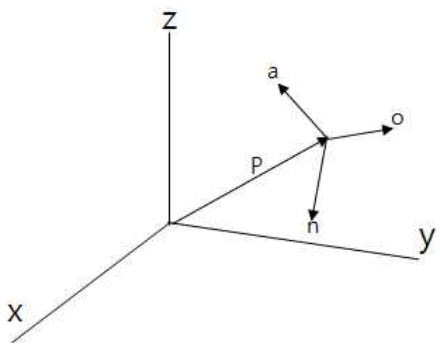


Figure 1-13. Representation of different coordinate system from one coordinate system

If a certain coordinate system is not at the origin or even though it is at the origin, the origin position of an arbitrary coordinate system with regard to a reference coordinate system should be represented. To do so, as shown in Figure 1-13, a vector, for the origin of an arbitrary

coordinate system and for the origin of a reference coordinate system which depicts the position of a coordinate system, should be represented. This vector is represented through the elements related to a reference coordinate system. Thus, the coordinate system can be represented by three vectors which depict a directional unit vector. The fourth vector which depicts its position can be represented as follows.

$$F = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

First three vectors in a fourth row are directional vectors that w=0, which indicate the direction of three unit vectors in $\overline{n}$, $\overline{o}$, and $\overline{a}$ coordinate system. A fourth vector that w=1 expresses the origin position of an arbitrary coordinate system with regard to a reference coordinate system. Unlike the unit vectors, the length of the vector P is very important, and thus the size factor can be united.

Exercise 1.2

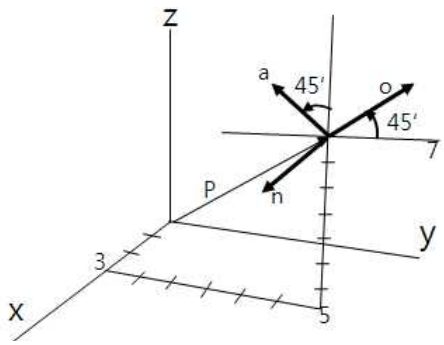


Figure-1-14. Representation example of spatial coordinate system

Regarding to Figure 1-14, a coordinate system F is located at (3,5,7), n axis is parallel to x-axis, o axis is declined by 45` with regard to y-axis, and a axis is declined by 45` with regard to z-axis. This coordinate system can be represented as follows.

☞ [Solution]

$$F = \begin{bmatrix} 1 & 0 & 0 & 3 \\ 0 & 0.707 & -0.707 & 5 \\ 0 & 0.707 & 0.707 & 7 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5) Representation for rigid body

The position of an object can be represented by attaching an coordinate system which indicates the coordinate in a space. Since this coordinate system is permanently attached, the position and orientation related to this coordinate system can be always known. Resultantly, the position and orientation of an object related to any fixed coordinate system, as far as the coordinate system can be represented in a space. In addition to the origin of a coordinate system in a space, the orientation component for a reference coordinate system can be also represented by the matrix.

$$F_{object} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

One point in a space has three DOFs moving along each of the three reference axes. However, a rigid body in a space not only moves along X-, Y-, and Z-axis but also has 6 DOFs rotating with regard to three axes. To define an object in a space, 6 information for representation of the orientation for three axes, the origin position in a reference coordinate system, etc. is necessary. However, as shown in the above F_{object} equation, total 12 information is given, which includes 9 information on the orientation and 3 information on the position. Obviously, 6 constraint equations is necessary in it. This constraint equation is found by the known characteristic of the coordinate system but has not been used yet.

- **Three unit vectors $\bar{n}$, $\bar{o}$, and $\bar{a}$ are perpendicular to one another.**
- **The length of each unit vector is '1'.**

The procedure can be transformed into 6 constraint equations which will be explained later.

① $\bar{n} \cdot \bar{o} = 0$ (Inner product of the vector $\bar{n}$ and the vector $\bar{o}$ is 0.)

② $\bar{n} \cdot \bar{a} = 0$

③ $\bar{a} \cdot \bar{o} = 0$

④ $\bar{n} = 1$ (Vector size must be 1.)

⑤ $\bar{o} = 1$

⑥ $\bar{a} = 1$

Resultingly, values representing one coordinate system in the matrix must be of above; otherwise, it cannot be a proper coordinate system. The above constraint equation can be replaced with the outer product of three different vectors, as follows.

$$\bar{n} \times \bar{o} = \bar{a}$$

6) Understanding spatial coordinate through matrix

In this section, we will explain how to represent the spatial vector by the matrix. This section is an important part because this is later related to representing the homogeneous transformation and D-H table with the matrix.

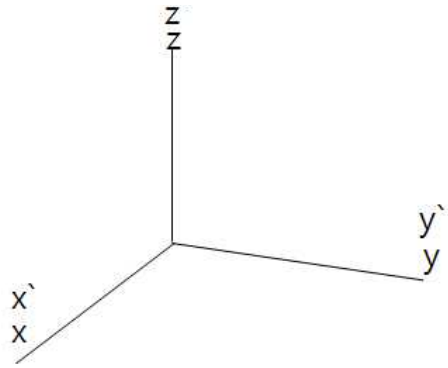
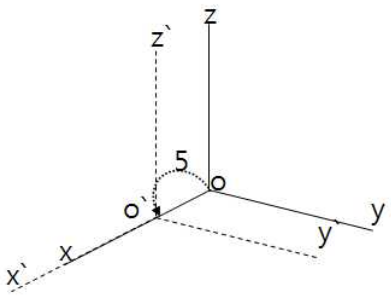


Figure1-15. A coordinate system identical to reference coordinate system

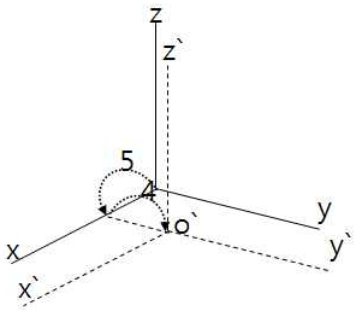
In Figure 1-15, a certain coordinate system identical to a reference coordinate system can be represented by the matrix as follows.

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

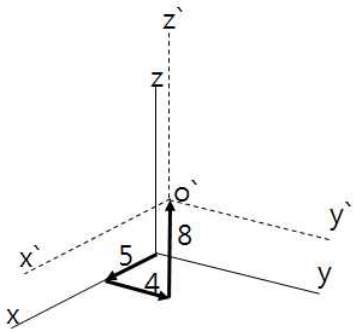
Parallel translation along x-axis results in:



$$T = \begin{bmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$T = \begin{bmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 4 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



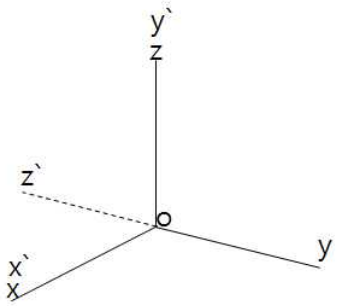
$$T = \begin{bmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 4 \\ 0 & 0 & 1 & 8 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Therefore, matrix F = $\begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$,

where p_x is the distance away from the origin along x-axis, p_y is the distance away from the origin along y-axis, and p_z is the distance away from the origin along z-axis.

That is, rows 1, 2, and 3 in the fourth column mean the distance value from the origin.

Next, vector rotation will be explained.



$$R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The above shows a vector rotated by -90 degree about x-axis. The matrix R can also be found by substituting the rotational matrix. However, we will introduce this later in order to know here the

meaning of the matrix per position.

When analyzing the vector rotation, X` becomes coincident with X, y` becomes coincident with z, and z` becomes coincident with -y by rotating the vector by -90 degree centering around x-axis.

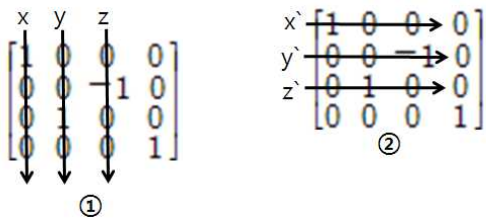


Figure1-15. Meaning per position of rotational matrix

Figure 1-15 ① shows the translation direction for a reference coordinate system, Figure 1-15 ② shows the translation direction for a certain coordinate system.

In a certain coordinate system, since x` is coincident with x and the first column of Figure ① is x, which corresponds to the 1st, 2nd, and 3rd rows of the first column, the 1st row 1st column becomes 1 and the remaining becomes 0.

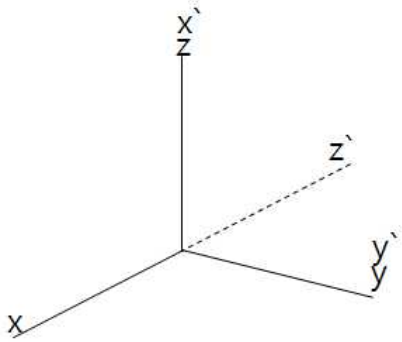
In a certain coordinate system, since y` is coincident with z of a reference coordinate system, it corresponds to the 2nd column in Figure ①, and since z` is positioned at the 2nd column 3rd row, this position becomes 1 and the remaining becomes 0.

Similarly, since z` is coincident with -y of reference coordinate system, the 3rd column 2nd row becomes -1 and the remaining becomes 0.

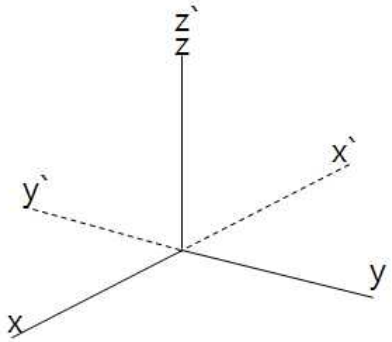
So far, we have discussed what each element of the matrix means. From now, we will explain about completing the origin moving and vector rotation into one matrix.

Exercise 1.3

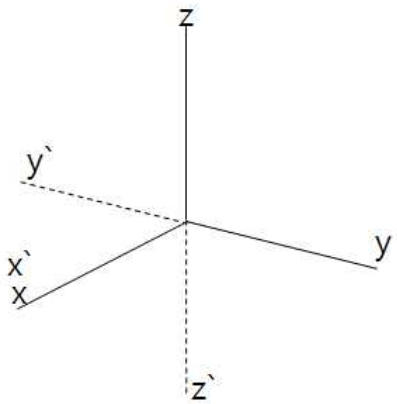
Represent the following vectors by 4x4 matrix.



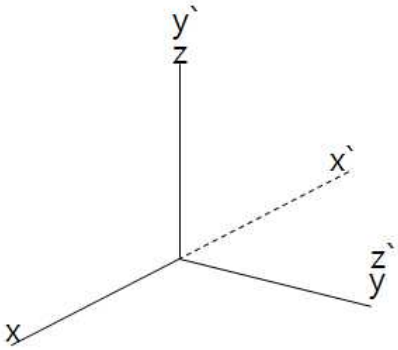
(a)



(b)



(c)



(d)

☞ [Solution]

(a) Rotate 90 degree about y-axis

$$\begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(b) Rotate 180 or -180 degree about z-axis

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

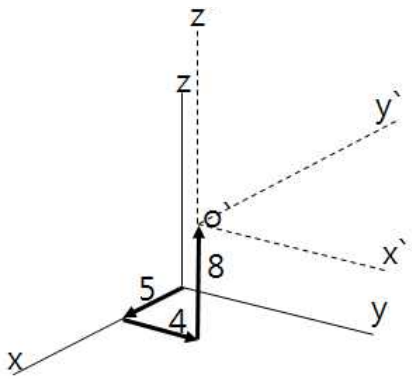
(c) Rotate 180 or -180 degree about x-axis

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(d) Rotate -90 degree about x-axis after rotating by 180 or -180 degree about z-axis

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The following shows a vector rotated after moving the origin in a reference coordinate system.



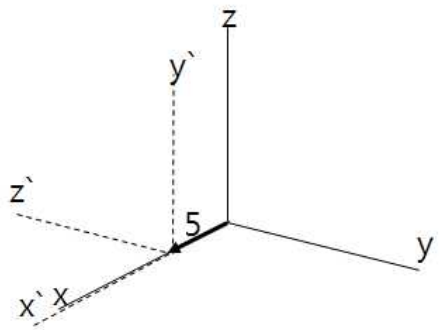
x`, y`, and z` are represented as the matrix.

$$T = \begin{bmatrix} 0 & -1 & 0 & 5 \\ 1 & 0 & 0 & 4 \\ 0 & 0 & 1 & 8 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

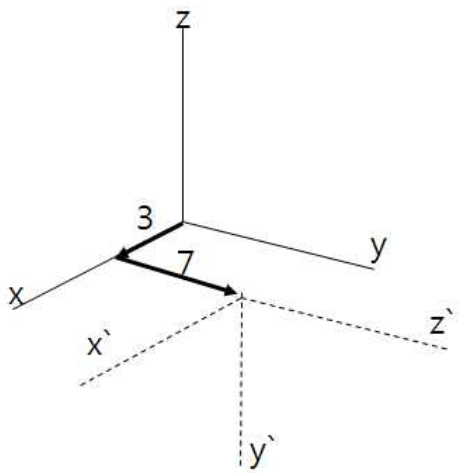
The above matrix will be understood without difficulty.

Exercise 1.4

Represent the following vectors by 4x4 matrix.



(a)



(b)

☞ [Solution]

(a) Translate 5 along x-axis and then rotate -90 degree about x-axis.

$$T = \begin{bmatrix} 1 & 0 & 0 & 5 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(b) Translate 3 along x-axis and 7 along y-axis, and then rotate 90 degree about x-axis.

$$T = \begin{bmatrix} 1 & 0 & 0 & 3 \\ 0 & 0 & 1 & 7 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

1-3-3. Matrix-Based Representation of Robot Kinematics

In the previous section, we have studied the robot kinematics using algebraic solution. Let's discuss the matrix-based representation of the robot kinematics using geometric solution.

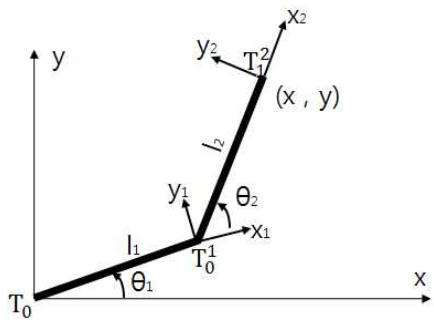


Figure 1-16. Vector of 2-joint link

·T_0 = Base coordinate

·T_0^1 = Position where l_1 and l_2 meet, from T_0.

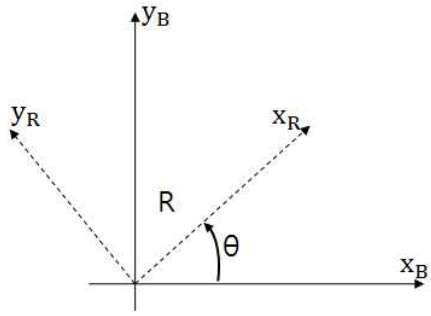
·T_1^2 = End position of l_2 from T_0^1.

Before explanation of coordinate transformation in Figure 7-16, we will introduce the matrix combinations for easy understanding the explanation.

- 2 Dimension -

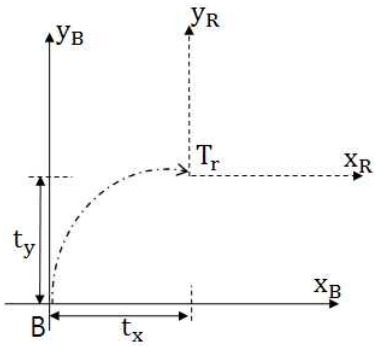
① The matrix rotated θ about the origin (Rotation matrix)

$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



② The matrix when the origin is translated by t_x, t_y (Translation matrix)

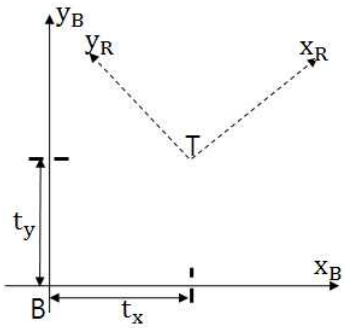
$$T_r = \begin{bmatrix} 1 & 1 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$



③ Combined transformation matrix

$$T = R \bullet T_r$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c\theta & -s\theta & t_x \cdot c\theta - t_y \cdot s\theta \\ s\theta & c\theta & t_x \cdot s\theta - t_y \cdot c\theta \\ 0 & 0 & 1 \end{bmatrix}$$



Notation abbreviation

$\cos\theta = c\theta$, $\sin\theta = s\theta$, $\tan\theta = t\theta$

-3 Dimension-

① The matrix rotated θ about x-axis

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

② The matrix rotated θ about y-axis

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

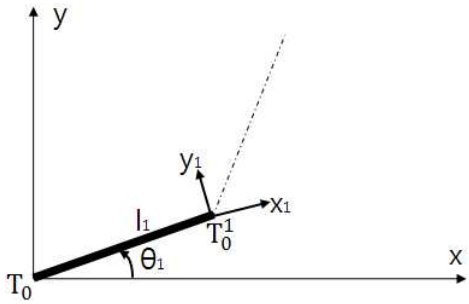
③ The matrix rotated θ about z-axis

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From the foregoing, preparations completed. Let's find the Cartesian coordinate T_0^2 of an end-effector of a robot by using the matrix. First find T_0^1.

$$T_0^1 = \mathrm{R}(\theta_1) \bullet T_x(l_1)$$

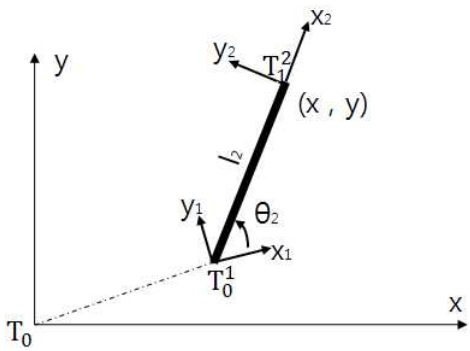
$$= \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & l_1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & l_1\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & l_1\sin\theta_1 \\ 0 & 0 & 1 \end{bmatrix}$$



Next, find T_1^2.

$$T_1^2 = \mathrm{R}(\theta_2) \bullet T_x(l_2)$$

$$= \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & l_2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & l_2\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & l_2\sin\theta_2 \\ 0 & 0 & 1 \end{bmatrix}$$



$$\therefore \quad T_0^2 = T_0^1 \bullet T_1^2$$

$$= \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & l_1\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & l_1\sin\theta_1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & l_2\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & l_2\sin\theta_2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c(\theta_1+\theta_2) & -s(\theta_1+\theta_2) & l_2 c(\theta_1+\theta_2)+l_1 c\theta_1 \\ s(\theta_1+\theta_2) & c(\theta_1+\theta_2) & l_2 s(\theta_1+\theta_2)+l_1 s\theta_1 \\ 0 & 0 & 1 \end{bmatrix}$$

Interpreting the meaning of the resulting matrix,

at the origin coordinate T_0, an end-effector vector T_0^2 is rotated as

much as $\begin{bmatrix} c(\theta_1+\theta_2) \\ s(\theta_1+\theta_2) \\ 0 \end{bmatrix}$ in the x direction and is rotated as much as $\begin{bmatrix} -s(\theta_1+\theta_2) \\ c(\theta_1+\theta_2) \\ 0 \end{bmatrix}$ in the y direction, and is translated as much as $\begin{bmatrix} l_2 c(\theta_1+\theta_2)+l_1 c\theta_1 \\ l_2 s(\theta_1+\theta_2)+l_1 s\theta_1 \\ 1 \end{bmatrix}$ from the origin.

1-3-4. Homogeneous Transformation

A homogeneous transformation is 4x4 matrix, is formed by four submatrices, and can represents the rotation and the translation simultaneously.

$$T = \begin{array}{cc} \text{Rotation} & \text{Translation} \\ \left[\begin{array}{c|c} R_{3\times3} & d_{3\times1} \\ \hline f_{1\times3} & 1 \end{array}\right] & \\ \text{perspective} & \text{Scale} \end{array} \qquad T = \left[\begin{array}{ccc|c} n_x & o_x & a_x & d_x \\ n_y & o_y & a_y & d_y \\ n_z & o_z & a_z & d_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

* Scale and Perspective are sometimes used to realize a graphic program, where Perspective is set to 0 and Scale is set to 1, generally. Scale means the size of a rigid body. If Scale is greater than 1, make it to 1.

$$Trans_{x,a} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, Trans_{y,b} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, Trans_{z,c} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot_{x,a} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, Rot_{y,b} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, Rot_{z,c} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ si\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Let's see one exercise about the homogeneous transformation.

If the matrix is rotated 'a' degree about a chord x-axis, it is

translated by 'b' along a chord x-axis, it is translated by 'd' along a chord z-axis, and finally it is rotated θ degree about a chord z-axis, the homogeneous transformation matrix T is as follows.

$$T = \mathrm{Rot}(z,\theta)\mathrm{Trans}(0,0,d)\mathrm{Trans}(b,0,0)\mathrm{Rot}(x,a)$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & b \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & 0 & b \\ \cos a\sin\theta & \cos a\cos\theta & -\sin\theta & -\sin ad \\ \sin a\sin\theta & \sin a\cos\theta & \cos a & \cos ad \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

※ Rot(rotational axis, rotating angle), Trans(x-axis moving amount, y-axis moving amount, z-axis moving amount)

Exercise 1.5

Using given four conditions, form a homogeneous transformation matrix.

(1) Rotate α about x-axis.
(2) Translate a along x-axis.
(3) Translate d along z-axis.
(4) Rotate θ about z-axis.

☞ **[Solution]**

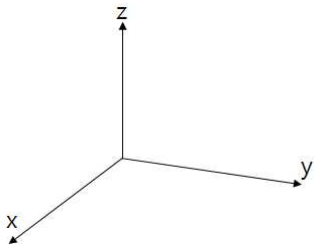
$$T = \mathrm{Rot}(z,\theta)\mathrm{Trans}(0,0,d)\mathrm{Trans}(a,0,0)\mathrm{Rot}(x,\alpha)$$

$$= \begin{bmatrix} c\theta & -s\theta & 0 & 0 \\ s\theta & c\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & ca & -sa & 0 \\ 0 & sa & ca & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c\theta & -s\theta c\alpha & s\theta s\alpha & c\theta a \\ s\theta & c\theta c\alpha & -c\theta s\alpha & s\theta a \\ 0 & s\alpha & c\alpha & d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Exercise 1.6

Referring to the drawings, form a homogeneous transformation matrix of Joint n+1.



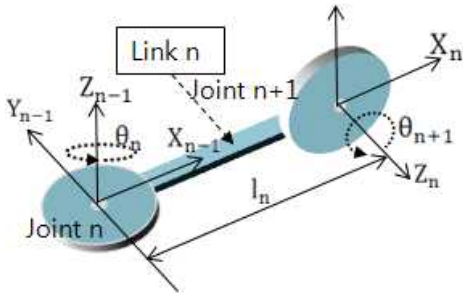


Figure 7-17.

- Joint n is the first joint.
- Joint n was rotated 180 degree in a reference coordinate system.
- z_n axis was rotated -90 degree from z_{n+1}. (It was rotated -90 degree about x_{n-1} axis of Joint n.)
- The distance between Joint n and Joint n+1 is l_n.
- The origin of Joint n+1 is located on an extended line along x-axis of Joint n.

☞[Solution]

T = Trans(l_n,0,0)Rot(x,-90)Rot(z,180)

Find the matrix representation of T by yourselves.

Exercise 1.7

Referring to the following drawing, form the homogeneous transformation equation.

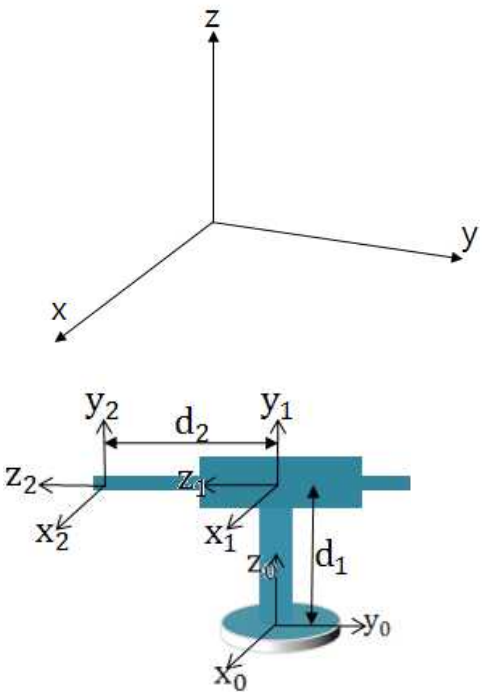
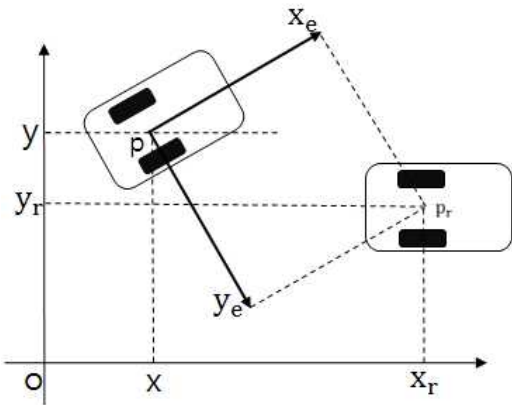


Figure 7-18.

☞[Solution]

$$T = \mathrm{Trans}(d_2,0,0)\mathrm{Trans}(0,0,d_1)\mathrm{Rot}(x,-90)$$

Exercise 1.8



This is kinematics for a mobile robot. Kinematics when moving from the present position p to the next position p_r is described.

The coordinate at p is p = $\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$, and the coordinate at the desired reference coordinate p_r is p_r = $\begin{bmatrix} x_r \\ y_r \\ \theta_r \end{bmatrix}$. In the above drawing, show how the following equation is derived when to find a new coordinate p_e = $\begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix}$. Draw each component in the following coordinate.

$$p_e = \begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r - x \\ y_r - y \\ \theta_r - \theta \end{bmatrix}$$

1-4. Denavit-Hartenberg

1-4-1. Denavit-Hartenberg Representation for Robot

In 1955, Denavit and Hartenberg presented a paper in ASME Journal of Applied Mechanics. This was later used to express and design a robot, and also used to derive the equation of motion.ons. This concept has been used as a standard for expression and modelling of robots, and is a must-know basic theory in learning the robots. Regardless of its sequence and complexity, Denavit-Hartenberg (D-H) representation model can be used for any robot link and joint of any robot forms. This not only can easily represent the transformation for a Cartesian, cylindrical, spherical, Euler, or PRY coordinate system, but is applicable to all rotational articulated robot, SCARA robot, or modelling any combination of joints and links. However faster a robot is directly modeled by the earlier learned representation method, this Denavit-Hartenberg representation has had the additional advantage, as many techniques, such as Jacobian calculation, force analysis, etc., has been being developed.

Assume that a robot is made with consecutive joints and links. The joints may include prismatic joints or revolute joints, which exist on any plane at any order. The length of the link can be changed to any length including 0, be bent, or be twisted, and can exist on any plane.

Therefore, a certain set of joint and link can be created into a robot which is easy to express and convenient to model.

To do this, a general procedure which transforms one joint into the next joint is defined, after assigning a reference coordinate system to each joint. If combining all transformations from a base to a first joint, from the first joint to a second joint, and to a final joint, the total homogeneous transformation matrices of a robot can be obtained.

In this section, for assignment of a reference coordinate system to each joint, we define a general procedure based on Denavit-Hartenberg

representation. We also define how the transformation between two consecutive coordinate systems is performed. Finally we explain about the total homogeneous transformation matrices of a robot.

1-4-2. Denavit-Hartenberg Representation

(1) α_i - Twist angle between z_i axis and z_{i-1} axis in i coordinate system

If two links are connected, the axis of rotation is made. As in Figure 7-19, raise a right-hand thumb along x_i direction and bend, or rotate, the remaining four fingers so that z_i axis and z_{i-1} axis are brought into coincidence. This rotation angle is $-\alpha_i$. Similarly, if the four fingers are rotated so that z_{i-1} axis and z_i axis are brought into coincidence, this rotation angle is α_i. Assuming all the axes are perpendicular to one another and rotating x_i axis centering about a right-hand thumb, if the order that the four fingers are bent is z_{i-1} to z_i, α_i is 90 degree, while if it is z_i to z_{i-1}, α_i is -90 degree.

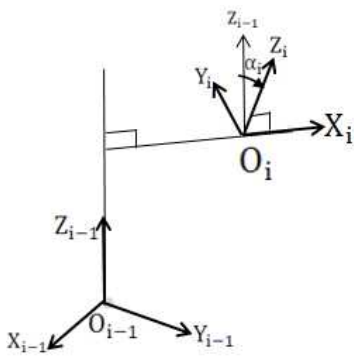


Figure 7-19. Twist angle

(2) θ_i - Angle of rotation around z_{i-1} axis needed to bring x_{i-1} axis

and x_i axis into coincidence. Angle that x_{i-1} rotates counter-clockwise around z_{i-1} axis is denoted as the positive, and its clockwise rotation is denoted as the negative. Generally θ_i is the joint variable as the angle of each joint. Putting a right-hand thumb along z_{i-1} axis and rotating x_{i-1} axis so that x_{i-1} axis x_i axis are brought into coincidence, if it is rotated in the direction of the four bent fingers, this angle is θ_i; otherwise, $-\theta_i$.

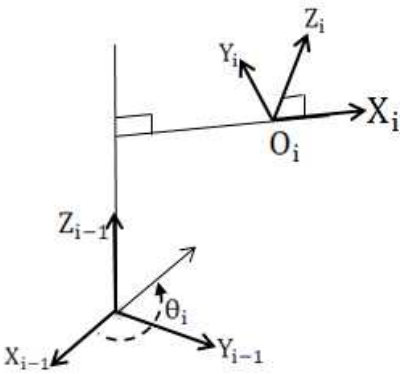


Figure 7-20. Rotated angle

(3) d_i - Shortest distance between X axes

If moving a point along x_i axis in the minus direction, it meets z_{i-1} axis. At this time, d_i is the displacement from the origin o_{i-1} of z_{i-1} axis to the met point. This direction is defined to be positive. That is, the positive displacement is defined for the direction that z_{i-1} axis increases. In general, d_i means the shortest distance between x_i axis and x_{i-1} axis, both of which do not meet each other. If x_i axis and x_{i-1} axis intersect, d_i=0.

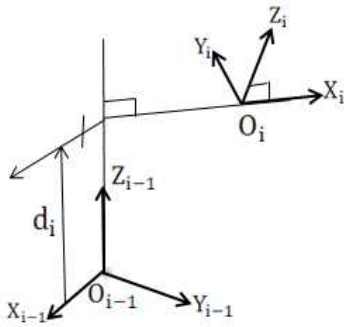


Figure 7-21. Distance between X axes

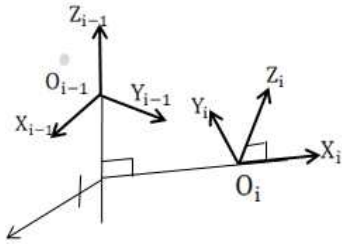


Figure 7-22. Distance between X axes

(4) a_i - Shortest distance between Z axes

If moving a point along x_i axis in the minus direction, it meets z_{i-1} axis. a_i is the displacement from the met point to the origin o_i of x_i axis. In general, a_i means the shortest distance between z_i axis and z_{i-1} axis, both of which do not meet each other. If the axes meet each other, a_i=0.

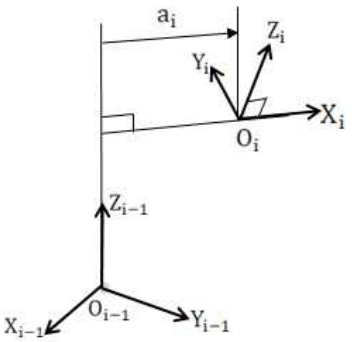


Figure 7-23. Distance between Z axes

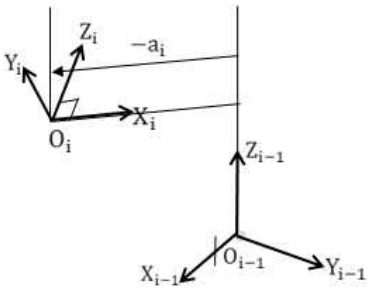


Figure 7-24. Distance between Z axes

As a result, let's represent the four D-H parameters for two adjacent joints.

In the drawing, the relationship between two coordinates i and i+1 which are connected with a link i is expressed with the four D-H parameters.

Pay attention to the coordinates that each parameter is denoted. Thus, in Figure 7-23, the origin of i+1 coordinate becomes the same by translating $-a_i$ along $x_i$axis and translating $-d_i$ along z_{i-1} axis. Next, i and i+1 coordinates become the same by rotating $-\alpha_i$ about x_i axis and rotating $-\theta_i$ about z_{i-1} axis.

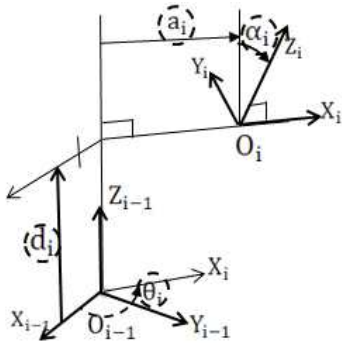


Figure 7-25. Relationship between two coordinates

The following drawing shows D-H parameters between two adjacent coordinates.

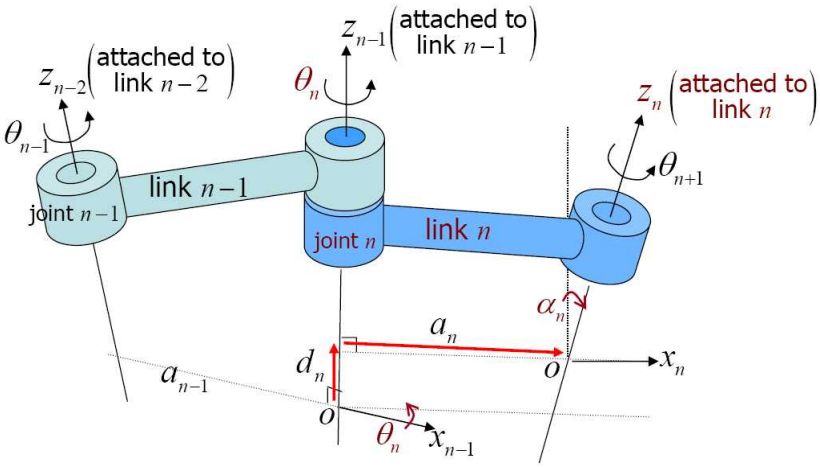


Figure 7-26. Relationship between two links

1-4-3. Denavit-Hartenberg Representation for Homogeneous Transformation Matrix

To represent the position of the endmost coordinate in Figure 7-23 with reference to a base coordinate, set the coordinated at each joint according to a regular procedure, as follows.

(1) First establish the base coordinate system X_0, Y_0, Z_0.

(2) Establish in advance z-axis for each joint with the axis of motion.

(3) Locate the axis origin o_i at the intersection of z_i axis and z_{i-1} axis or at the intersection of common normal between the z_i and z_{i-1} axes and z_i axis.

(4) Establish x_i axis in normal to the plane z_{i-1} axis and z_i axis form. That is, establish x_i axis perpendicularly to both z_{i-1} axis and z_i axis respectively, and in the direction away from z_{i-1} axis.

(5) Establish y_i axis based on x_i and z_i axis, according to the right-hand law.

(6) To bring x_{i-1} axis and x_i axis into coincidence, rotate z_{i-1} axis

and get θ_i (θ_i is positive).

(7) Find d_i, displacement from x_{i-1} to x_i.

(8) Find a_i, if z_{i-1} and z_i do not meet on the extended line.

(9) α_i is the angle necessary to bring z_{i-1} axis and z_i axis into coincidence when rotating z_{i-1} axis about x_i axis.

(10) Make the table by find D-H parameters for each link.

Link i	θ_i	α_i	a_i	d_i
1				
2				
.				
i				

(11) Find a transformation matrix for each joint, using the parameters obtained above.

i) Revolute joint

In Figure 7-26, in order to bring the coordinates $i-1$ and i into coincidence,

① Rotate about z_{i-1} axis an angle of θ_i to align x_i axis with x_{i-1} axis in parallel.

② Translate along z_{i-1} axis a distance of d_i to bring x_i axis and x_{i-1} axis into coincidence.

③ Translate along x_i axis a distance of a_i to bring the origins of the two coordinate into coincidence.

④ Rotate about x_i axis an angle of α_i to bring the two coordinates into coincidence.

In addition, regarding the axis of rotation as the center, the transformation matrix found by rotating α_i about x-axis, translate a_i along x-axis, rotate θ_i and translate d_i about and along z-axis is as follows.

In the above, the order of ① and ② may be exchanged.

$$A_{i-1}^{i} = T_{z,\theta}\,T_{z,d}\,T_{x,a}\,T_{x,\alpha} = T_{z,d}\,T_{z,\theta}\,T_{x,a}\,T_{x,\alpha}$$

$$= \begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & c\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\alpha_i & -s\alpha_i & 0 \\ 0 & s\alpha_i & c\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c\theta_i & -s\theta_i c\alpha_i & s\theta_i s\alpha_i & c\theta_i a_i \\ s\theta_i & c\theta_i c\alpha_i & -c\theta_i s\alpha_i & s\theta_i a_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

If substituting the parameters found in the D-H parameter table, the transformation matrix can be obtained. In a rotational joint, the value of θ_i varies as its rotation and the remaining parameters become constant.

ii) Prismatic joint

For the prismatic joint, only d_i becomes a variable and the remaining becomes constant, however a_i=0.

$$A_{i-1}^{i} = T_{z,\theta}\,T_{z,d}\,T_{x,\alpha}$$

$$= \begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & c\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\alpha_i & -s\alpha_i & 0 \\ 0 & s\alpha_i & c\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c\theta_i & -s\theta_i c\alpha_i & s\theta_i s\alpha_i & 0 \\ s\theta_i & c\theta_i c\alpha_i & -c\theta_i s\alpha_i & 0 \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For the prismatic joint, prismatic motion changes according to d_i, and the remaining parameters become constant. Therefore, parameters α_i and θ_i are always the constant.

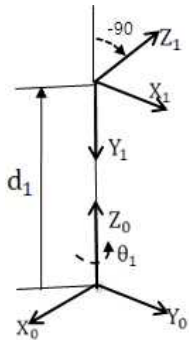
(12) Find the whole homogeneous transformation matrices.

The whole transformation matrices can be obtained by sequentially multiplying the transformation matrix for each joint.

$$T_0^n = A_0^1 A_1^2 \ \ldots \ A_{n-1}^n$$

Exercise 1.9

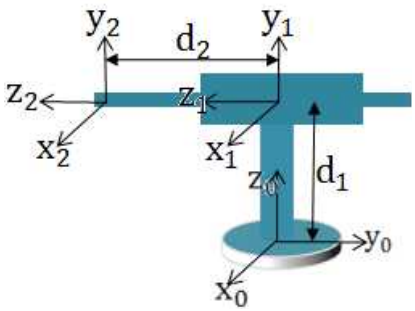
Find D-H parameters in the following drawing. A link type is the revolute joint, whose length is d_1.



Joint	θ_i	α_i	d_i	a_i
1	θ_1	−90	d_1	0

Exercise 1.10

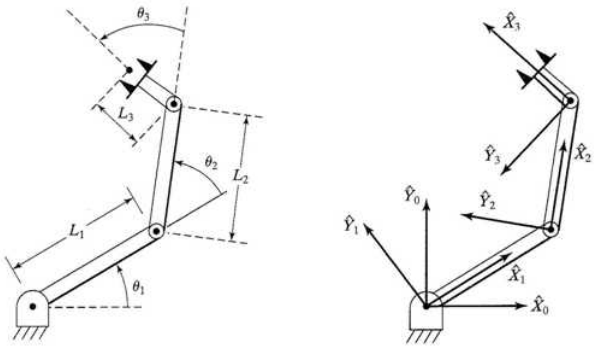
In the drawing, two joints are formed on a basis of the prismatic joint manner. Find D-H parameters. Real parameters are d_1 and d_2.



Joint	θ_i	α_i	d_i	a_i
1	0	90	d_1	0
2	0	0	d_2	0

Exercise 1.11

In the drawings, complete D-H parameters and find the coordinate x,y of an end-effector.



☞[Solution]

Joint	θ_i	α_i	d_i	a_i
1	θ_1	0	0	0
2	θ_2	0	0	L_1
3	θ_3	0	0	L_2

Since all joints are of the revolute joint, set variables in a basic transformation matrix. The following result can be obtained.

$$\begin{bmatrix} c\theta_i & -s\theta_i c\alpha_i & s\theta_i s\alpha_i & c\theta_i a_i \\ s\theta_i & c\theta_i c\alpha_i & -c\theta_i s\alpha_i & s\theta_i a_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

※ In the above equation, the following abbreviations are used.

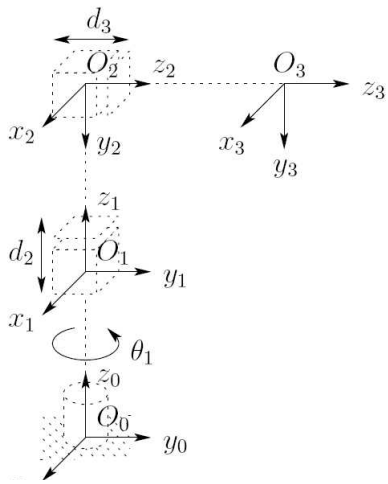
$c\theta_i = \cos\theta_i$, $s\theta_i = \sin\theta_i$, $c\theta_{23} = \cos(\theta_2+\theta_3)$, $s\theta_{23} = \sin(\theta_2+\theta_3)$,

$c\theta_{23} \neq \cos\theta_2+\cos\theta_3$, $s\theta_{23} \neq \sin\theta_2+\sin\theta_3$

Exercise 1.12

In the drawings, complete D-H parameters and find the coordinate

x,y,z of an end-effector.



☞[Solution]

Joint	θ_i	α_i	d_i	a_i
1	θ_1	0	d_1	0
2	0	-90	d_2	0
3	0	0	d_3	0

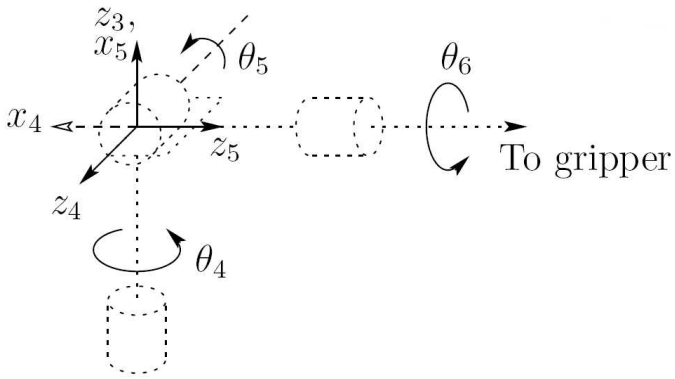
$$T_0^3 = T_0^1\, T_1^2\, T_2^3$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_2 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & d_2 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c_1 & 0 & -s_1 & -s_1 d_3 \\ s_1 & 0 & c_1 & c_1 d_3 \\ 0 & -1 & 0 & d_1 + d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Exercise 1.13

The drawing shows a gripper of a manipulator. Complete a D-H table and find the homogeneous matrix.



☞[Solution]

Joint	θ_i	α_i	d_i	a_i
4	θ_4	−90	0	0
5	θ_5	90	0	0
6	θ_6	0	d_6	0

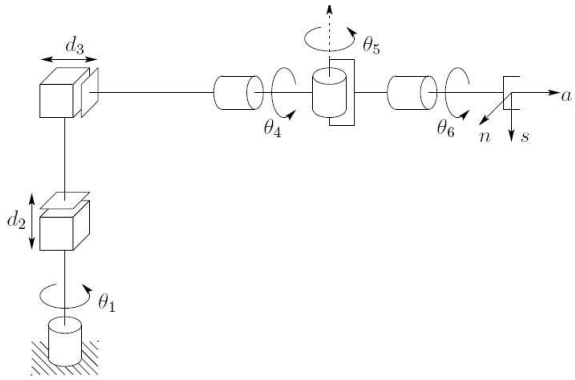
$$T_3^6 = T_3^4\, T_4^5\, T_5^6$$

$$= \begin{bmatrix} c_6 & -s_6 & 0 & 0 \\ s_6 & c_6 & 0 & 0 \\ 0 & 0 & 1 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_5 & 0 & s_5 & 0 \\ s_5 & 0 & -c_5 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_4 & 0 & -s_4 & 0 \\ s_4 & 0 & c_4 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c_4c_5c_6 - s_4s_6 & -c_4c_5s_6 - s_4c_6 & c_4s_5 & c_4s_5d_6 \\ s_4c_5c_6 + c_4s_6 & -s_4c_5s_6 + c_4c_6 & s_4s_5 & s_4s_5d_6 \\ -s_5c_6 & s_5s_6 & c_5 & c_5d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Exercise 1.14

The following drawing combines Exercise 11 and Exercise 12. Find the homogeneous matrix.



☞[Solution]

$$T_0^6 = T_3^6\, T_0^3$$

$$\begin{bmatrix} c_1 & 0 & -s_1 & -s_1 d_1 \\ s_1 & 0 & c_1 & c_1 d_3 \\ 0 & -1 & 0 & d_1 + d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_4c_5c_6 - s_4s_6 & -c_4c_5s_6 - s_4c_6 & c_4s_5 & c_4s_5d_6 \\ s_4c_5c_6 + c_4s_6 & -s_4c_5s_6 + c_4c_6 & s_4s_5 & s_4s_5d_6 \\ -s_5c_6 & s_5c_6 & c_5 & c_5d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} r_{11} & r_{12} & r_{13} & d_x \\ r_{21} & r_{22} & r_{23} & d_y \\ r_{31} & r_{32} & r_{33} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned}
r_{11} &= c_1c_4c_5c_6 - c_1s_4s_6 + s_1s_5c_6 \\
r_{21} &= s_1c_4c_5c_6 - s_1s_4s_6 - c_1s_5c_6 \\
r_{31} &= -s_4c_5c_6 - c_4s_6 \\
r_{12} &= -c_1c_4c_5s_6 - c_1s_4c_6 - s_1s_5c_6 \\
r_{22} &= -s_1c_4c_5s_6 - s_1s_4s_6 + c_1s_5c_6 \\
r_{32} &= s_4c_5c_6 - c_4c_6 \\
r_{13} &= c_1c_4s_5 - s_1c_5 \\
r_{23} &= s_1c_4s_5 + c_1c_5 \\
r_{33} &= -s_4s_5 \\
d_x &= c_1c_4s_5d_6 - s_1c_5d_6 - s_1d_3 \\
d_y &= s_1c_4s_5d_6 + c_1c_5d_6 + c_1d_3 \\
d_z &= -s_4s_5d_6 + d_1 + d_2.
\end{aligned}$$

Exercise 1.15

The drawing is schematic of Stanford Manipulator in Figure 7-25. Complete the D-H table and find the homogeneous matrix.

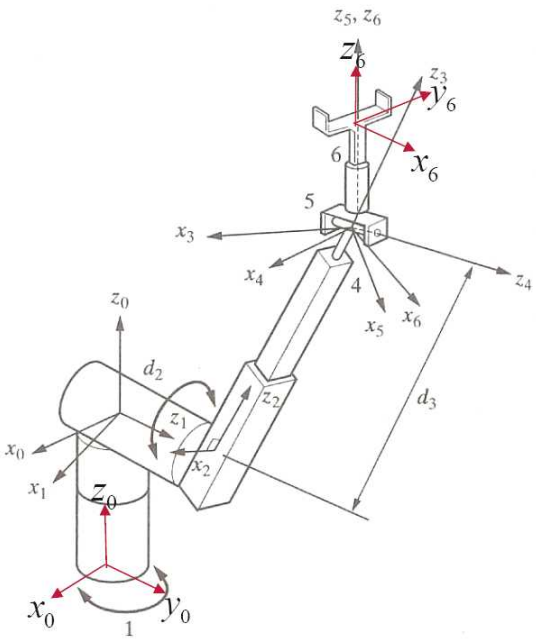
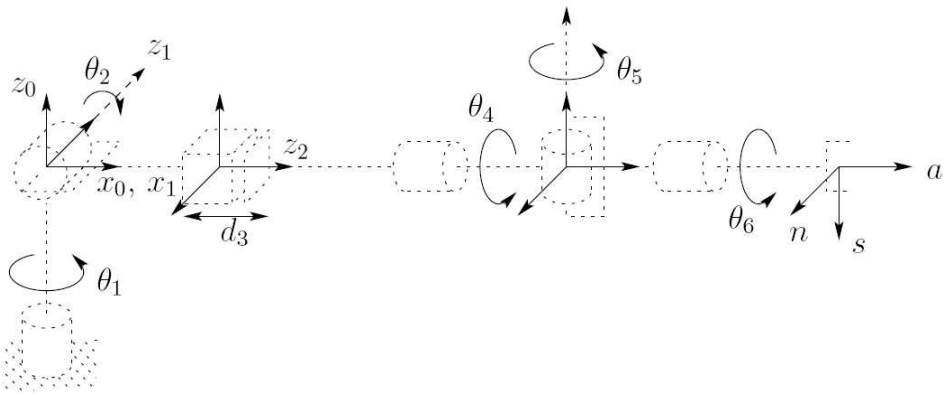


Figure 1-27. Stanford Arm



☞[Solution]

Joint	θ_i	α_i	d_i	a_i
1	*	−90	0	0
2	*	90	d_2	0
3	0	0	*	0
4	*	−90	0	0
5	*	90	0	0
6	*	0	d_6	0

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & d_x \\ r_{21} & r_{22} & r_{23} & d_y \\ r_{31} & r_{32} & r_{33} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

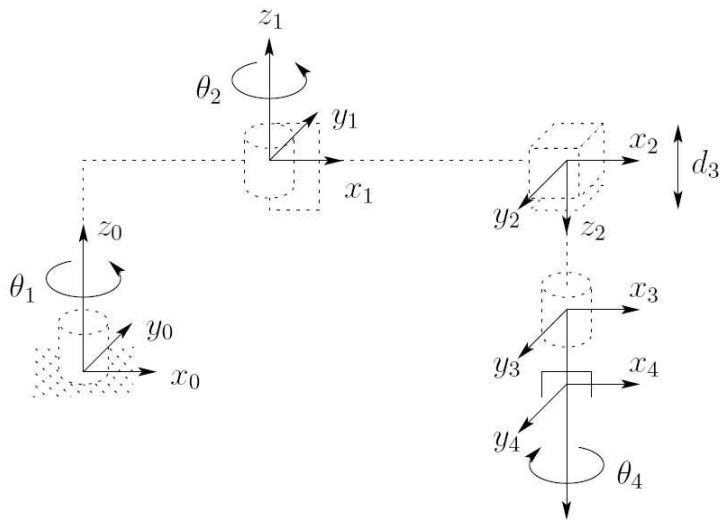
$$\begin{aligned}
r_{11} &= c_1[c_2(c_4c_5c_6 - s_4s_6) - s_2s_5c_6] - d_2(s_4c_5c_6 + c_4s_6) \\
r_{21} &= s_1[c_2(c_4c_5c_6 - s_4s_6) - s_2s_5c_6] + c_1(s_4c_5c_6 + c_4s_6) \\
r_{31} &= -s_2(c_4c_5c_6 - s_4s_6) - c_2s_5c_6 \\
r_{12} &= c_1[-c_2(c_4c_5s_6 + s_4c_6) + s_2s_5s_6] - s_1(-s_4c_5s_6 + c_4c_6) \\
r_{22} &= -s_1[-c_2(c_4c_5s_6 + s_4c_6) + s_2s_5s_6] + c_1(-s_4c_5s_6 + c_4c_6) \\
r_{32} &= s_2(c_4c_5s_6 + s_4c_6) + c_2s_5s_6 \\
r_{13} &= c_1(c_2c_4s_5 + s_2c_5) - s_1s_4s_5 \\
r_{23} &= s_1(c_2c_4s_5 + s_2c_5) + c_1s_4s_5 \\
r_{33} &= -s_2c_4s_5 + c_2c_5 \\
d_x &= c_1s_2d_3 - s_1d_2 + +d_6(c_1c_2c_4s_5 + c_1c_5s_2 - s_1s_4s_5) \\
d_y &= s_1s_2d_3 + c_1d_2 + d_6(c_1s_4s_5 + c_2c_4s_1s_5 + c_5s_1s_2) \\
d_z &= c_2d_3 + d_6(c_2c_5 - c_4s_2s_5).
\end{aligned}$$

Exercise 1.16

The drawing is a schematic of SCARA Manipulator. Complete the D-H table and find the homogeneous matrix.

※ Horizontal Articulated Robot: Often called as SCARA (Selectively Compliant Arm for Robotic Assembly) which means plurality of parallel joints are connected. Two axes are parallel to the gravity direction, and three axes only moves along Z direction (up-down motion). This is manufactured as four-axis type in which three axes can move up-down and rotate.

Advantages: Motion on a horizontal plane perpendicular to two axes of rotation (X,Y) is very fast. Thus, this exhibits the higher productivity, since the cycle time for SCARA becomes shorter because the initial motion on a plane perpendicular to the two axes is faster than that of a Cartesian coordinate robot. In addition, since it has high maximum payload, it can carry a heavier object.



☞[Solution]

Joint	θ_i	α_i	d_i	a_i
1	*	0	0	a_1
2	*	180	0	a_2
3	0	0	*	0
4	*	0	d_4	0

$$\begin{bmatrix} c_{12}c_4 + s_{12}s_4 & -c_{12}s_4 + s_{12}c_4 & 0 & a_1c_1 + a_2c_{12} \\ s_{12}c_4 - c_{12}s_4 & -s_{12}s_4 - c_{12}c_4 & 0 & a_1s_1 + a_2s_{12} \\ 0 & 0 & -1 & -d_3 - d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Exercise 1.17

The drawing is an image of PUMA560. Complete the D-H table and find the homogeneous matrix.

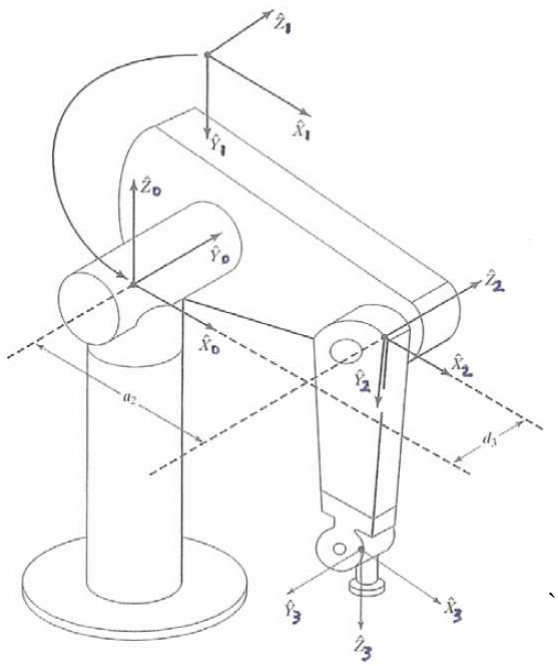


Figure 1-28. PUMA 560

Appendix 2

Inverse Kinematics

2-1. Solution-Based Classification of Inverse Kinematics

2-1-1. Analytical Algorithm vs. Numerical Algorithm

2-1-2. Parallel Mechanism vs. Serial Mechanism

2-2. Geometric method

2-3. Algebraic method

2-3-1. Example of 2-Axis Robot

2-3-2. Example of 3-Axis Robot

2-4. Differential Equations Method

2-4-1. Differential Relationship of Robot Kinematics

2-4-2. Physical Meaning of Jacobian Matrix

2-5. Singularities

2-6. Easiness of Motion

2-7. Source Code for Inverse Kinematics

ED Corporation

Appendix 2

Inverse Kinematics

Inverse kinematics, in contrast to the forward kinematics, determines the joint variables from the Cartesian variables (position and orientation). In the forward kinematics, the position of an end-effector of a robot could be determined for a given joint angle. However, there may be many angles of each joint found from a given position of the end-effector.

- Problem of finding the joint variables able to make the target position and orientation of an end-effector that an operator wants.
- Process of finding a solution to how to control each joint of a manipulator to make the desired pose and position.
- The opposite concept of forward kinematics for determining the position and orientation of an end-effector of a manipulator, which are made from the specific joint variable.

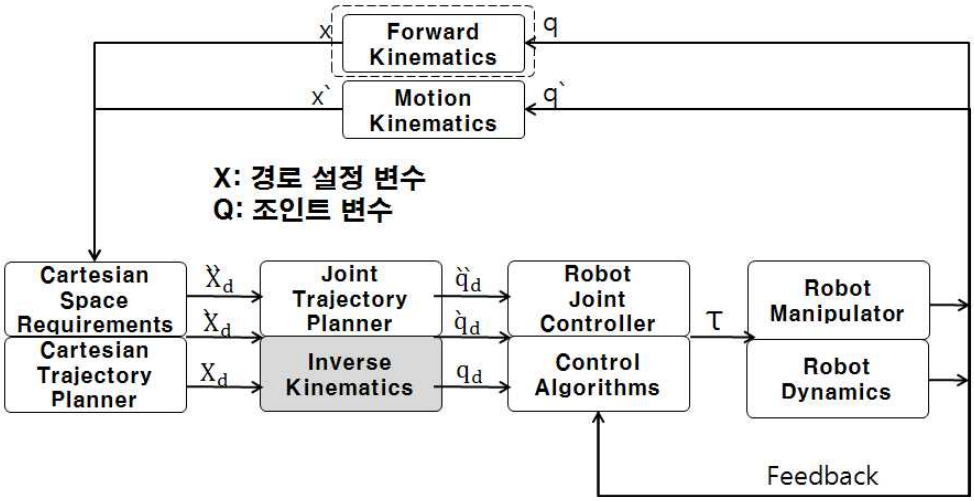


Figure 2-1. Whole block diagram of robot

X: Path Setting Variable
Q: Joint Variable

2-1. Solution-Based Classification of Inverse Kinematics

2-1-1. Analytical Algorithm vs. Numerical Algorithm

1) Analytical Algorithm

- All possible solutions can be found. Thus, the analytical algorithm is said to be Complete.
- It selects the desired solution from many.
- Calculation time is faster than Numerical Algorithm.
- More reliable than Numerical. Thus, errors do not arise at all.
- It is further subdivided into Closed-Form and Algebraic Elimination.
- In a Closed-Form method, the solutions can be directly expressed as a set of closed-form equations.
- The solutions of Closed-Form can only be obtained for a special case or a 6-DOF system.
- Algebraic Elimination expresses the solutions as multivariable polynomial

equations.

2) Numerical Algorithm

- It cannot find all solutions, but it finds only a single solution by judging the convergence.
- Calculation time is slower than Analytical.
- Not reliable than Analytical, because there is the convergence error.
- The advantage is it can be used in cases where the systems is ill-posed. Analytical method cannot find the solutions in ill-posed conditions.

→ **Three popular methods**

1. Newton-Raphson algorithm for solving systems of nonlinear equations
2. Converting the inverse kinematics problem into a differential equation.
3. Recasting the inverse kinematics problem into a nonlinear optimization problem

2-1-2. Parallel Mechanism vs. Serial Mechanism

Parallel Mechanism : Only one inverse kinematics solution is obtained.
Many forward kinematics solutions are obtained.

Serial Mechanism : Many inverse kinematics solutions are obtained.
Only one forward kinematics solution is obtained.

If the solution of the kinematics is properly determined, an end-effector can be moved to the desired position in the orientation.

2-2. Geometric Method

Literally, the joint angle is determined using geometry. Finding the angle of each joint may be easier for a simple-structured robot having

few links, because simpler geometry can be used to find it. Be given an example for a one-link robot as follows.

In the drawing, find θ for a given a Cartesian position P_x, P_y.

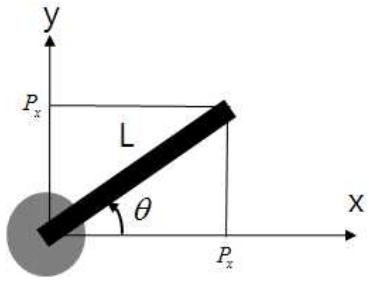


Figure 2-2. One-link robot

$$P_x = L\cos\theta \qquad \theta = \cos^{-1}\frac{P_x}{L}$$

$$P_y = L\sin\theta \qquad \theta = \sin^{-1}\frac{P_y}{L}$$

As is shown in Figure 8-2, the angle θ can be found by two ways in the above equations. In general, however, it is found by the value of $\tan^{-1}$, of which the value exists exclusively at all the upper limits, instead of $\cos^{-1}$ and $\sin^{-1}$. $\cos^{-1}$ has '0' value either for +90 or −90 degree. That is, since cos(q) = cos(-q), the sign of the angle cannot be determined. Similarly, $\sin^{-1}$ has '0' value either for 0 or 180 degree, and thus if it is divided by sin(q) nearby such an angle value, it will has the incorrect value. Therefore, atan function, in which the sign and the magnitude are defined at each the upper limit and which can define the values at all the upper limits, is used to find the angle of each joint.

$$\tan\theta = \frac{P_y}{P_x} \qquad \theta = \tan^{-1}\frac{P_y}{P_x}$$

Exercise 2.1

Consider the inverse kinematics for the following 2-link robot.

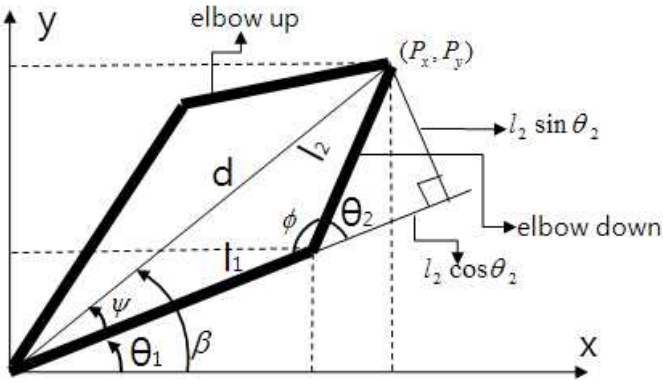


Figure 2-3. 2-link robot

For a given Cartesian position P_x, P_y, the distance from the origin, d, can be determined by

$$d = \sqrt{P_x^2 + P_y^2}.$$

By the cosine law, an equation for the two lines and the angle between them is

$$d^2 = l_1^2 + l_2^2 - 2l_1 l_2 \cos\phi.$$

Rearranging the above equation in terms of cosΦ,

$$\cos\phi = \frac{l_1^2 + l_2^2 - d^2}{2l_1 l_2}.$$

Since $\phi = 180 - \theta_2$,

$$\cos(180\text{-}\theta_2) = -\cos\theta_2 = \frac{l_1^2 + l_2^2 - d^2}{2l_1 l_2}$$

$$\cos\theta_2 = \frac{d^2 - l_1^2 - l_2^2}{2l_1 l_2}.$$

From $\sin^2\theta + \cos^2\theta = 1$,

$$\sin\theta_2 = \pm\sqrt{(1-\cos\theta_2)}.$$

From the above equation,

$$\tan\theta_2 = \pm\frac{\sqrt{(1-\cos\theta_2)}}{\cos\theta_2}.$$

θ_2 can be determined by

$$\theta_2 = \mathrm{atan}(\pm\frac{\sqrt{(1-\cos\theta_2)}}{\cos\theta_2}).$$

If, in Figure 8-3, d, l_1, and l_2 are known, ψ can be found.

$$l_2^2 = d^2 + l_2^2 - 2\mathrm{d}l_1\cos\psi$$

Rearranging the equation in terms of $\cos\psi$,

$$\cos\psi = \frac{l_1^2 + d^2 - l_2^2}{2l_1 d} = \mathrm{A}.$$

Find θ_1 the angle of joint 1. Since an angle between the planes is an acute angle, ψ can be determined directly by the cosine function.

$$\psi = \cos^{-1}\mathrm{A}$$

Since an angle θ_1 is the difference between the angle ψ and the

angle β, we can obtain the followings. Pay attention that the solution may differ from the positions of a robot arm.

$$\beta = \tan^{-1}\left(\frac{P_y}{P_x}\right)$$

$$\theta_1 = \beta \pm \psi \quad (+\text{: elbow up, } -\text{: elbow down})$$

Exercise 2.2

Show in Exercise 2.2 that the following is valid.

$$\tan\psi = \frac{l_2 + \sin\theta_2}{l_1 + l_2\cos\theta_2}$$

2-3. Algebraic Method

2-3-1. Example of 2-Axis Robot

The angles in a simple robot can be easily determined by the geometric method, but it is not very easy to determine the angle in a multi-link robot by the geometric method. To this type of robot, the whole result from the transformation matrix may be used to calculate it. Find the angle of each joint from the following forward kinematics equation for a 2-axis robot.

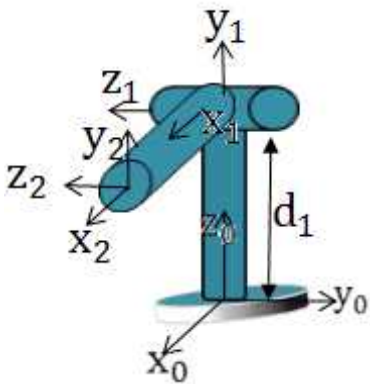


Figure 2-4. 2-link 2-axis robot

$$T_0^2 = \begin{bmatrix} c_1c_2 & -c_1s_2 & s_1 & d_1c_1c_2 \\ s_1c_2 & -s_1s_2 & -c_1 & d_2s_1c_2 \\ s_2 & c_2 & 0 & d_1+d_2s_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From the above matrix, a Cartesian position (x,y,z) is

$$P_x = d_2c_1c_2$$
$$P_y = d_2s_1c_2$$
$$P_z = d_1+d_2s_1.$$

Since x and y values of the end of the second joint are prescribed, divide P_x by P_y in the above equations.

$$\frac{P_y}{P_x} = \frac{d_2s_1c_2}{d_2c_1c_2} = \frac{s_1}{c_1} = \tan\theta_1$$

Find the angle θ_1 from the above equation.

$$\theta_1 = \text{atan}(\frac{P_y}{P_x}) = \text{atan2}(P_y, P_x)$$

However, you can recognize that you cannot find the solution, by investigating the foregoing to find θ_2 through the same procedure. In this case, to find the angle θ_2, if multiplying the matrix T_0^2 by $\left[A_0^1\right]^{-1}$, A_1^2 is made.

$$\left[A_0^1\right]^{-1} T_0^2 = \left[A_0^1\right]^{-1} A_0^1 A_1^2 = A_1^2$$

Since the left term is a given variable, the value of the right variable can be calculated, using the left term variables.

$$\left[A_0^1\right]^{-1} T_0^2 = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ 0 & 0 & 1 & -d_1 \\ 0 & -c_1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c_2 & -s_2 & 0 & d_2 c_2 \\ s_2 & c_2 & 0 & d_2 s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Comparing the left term with the right one, we can obtain the following two equations.

$$p_x c_1 + p_y s_1 = d_2 c_2$$

$$p_z - d_1 = d_2 s_2$$

Since values p_x, p_y, s_1, c_1 are given, θ_2 can be determined by

$$\tan\theta_2 = \frac{p_z - d_1}{p_x c_1 + p_y s_1}.$$

θ_2 is calculated as

$$\theta_2 = \text{atan2}(p_z - d_1 p_x c_1 + p_y s_1)$$

2-3-2. Example of 3-Axis Robot

Find the angle of each joint from the forward kinematics of the following 3-axis robot.

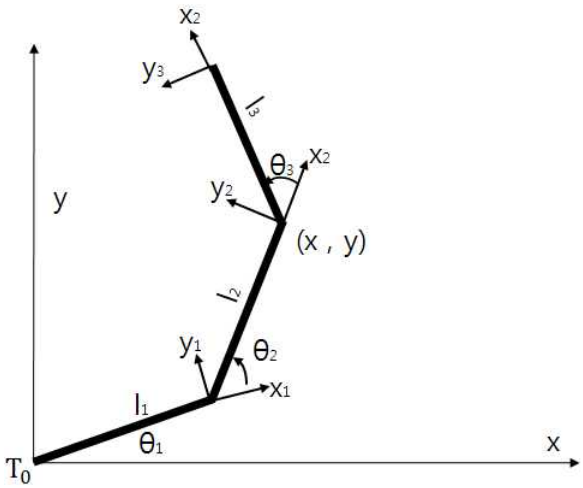


Figure 2-5. 3-axis robot

$$T_0^3 = \begin{bmatrix} c_{123} & -s_{123} & 0 & l_1c_1 + l_2c_{12} + l_3c_{123} \\ s_{123} & c_{123} & 0 & l_1s_1 + l_2s_{12} + l_3s_{123} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A Cartesian position (x, y, z), i.e., the forward kinematics is as follows.

$$P_x = l_1c_1 + l_2c_{12} + l_3c_{123}$$

$$P_y = l_1s_1 + l_2s_{12} + l_3s_{123}$$

$$P_z = 0$$

The orientation is as follows.

$$\phi = \theta_1 + \theta_2 + \theta_3$$

Since the end-effector P_x, P_y and ϕ are given, $P_x^{'}$ and $P_y^{'}$ can be determined.

$$P_x^{'} = P_x - l_3 c_{123} = P_x - l_3 c_\phi$$
$$P_y^{'} = P_y - l_3 s_{123} = P_y - l_3 s_\phi$$

From the above equations,

$$P_x^{'} = l_1 c_1 + l_2 c_{12}$$
$$P_y^{'} = l_1 s_1 + l_2 s_{12}.$$

From the above equations, the following equation can be derived.

$$P_x^{'2} + P_y^{'2} = (l_1 c_1 + l_2 c_{12})^2 + (l_1 s_1 + l_2 s_{12})^2$$
$$= l_1^2 + l_2^2 + 2 l_1 l_2 c_2$$

Rearranging the above equation to find the angle θ_2,

$$c_2 = \frac{P_x^{'2} + P_y^{'2} - l_1^2 - l_1^2}{2 l_1 l_2}$$
$$s_2 = \pm \sqrt{(1 - c_2^2)} \ .$$

θ_2 can be calculated as

$$\theta_2 = \text{atan2}(s_2, c_2).$$

Substituting θ_2 and the following equations determined above

$$P_x^{'} = l_1c_1 + l_2c_{12}$$

$$P_y^{'} = l_1s_1 + l_2s_{12}$$

to the following equations,

$$P_x^{'} = l_1c_1 + l_2(c_1c_2 - s_1s_2) = (l_1 + l_2c_2)c_1 - l_2s_2s_1$$

$$P_y^{'} = l_1s_1 + l_2(s_1c_2 + c_1s_2) = (l_1 + l_2c_2)s_1 - l_2s_2c_1.$$

Since θ_2 is already known, represent the above equations as the matrix as follows, to find s_2 and c_2.

$$\begin{bmatrix} -l_2s_2 & l_1 + l_2c_2 \\ l_1 + l_2c_2 & l_2s_2 \end{bmatrix} \begin{bmatrix} s_1 \\ c_1 \end{bmatrix} = \begin{bmatrix} P_x^{'} \\ P_y^{'} \end{bmatrix}$$

$$\begin{aligned} \det[] &= -(l_2s_2)^2 - (l_1 + l_2c_2)^2 \\ &= -(l_1^2 + l_2^2 + 2l_1l_2c_2) \\ &= -(P_x^{'2} + P_y^{'2}) \end{aligned}$$

Thus

$$\begin{bmatrix} s_1 \\ c_1 \end{bmatrix} = -\frac{1}{-(P_x^{'2} + P_y^{'2})} \begin{bmatrix} -l_2s_2 & -(l_1 + l_2c_2) \\ -(l_1 + l_2c_2) & -l_2s_2 \end{bmatrix} \begin{bmatrix} P_x^{'} \\ P_y^{'} \end{bmatrix}.$$

From the above equation, θ_1 can be calculated as

$$\theta_1 = \text{atan2}(s_2, c_2)$$

Finally, θ_3 can be calculated from $\phi = \theta_1 + \theta_2 + \theta_3$.

$$\theta_3 = \phi - \theta_1 - \theta_2$$

2-4. Differential Equations Method

So far, we have discussed the relationships between the angle of rotation of each joint and the end-effector of a robot, through kinematics calculation of a robot. For 2-DOF robot, the forward and the inverse kinematics can be calculated through the analytic approach. However, for a robot of complicated shape, the kinematics calculation is very difficult, generally. None the less, using the differential relationship between the rotation angle and the end-effector, the difficulty in kinematics calculation can be improved for a general robot.

2-4-1. Differential Relationship of Robot Kinematics

Let's examine the angular speed of each joint of a robot and the differential values of an end-effector, for example of a 2-link planar robot shown in Figure 8-5.

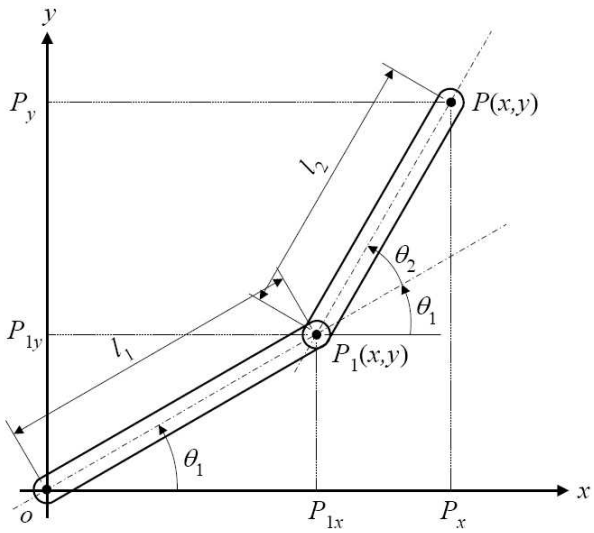


Figure 2-6. 2-link planar robot having rotational joints

To calculate the differential relationship between the rotation angle and an end-effector of a robot, the forward kinematics of a 2-link planar robot having rotational joint is provided as follows.

$$P_x = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2)$$

$$P_y = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2)$$

By differentiating both terms of the above equation for time, the following relationships can be obtained.

$$\dot{P}_x = -l_1\dot{\theta}_1 \sin(\theta_1) - l_2(\dot{\theta}_1 + \dot{\theta}_2)\sin(\theta_1 + \theta_2)$$

$$\dot{P}_y = l_1\dot{\theta}_1 \cos(\theta_1) + l_2(\dot{\theta}_1 + \dot{\theta}_2)\cos(\theta_1 + \theta_2)$$

At this time, we used the following differential equations of trigonometric function, during the differentiation.

$$\frac{d\sin(\theta)}{dt} = \frac{d\sin(\theta)}{d\theta}\cdot\frac{d\theta}{dt} = \frac{d\sin(\theta)}{d\theta}\dot{\theta},\ \frac{d\cos(\theta)}{dt} = \frac{d\cos(\theta)}{d\theta}\cdot\frac{d\theta}{dt} = \frac{d\cos(\theta)}{d\theta}\dot{\theta}$$

$$\frac{d\sin(\theta_1 + \theta_2)}{dt} = \frac{d\sin(\theta_1 + \theta_2)}{d(\theta_1 + \theta_2)}\cdot\frac{d(\theta_1 + \theta_2)}{dt} = \cos(\theta_1 + \theta_2)\cdot(\dot{\theta}_1 + \dot{\theta}_2)$$

$$\frac{d\cos(\theta_1 + \theta_2)}{dt} = \frac{d\cos(\theta_1 + \theta_2)}{d(\theta_1 + \theta_2)}\cdot\frac{d(\theta_1 + \theta_2)}{dt} = -\sin(\theta_1 + \theta_2)\cdot(\dot{\theta}_1 + \dot{\theta}_2)$$

The above equations can be notated in the form of matrix.

$$\begin{bmatrix} \dot{P}_x \\ \dot{P}_y \end{bmatrix} = \begin{bmatrix} -l_1 \sin(\theta_1) - l_2 \sin(\theta_1 + \theta_2) & -l_2 \sin(\theta_1 + \theta_2) \\ l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) & l_2 \cos(\theta_1 + \theta_2) \end{bmatrix} \cdot \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$

$$\dot{\mathbf{P}} = \mathbf{J}\dot{\boldsymbol{\theta}}$$

The matrix rearranged in terms of J for convenience is as follows and is said to be Jacobian matrix.

[Eq. 2-1]

$$\mathbf{J}=\begin{bmatrix}-l_1\sin(\theta_1)-l_2\sin(\theta_1+\theta_2) & -l_2\sin(\theta_1+\theta_2)\\ l_1\cos(\theta_1)+l_2\cos(\theta_1+\theta_2) & l_2\cos(\theta_1+\theta_2)\end{bmatrix},\ \mathbf{P}=\begin{bmatrix}\dot{p}_x\\ \dot{p}_y\end{bmatrix},\ \boldsymbol{\theta}=\begin{bmatrix}\dot{\theta}_1\\ \dot{\theta}_2\end{bmatrix}$$

The meaning of the Jacobian matrix J for robot kinematics is that the linear speed of an end-effector can be calculated by using the above equation, if the angular speed of each rotational joint is given, and also that the via speed of each rotational joint can be calculated by using the inverse matrix J^{-1} of Jacobian matrix, if the linear speed of an end-effector is given.

2-4-2. Physical Meaning of Jacobian Matrix

First, the equation 2-1 can be represented in detail by the differential definition equation.

$$\begin{bmatrix}\Delta P_x/\Delta t\\ \Delta P_y/\Delta t\end{bmatrix}=\begin{bmatrix}-l_1\sin(\theta_1)-l_2\sin(\theta_1+\theta_2) & -l_2\sin(\theta_1+\theta_2)\\ l_1\cos(\theta_1)+l_2\cos(\theta_1+\theta_2) & l_2\cos(\theta_1+\theta_2)\end{bmatrix}\cdot\begin{bmatrix}\Delta\theta_1/\Delta t\\ \Delta\theta_2/\Delta t\end{bmatrix}$$

Multiplying both terms by Δt, the above equation can be rearranged into the following, since time Δt is a scalar.

$$\begin{bmatrix}\Delta P_x\\ \Delta P_y\end{bmatrix}=\begin{bmatrix}-l_1\sin(\theta_1)-l_2\sin(\theta_1+\theta_2) & -l_2\sin(\theta_1+\theta_2)\\ l_1\cos(\theta_1)+l_2\cos(\theta_1+\theta_2) & l_2\cos(\theta_1+\theta_2)\end{bmatrix}\cdot\begin{bmatrix}\Delta\theta_1\\ \Delta\theta_2\end{bmatrix}\Rightarrow\Delta P=\mathbf{J}\cdot\Delta\theta$$

That is, the above equation shows, through Jacobian matrix J, the relationship between increment in the rotating angle of each joint $[\Delta\theta_1\ \Delta\theta_2]^T$ and increment in the position of an end-effector of a robot $[\Delta P_x\ \Delta P_y]^T$. The following Figure 8-6 visualizes the relationship

between increment in the rotating angle of each joint and increment in the position of an end-effector of a robot

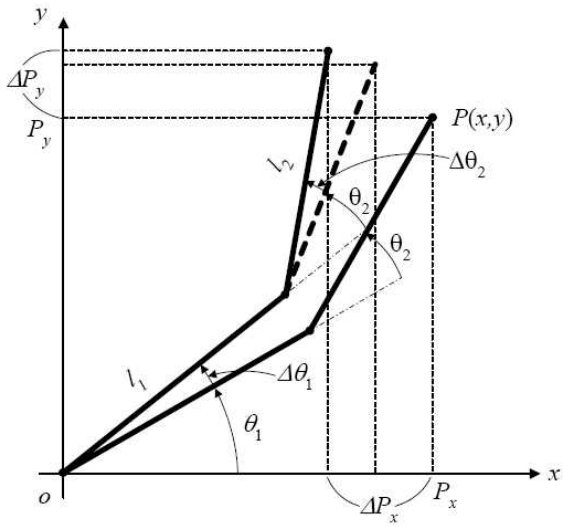


Figure 2-7. The relationship between increment in the rotating angle of each joint and increment in the position of an end-effector of a robot, in a 2-link planar robot having rotational joints

Exercise 2.3

In a 2-link planar manipulator having rotational joints given in Figure 8-6, the length of a link l_1 is 0.15m, the length of a link l_2 is 0.12m, and the joint angle θ_1 = 30′, θ_2 = 60′. Find the coordinate of an end-effector P_1(x,y). Further, if the joint angle is changed to θ_1 = 32′, θ_2 = 63′, approximately calculate P_2(x,y) by using Equation 3-2.

☞[Solution]

First, determine P(x,y) by using the forward kinematics equation.

$$P_{1x} = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2) = 0.15 \times \cos(30^0) + 0.12 \times \cos(30^0 + 60^0) = 0.1299(m)$$

$$P_{1y} = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2) = 0.15 \times \sin(30^0) + 0.12 \times \sin(30^0 + 60^0) = 0.1950(m)$$

Since a new position $P_2(x,y)$ corresponds to the increased ΔP_x and

ΔP_y from $P_1(x,y)$, it is necessary to first determine ΔP_x and ΔP_y before finding $P_2(x,y)$. This can be determined as follows by Equation 3-2. At this time, use θ_1 = 30′ and θ_2 = 60′ for Jacobian matrix and use $[2 \quad 3]^T$ for increment in the rotating joint.

$$\begin{bmatrix} -0.0131 \\ 0.0045 \end{bmatrix} = \begin{bmatrix} -0.1950 & -0.12 \\ 0.1299 & 0.00 \end{bmatrix} \cdot \begin{bmatrix} 0.0349 \\ 0.0524 \end{bmatrix}$$

When calculating the above equation, discriminate the use of Radian or Degree according to the calculator mode. Radian is used for the above equation.

Finally, calculate $P_2(x,y)$ using the following relation equations.

$$P_{2x} = P_{1x} + \Delta P_x = 0.1299 + (-0.0131) = 0.1168$$

$$P_{2y} = P_{1y} + \Delta P_y = 0.1950 + 0.0045 = 0.1995$$

※ Verification of the resulting $P_2(x,y)$ is performed as follows, using the forward kinematics, for checking the correctiveness.

$$P_{2x} = 0.15 \times \cos(32^0) + 0.12 \times \cos(32^0 + 63^0) = 0.1167(m)$$

$$P_{2y} = 0.15 \times \sin(30^0) + 0.12 \times \sin(30^0 + 60^0) = 0.1990(m)$$

Error may be calculated as 1.48% and -0.56%, respectively, which are within the valid range in terms of engineering science. That is, kinematics based on Jacobian matrix is acceptable in terms of engineering.

2-5. Singularities

In Figure 2-7, take into consideration the special status of a robot

where two links come one after another. A first case shows that the two links extend. A second case shows that the two links get folded. At this moment, robot end-effector's intention to move in the direction of an arrow mark cannot be accomplished. This is because of the limitation of the physical length of robot links for the first case. But, for the second case, the physical length of robot links is not of limitation.

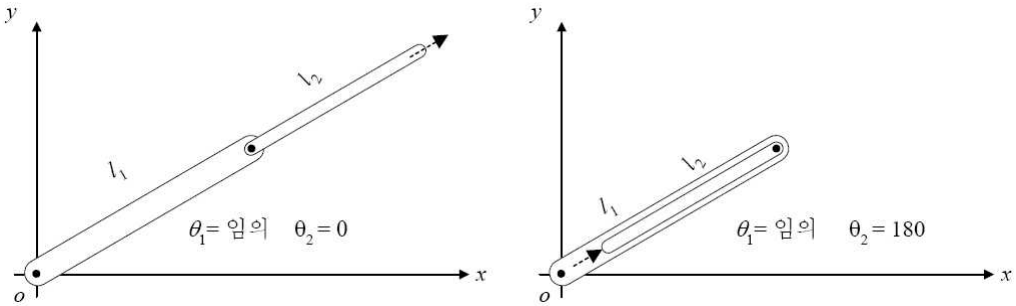


Figure 2-8. Special status of robot

임의: arbitrary

1) Singularity in a work space boundary : This takes place when an end-effector is on a work space boundary or close to it since a manipulator completely extends or gets folded.

2) Singularity inside a work space: This takes place away from the work space generally when two or more joint axes line up.

In the first case of Figure 2-8, if slightly moving θ_1 and θ_2, the end-effector of a robot slightly moves only in the direction marked in the following Figure 8-9. That, it moves only in the direction perpendicular to the intended direction.

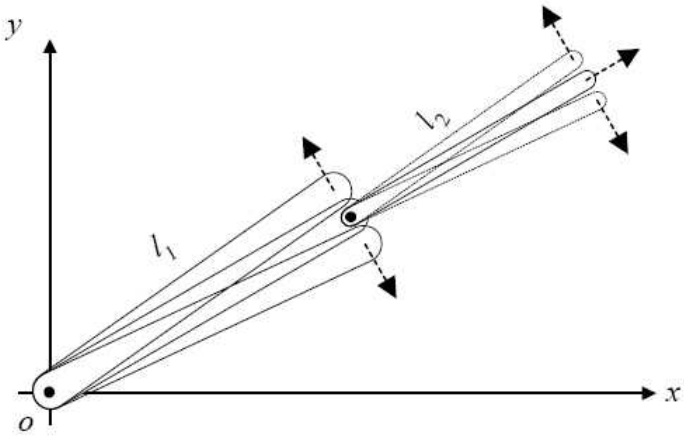


Figure 2-9. Small motion in special status of robot

Meanwhile, in the case of Figure 3.5, an end-effector of a robot can move in any direction by slightly moving θ_1 and θ_2. That is to say, for any Δx and Δy, the corresponding $\Delta\theta_1$ and $\Delta\theta_2$ can be determined. This means that Jacobian inverse matrix J^{-1} which is included in Equation $\Delta\theta = J^{-1} * \Delta P$ can be easily determined. Thus, the characteristic of Jacobian matrix relevant to the special case of Figure 8-7 can be analogized. That is, Jacobian inverse matrix J^{-1} does not exist.

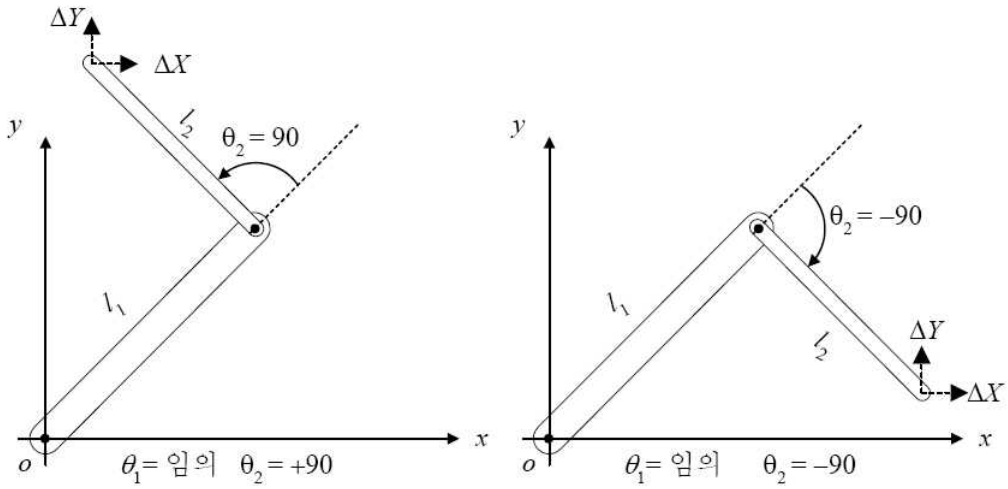


Figure 2-10. Special status of robot 2

To confirm such fact, let's find analytically the Jacobian matrix equation for Figures 8-9 and 8-10, by using the matrix equation exhibiting whether the inverse matrix is calculated

Case	Jacobian matrix equation
Figure8-7 left	0
Figure8-7 right	0
Figure8-9 left	$l_1 l_2$
Figure8-9 right	$-l_1 l_2$

Table 2-1 Jacobian equation for singularity

2-6. Easiness of Motion

In the previous explanation, according to the status of a robot, there was a case that an end-effector of a robot cannot move although each joint rotates, while there was a case that it can move to any position. It could be understood that the difference between the cases resulted from the solution value of Jacobian matrix equation. That is, it is preferable that a robot is operated in the status that the value of Jacobian matrix equation is the maximum as possible. It is necessary to properly adjust the link length so that the rotating angle θ_2 of a No. 2 link can move around ±90 degree for a given work, as possible.

For any case, it must be avoided that θ_2 is 0 or 180 degree. So then, what about θ_2 being 1 or 179 degree? What about θ_2 being 2 or 178 degree? Or, what about θ_2 being 3 or 177 degree? Of course, if θ_2 is not 0, a robot will be able to move smoothly. However, the smoothness will increase until θ_2 increases and becomes 90 degree.

Inversely, the smoothness will decrease until θ_2 decreases and becomes 0. When θ_2 reaches 0 degree, the smoothness will be zero.

Thus, for the intermediate value of θ_2, the smoothness will be intermediate amount. Accordingly, the smoothness of motion of an end-effector can be replaced with the magnitude of Jacobian matrix equation. Pay attention that this is all the way 'replaceable'. In other higher order situations, a case that the matrix equation cannot replace the smoothness may occur. We assume that Jacobian matrix equation exhibits the smoothness of motion of an end-effector by dealing with the easy cases in this study subject.

2-7. Source Code for Inverse Kinematics

PUMA Inverse Kinematics

Input :

$$T=\begin{bmatrix} 0.890919 & -0.295633 & 0.344767 & 337.389346 \\ 0.163166 & 0.916796 & 0.364502 & 631.419259 \\ -0.423840 & -0.268488 & 0.865028 & 94.933257 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Output :

$$q=\begin{bmatrix} q1 \\ q2 \\ q3 \\ q4 \\ q5 \\ q6 \end{bmatrix}$$

Real values of q are

$$q=\begin{bmatrix} 50 \\ 35 \\ 15 \\ 5 \\ -20 \\ -40 \end{bmatrix}$$

- Program Source -

```
// inv_roboto.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "stdio.h"
#include "conio.h"
#include "math.h"
#include "stdlib.h"

#define PI          3.141592
#define             ERROR    0.0001

//dh parameter (a alpha d theta)
double dh_parameter[6][4] = {
        0.0,        -90.0,      0.0,        60.0,
        431.8,      0.0,        149.09,     45.0,
        -20.33,     90.0,       0.0,        25.0,
        0.0,        -90.0,      433.07,     15.0,
        0.0,        90.0,       0.0,        -10.0,
        0.0,        0.0,        56.25,      -30.0
};

double input_matrix[4][4] = {
        0.890919,  -0.295633, 0.344767,  337.389346,
        0.163166,  0.916796,  0.364502,  631.419259,
        -0.423840, -0.268488, 0.865028,  94.933257,
        0,         0,         0,         1
};

//////////////////////////////////////////////////
//global variables
double A_Matrix[4][4][6];
double T_Matrix[4][4][6];
double jacobian[6][6];
double T_Matrix_Inverse[4][4];
double TAInverse[4][4];
double e[6];
double R_Matrix[6][6];
double Sum;
```

```
doubleSumR[6];
doubleV_Matrix[6][6];
doublex[6];


////////////////////////////////////////////////
//degree to radian conversion
doubledeg2rad(double deg)
{
        return (deg*PI)/180;
}

//radian to degree
doublerad2deg(double rad)
{
        return (rad*180)/PI;
}

//compute a matrix
void solve_a_matrix(int link)
{
        A_Matrix[0][0][link-1] = cos(deg2rad(dh_parameter[link-1][3]));
        A_Matrix[0][1][link-1] =
-cos(deg2rad(dh_parameter[link-1][1]))*sin(deg2rad(dh_parameter[link-1][3]));
        A_Matrix[0][2][link-1] =
sin(deg2rad(dh_parameter[link-1][1]))*sin(deg2rad(dh_parameter[link-1][3]));
        A_Matrix[0][3][link-1] =
dh_parameter[link-1][0]*cos(deg2rad(dh_parameter[link-1][3]));
        A_Matrix[1][0][link-1] = sin(deg2rad(dh_parameter[link-1][3]));
        A_Matrix[1][1][link-1] =
cos(deg2rad(dh_parameter[link-1][1]))*cos(deg2rad(dh_parameter[link-1][3]));
        A_Matrix[1][2][link-1] =
-sin(deg2rad(dh_parameter[link-1][1]))*cos(deg2rad(dh_parameter[link-1][3]));
        A_Matrix[1][3][link-1] =
dh_parameter[link-1][0]*sin(deg2rad(dh_parameter[link-1][3]));
        A_Matrix[2][0][link-1] = 0;
        A_Matrix[2][1][link-1] = sin(deg2rad(dh_parameter[link-1][1]));
        A_Matrix[2][2][link-1] = cos(deg2rad(dh_parameter[link-1][1]));
        A_Matrix[2][3][link-1] = dh_parameter[link-1][2];
```

```
        A_Matrix[3][0][link-1] = 0;
        A_Matrix[3][1][link-1] = 0;
        A_Matrix[3][2][link-1] = 0;
        A_Matrix[3][3][link-1] = 1;
}

//compute a t matrix
void solve_t_matrix(int link)
{
        int i,j,k,m;
        double out=0.0;

        for(k=0;k<link;k++)
                for(i=0;i<4;i++)
                        for(j=0;j<4;j++)
                        {
                                if(k==0) T_Matrix[i][j][k] = A_Matrix[i][j][k];
                                else
                                {
                                        out = 0;
                                        for(m=0;m<4;m++)

out+=T_Matrix[i][m][k-1]*A_Matrix[m][j][k];

                                        T_Matrix[i][j][k] = out;
                                }
                        }
}

//compute jacobian matrix
void solve_jacobian()
{
        jacobian[0][0] = 0;
        jacobian[1][0] = 0;
        jacobian[2][0] = 0;
        jacobian[3][0] = 0;
        jacobian[4][0] = 0;
        jacobian[5][0] = 1;
```

```
        for(int i=1;i<6;i++)
        {
                jacobian[0][i]  =  T_Matrix[1][3][i-1]*T_Matrix[2][2][i-1]  -
T_Matrix[2][3][i-1]*T_Matrix[1][2][i-1];
                jacobian[1][i]  =  T_Matrix[2][3][i-1]*T_Matrix[0][2][i-1]  -
T_Matrix[0][3][i-1]*T_Matrix[2][2][i-1];
                jacobian[2][i]  =  T_Matrix[0][3][i-1]*T_Matrix[1][2][i-1]  -
T_Matrix[1][3][i-1]*T_Matrix[0][2][i-1];
                jacobian[3][i] = T_Matrix[0][2][i-1];
                jacobian[4][i] = T_Matrix[1][2][i-1];
                jacobian[5][i] = T_Matrix[2][2][i-1];
        }
  }

  //print A Matrix
  void print_a_matrix()
  {
        printf("\t************A MATRIX************\n");
        for(int i=0;i<6;i++)
        {
                printf("%d A %d = \n",i,i+1);
                for(int j=0;j<4;j++)
                {
                        for(int k=0;k<4;k++)
                        {
                                printf("\t%.4f",A_Matrix[j][k][i]);
                        }
                        printf("\n");
                }
                printf("\n");
        }
  }

  //print all T matrix
  void print_t_matrix()
  {
        printf("\t************T MATRIX************\n");
        for(int i=0;i<6;i++)
        {
```

```
                printf("0 T %d = \n",i+1);
                for(int j=0;j<4;j++)
                {
                        for(int k=0;k<4;k++)
                        {
                                printf("\t%.4f",T_Matrix[j][k][i]);
                        }
                        printf("\n");
                }
                printf("\n");
        }
}

//print jacobian matrix
void print_jacobian()
{
        printf("\t************JACOBIAN MATRIX************\n");
        for(int i=0;i<6;i++)
        {
                for(int j=0;j<6;j++)
                {
                        printf("\t%.4f",jacobian[i][j]);
                }
                printf("\n");
        }
        printf("\n");
}

//solve e
void solve_e()
{
        e[0] = TAInverse[0][3];
        e[1] = TAInverse[1][3];
        e[2] = TAInverse[2][3];
        e[3] = -(TAInverse[1][2]-TAInverse[2][1])/2;
        e[4] = -(TAInverse[2][0]-TAInverse[0][2])/2;
        e[5] = -(TAInverse[0][1]-TAInverse[1][0])/2;
}
```

```
//solve T inverse
void T_Inverse()
{
        T_Matrix_Inverse[0][0] = T_Matrix[0][0][5];
        T_Matrix_Inverse[0][1] = T_Matrix[1][0][5];
        T_Matrix_Inverse[0][2] = T_Matrix[2][0][5];
        T_Matrix_Inverse[0][3] =
-((T_Matrix[0][0][5]*T_Matrix[0][3][5])+(T_Matrix[1][0][5]*T_Matrix[1][3][5])+(T_Matrix[2][0][5]*
T_Matrix[2][3][5]));
        T_Matrix_Inverse[1][0] = T_Matrix[0][1][5];
        T_Matrix_Inverse[1][1] = T_Matrix[1][1][5];
        T_Matrix_Inverse[1][2] = T_Matrix[2][1][5];
        T_Matrix_Inverse[1][3] =
-((T_Matrix[0][1][5]*T_Matrix[0][3][5])+(T_Matrix[1][1][5]*T_Matrix[1][3][5])+(T_Matrix[2][1][5]*
T_Matrix[2][3][5]));
        T_Matrix_Inverse[2][0] = T_Matrix[0][2][5];
        T_Matrix_Inverse[2][1] = T_Matrix[1][2][5];
        T_Matrix_Inverse[2][2] = T_Matrix[2][2][5];
        T_Matrix_Inverse[2][3] =
-((T_Matrix[0][2][5]*T_Matrix[0][3][5])+(T_Matrix[1][2][5]*T_Matrix[1][3][5])+(T_Matrix[2][2][5]*
T_Matrix[2][3][5]));
        T_Matrix_Inverse[3][0] = 0;
        T_Matrix_Inverse[3][1] = 0;
        T_Matrix_Inverse[3][2] = 0;
        T_Matrix_Inverse[3][3] = 1;
}

//print to screen
void output()
{
        printf("\n\n************Result************\n");
        printf("Output Homogeneous Transformation Matrix\n");
        printf("-------------------------------------\n");
        for(int i=0;i<4;i++)
        {
                for(int j=0;j<4;j++)
                {
                        printf("%.4f\t",T_Matrix[i][j][5]);
                }
```

```
                printf("\n");
        }
        printf("--------------------------------------\n");
        printf("Each Joint Angle Value\n");
        for(i=0;i<6;i++)        printf("%dth = %.6f\n",i+1,dh_parameter[i][3]);

}

//all variables initialization
void init()
{
        Sum = 0.0;
        int i,j,k;
        for(i=0;i<6;i++)
        {
                SumR[i] = 0.0;
                e[i] = 0;
                x[i] = 0;
                for(j=0;j<6;j++)
                        //init 6x6
                        jacobian[i][j] = 0;
                        R_Matrix[i][j] = 0;
                        V_Matrix[i][j] = 0;
        }

        for(k=0;k<6;k++)
                for(i=0;i<4;i++)
                        for(j=0;j<4;j++)
                        {
                                A_Matrix[i][j][k]=0;
                                T_Matrix[i][j][k]=0;
                                T_Matrix_Inverse[i][j] = 0;
                                TAInverse[i][j] = 0;
                        }
}

//solve r matrix
void solve_r_matrix()
{
```

```
        for(int i=0;i<6;i++)
        {
                if(i==0) for(int j=0;j<6;j++) R_Matrix[j][0] = jacobian[j][0];      //Ri=Ji when
i=0
                else
                {
                        for(int k=0;k<i;k++)
                                {
                                        Sum     =  (R_Matrix[0][k]*jacobian[0][i]  +
R_Matrix[1][k]*jacobian[1][i] + R_Matrix[2][k]*jacobian[2][i] +
                                R_Matrix[3][k]*jacobian[3][i]                      +
R_Matrix[4][k]*jacobian[4][i] + R_Matrix[5][k]*jacobian[5][i]) /
                                (R_Matrix[0][k]*R_Matrix[0][k]                     +
R_Matrix[1][k]*R_Matrix[1][k] + R_Matrix[2][k]*R_Matrix[2][k] +
                                R_Matrix[3][k]*R_Matrix[3][k]                      +
R_Matrix[4][k]*R_Matrix[4][k] + R_Matrix[5][k]*R_Matrix[5][k]);

                                SumR[0] += R_Matrix[0][k]*Sum;
                                SumR[1] += R_Matrix[1][k]*Sum;
                                SumR[2] += R_Matrix[2][k]*Sum;
                                SumR[3] += R_Matrix[3][k]*Sum;
                                SumR[4] += R_Matrix[4][k]*Sum;
                                SumR[5] += R_Matrix[5][k]*Sum;
                                }

                        for (int t=0;t<6;t++)
                                {
                                        R_Matrix[t][i] = jacobian[t][i] - SumR[t];
                                        SumR[t] = 0;        //initialization for next
loop steps
                                }
                }
        }

  }

  //solve xi
  void solve_x()
  {
```

```
        for (int i=0;i<6;i++){
                        V_Matrix[i][5] = e[i];
                }

                for (i=5;i>=0;i--)
                {
                        x[i] = ( (R_Matrix[0][i]*V_Matrix[0][i] +
R_Matrix[1][i]*V_Matrix[1][i] + R_Matrix[2][i]*V_Matrix[2][i] + R_Matrix[3][i]*V_Matrix[3][i]
+ R_Matrix[4][i]*V_Matrix[4][i] + R_mattrix[5][i]*V_Matrix[5][i]) /
(R_Matrix[0][i]*R_Matrix[0][i] + R_Matrix[1][i]*R_Matrix[1][i] + R_Matrix[2][i]*R_Matrix[2][i]
+ R_Matrix[3][i]*R_Matrix[3][i] + R_Matrix[4][i]*R_Matrix[4][i] +
R_Matrix[5][i]*R_Matrix[5][i]) );

                        for (int j=0;j<6;j++)
  {
                                if(i>0){
                                        V_Matrix[j][i-1] = V_Matrix[j][i] -
jacobian[j][i] * x[i] ;
                                        }
                                }
                        }
  }

  void initT()
  {
        for(int i=0;i<6;i++)
                for(int j=0;j<6;j++)
                        for(int k=0;k<6;k++)
                                T_Matrix[i][j][k] = 0.0;
  }
  ////////////////////////////////////////////////
  //main function
  void main()
  {
        double cur_error = 1;   //Initialize by the value greater than error
        int temp;

        //all variables initialization
        init();
```

```
while(cur_error>ERROR)
{
        cur_error = 0; //for loop start
        //cal a matrix
        for(int i=0;i<6;i++)    solve_a_matrix(i+1);

        //cal t matrix(1~6 joint)
        for(i=0;i<6;i++)        solve_t_matrix(i+1);

        //cal jacobianobian matrix
        solve_jacobian();

        //print and verification ->check
        //print_a_matrix();
        //print_t_matrix();
        //print_jacobian();

        //cal 0A6 Inverse
        T_Inverse();

        //cal input_matrix*T_matrix_inverse
        for(i=0;i<4;i++)
                for(int j=0;j<4;j++)
                        for(int k=0;k<4;k++)

TAInverse[i][j]+=input_matrix[i][k]*T_Matrix_Inverse[k][j];

        //cal e matrix
        solve_e();

        for (i=0;i<4;i++)
        {
                for(int j=0;j<4;j++)
                {
                        TAInverse[i][j] = 0;
                }
        }
```

```
                //solve r matrix
                solve_r_matrix();

                //solve xi
                solve_x();

                //error
                for (i=0;i<6;i++)
                {
                        dh_parameter[i][3] = dh_parameter[i][3] + (rad2deg(x[i]));
                        temp=(int)(rad2deg(x[i]));
                        cur_error = cur_error + pow(temp,2);
                }

                //check
                if(cur_error < ERROR) break;

                //print result
                output();
        }
}
```

Appendix 3

Trajectory Planning

3-1. Basics of Trajectory Planning

3-2. Trajectory Planning for 3rd-Order Polynomials

ED Corporation

Appendix 3

Trajectory Planning

Trajectory planning is to establish in advance the desired path, along which a robot will move over time in a Cartesian coordinate system or a joint space. The requirement for a general trajectory planning is to traject a path, that a robot wants, as fastest and most smoothly as it can.

For example, let's Assume that a robot moves from point A to point B. There are infinite ways from A to B. But, the fastest and most smooth path is determinate.

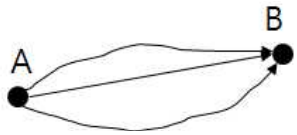


Figure 3-1. A variety of paths between two points

For 6-axis robot, let's examine the case that a robot moves along a certain far distance from one point to another point in a Cartesian space. Moving in a Cartesian space can be represented by the linear motion of each joint. However, this may also represent very unexpected motion of an end-effector of a robot. Although it is possible that a robot is activated to move from point A to B along the straight line the fastest path, this is not preferable. This is because, as shown in Figure 3-2 below, the velocity at the start point and the end point of the path is discontinuous. Since this indicates that it is necessary for a robot to provide abrupt acceleration at both points, a robot may be given such a heavy burden.

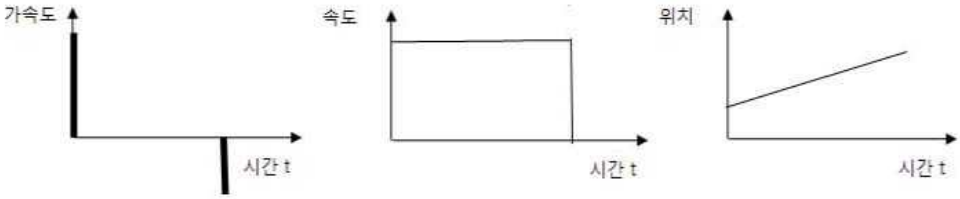


Figure 3-2. Problem on straight line path

가속도: acceleration
시간: time
속도: velocity
위치: position

Therefore, to move a robot from point A to a target point B in Cartesian coordinate space, plenty of via points should be set so that a robot can move as most smoothly as it can. Like this, planning for a robot to pass through many via points and arrive the target point smoothly is the purpose of the "trajectory planning". Such trajectory planning is pre-determined and a motion controller is operated with the control reference input which is one point on a path that a robot has to move at every sampling times. Calculation of the trajectory planning is performed by using polynomial equations of the function of time.

Resultantly, when a robot moves to the desired position in a Cartesian space, the joint variables has to be actually controlled and operated. Thus, via points that the corresponding joint variables must pass through is necessary, and also the trajectory planning in a joint space is necessary. Figure 9-3 shows a trajectory planning block in the whole block diagram of a robot. A robot controller block is controlled in a joint space, and on the other hand the trajectory planning can be performed either in a joint space or in a Cartesian space. In an actual realization, the path trajectory planning can be performed in advance in an off-line mode rather than in an on-line mode, and then can be provided to a control system at every sampling times.

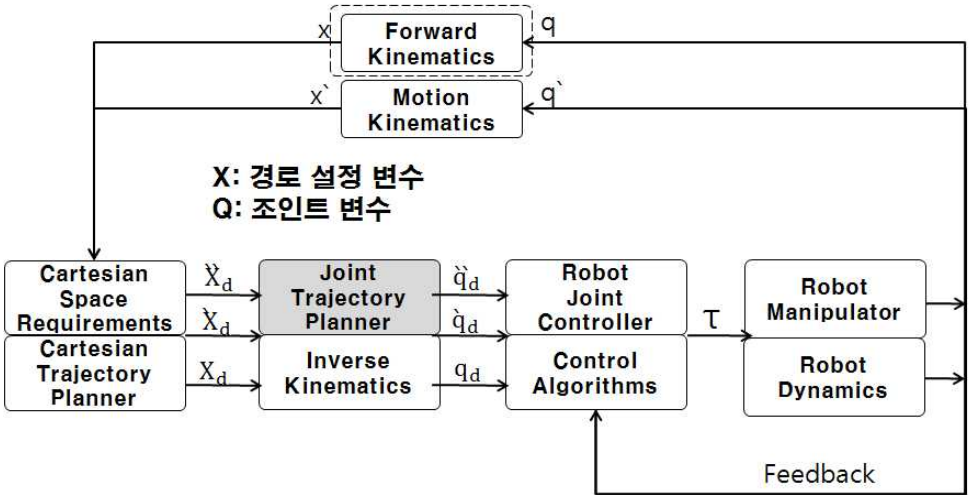
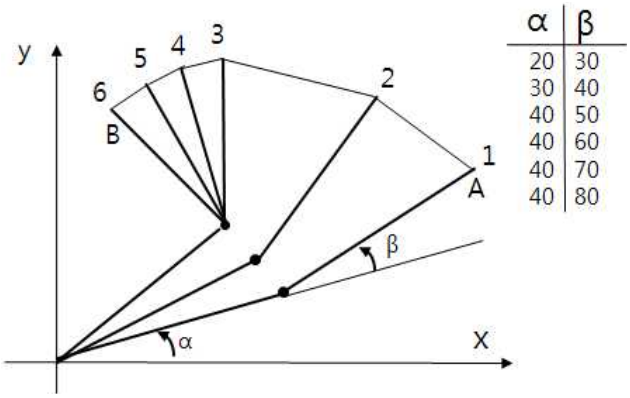


Figure 3-3. Whole block diagram of robot: Trajectory planning

X: Path Setting Variable
Q: Joint Variable

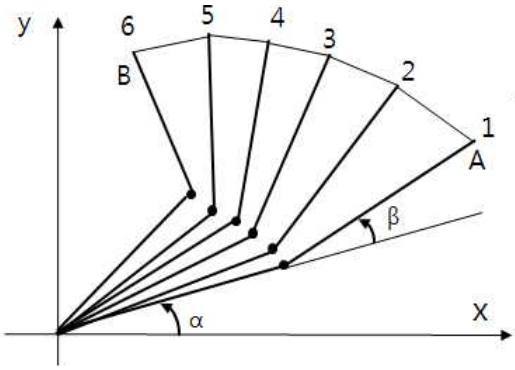
3-1. Basics of Trajectory Planning

To understand the principle of trajectory planning in a joint space and a Cartesian space, consider a simple 2-DOF robot. We want to move a robot from point A to point B as shown in Figure 3-4.



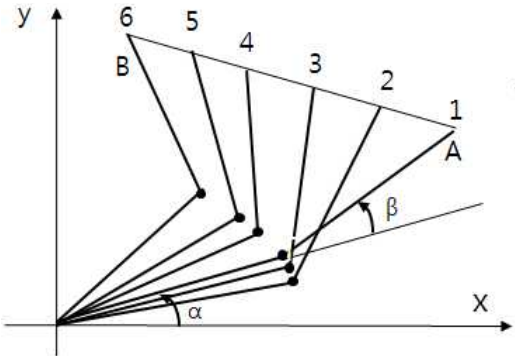
α	β
20	30
30	40
40	50
40	60
40	70
40	80

Figure 3-4. De-normalized moving of 2-DOF robot in joint-space representation



α	β
20	30
24	40
28	50
32	60
36	70
40	80

Figure 3-5. Normalized moving of 2-DOF robot in joint-space representation



α	β
20	30
14	55
16	69
21	77
29	81
40	80

Figure 3-6. Moving of 2-DOF robot in joint-space representation

At point A, a robot shape has α=20‘ and β=30‘. Assuming that a robot has been calculated to be positioned at point B, the angles have to be α=40‘ and β=80’. In addition, let's assume that both joints of a robot move at the velocity of 10 degree/sec maximum. One way to move a robot from point A to B is to run both of the two joints at their maximum angular velocities. After 2 seconds, a lower link of a robot will have completed its motion, while an upper link continues for another 3 seconds, as shown in the drawings. In the drawings, the path of an end-effector of a robot is also illustrated. As illustrated here, the path is irregular and the distances traveled by the end-effector are not uniform.

Now, Assume that the motion of both joints has been normalized by a common factor such that the joint with smaller motion will move proportionally slower and the both joints will simultaneously start and stop their motion. At this time, the both joints move at different speeds, but move together continuously. Whenever α varies at every 4', β varies at every 10'. The result of trajectory planning will be different as in Figure 3-5.

Although the path is still irregular, note that each part of motion is similar to one another, unlike any previous example. All these two cases have been planned in a joint space, and here only the target position and for a second case and only the calculation on the normalization of the joint velocity are necessary.

Now, let's assume that a robot hand follows a known path (i.e., straight line) between point A and point B. The simplest solution, as shown in Figure 3-6, is to draw a line from A to B, and thus divide the straight line into 5 segments and calculate the necessary angles α and β at each point. This is called as 'interpolation' between point A and B[3, 4, 5]. It this case, it is to be noted that the straight line or the joint angles do not change uniformly. Although the result of motion follows the straight path, it is necessary to analyze about the joint values at each point. Of course, the calculation at the more points has to be done more for more preciseness. If dividing the line into so little segments, a robot cannot exactly follow the straight line at each segment. Since such a trajectory is calculated based on the information that all portions of the motion are represented in a Cartesian frame, the trajectory exists in a Cartesian space.

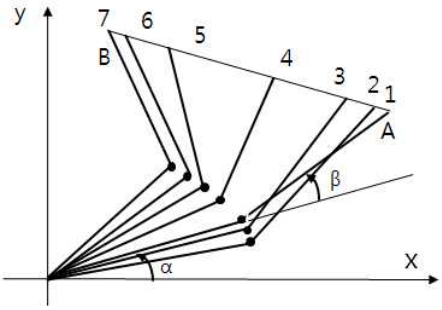


Figure 3-7. Trajectory planning in accelerated-decelerated area

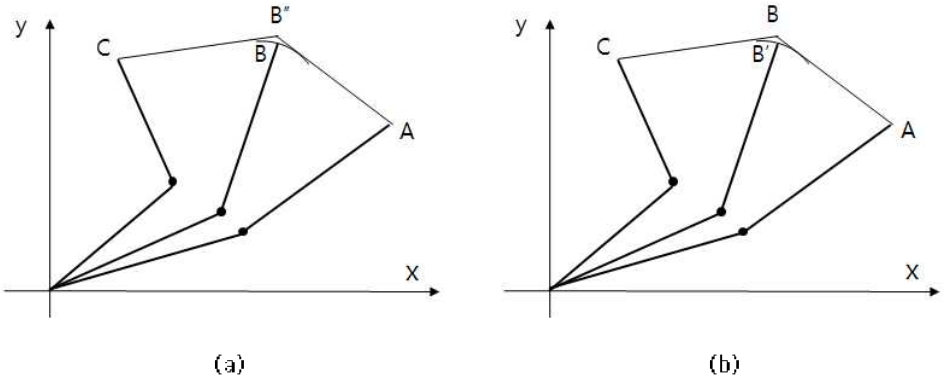


Figure 3-8. Blending different motion segments in a path

In the above case, assume that the necessary amount of accelerating force and decelerating force is provided by a joint, since an actuator of a robot is sufficiently powerful. Note that we assume that a robot hand is accelerated in a moment, such that the robot hand obtains the desired velocity to start its motion, for example, at a first segment. Otherwise, the robot will follow another trajectory different from the prediction, and acceleration time to the desired acceleration value will be slightly delayed. To improve this problem, divide the motion segment with different sizes, one of which at the beginning point is more smaller divided, in order to increase the robot arm's velocity, and then the robot goes at a constant acceleration rate. As the robot approaches point B, decelerate it at smaller segments. These are illustrated in Figure 9-7. Of course, the inverse kinematics equation for a robot has to be solved

at each point, as in the previous description.

However, instead of dividing the straight line AB into the same segments, we will divide it based upon $x=1/2at^2$ until obtaining the acceleration rate of y=at by arriving the time t. Similarly, the final segment of motion can be divided based upon the acceleration and deceleration regions.

Another kinds of the trajectory planning is to make trajectory planning following not the linear path but a certain desired path. To do so, the coordinate of each motion segment has to be calculated based on the desired path, and this coordinate is used to calculate the joint variables at each motion segment.

So far, we have considered the robot motion between two points A and B. However, in the most cases, a robot has to pass many continuous points as well as via points. The second step of trajectory planning is between multiple points and finally it focuses on continuous movements.

Let's assume that the robot moves from A to C by way of B as shown in Figure 9-8. One way to move the robot is to accelerate it at point A, to decelerate it before approaching B and stop it at B, to start again it from B and accelerate it till C, and to decelerate it when approaching C and stop it at C. Such a stop-and-go motion will create severe jerky motions because of unnecessary stops. An alternative scheme is to blend the motion segments at point B. When the robot approaches point B, decelerate it to follow the blended path, again accelerate it towards C, and stop it at C. By doing so, much better motion can be accomplished and the stress applied to the robot and the energy consumption can be lessen. If the motion comprises more segments, all the via points can also be blended. Blending the motion segments as above has the robot pass through point B' not point B(see Figure 3-8 (a).) If the robot is needed to exactly pass through point B, another point B" (see Figure 3-8 (b)) has to be specified so that the robot can exactly pass through point B, after creating a new point

considering the blending.

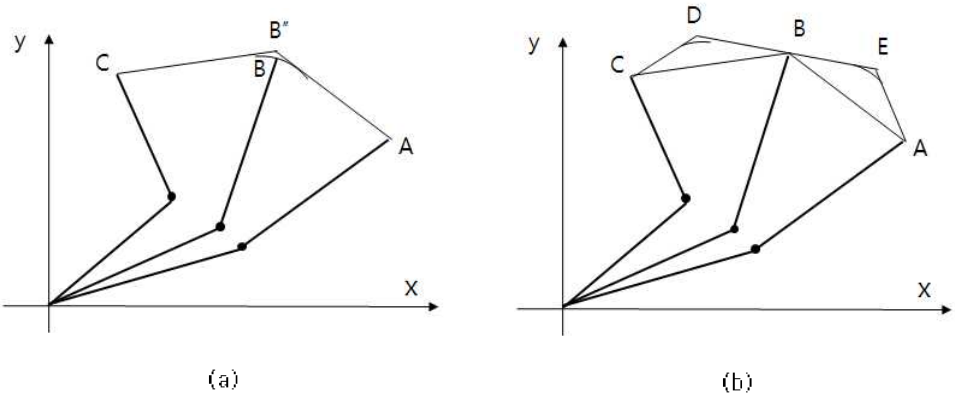


Figure 3-9.

Another alternative specifies, as in Figure 9-9, two via points D and E, through which the robot passes before and after points B. By doing so, point B will fall on the straight line section of the segment ensuring that the robot will pass through point B.

3-2. Trajectory Planning for 3rd-Order Polynomials

Now, when the initial location and orientation of the robot are known, find the joint angles for the desired position and rotation, using the inverse kinematic equations. Now, consider θ_i which is one of the joints at the initial point of motion segments at time t_i. We here wants for the joint to move to a new value θ_f. This way uses the polynomial for trajectory planning, which lets the already known θ_i and θ_f be coincident with the initial and final boundary conditions, that is, setting the velocities of intial and final motion segments to zero. Using the four information, we can solve the following four unknowns.

$$\theta(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3$$

Eq. 3-1

Here, the intial and final conditions are as follows.

$$\theta(t_i) = \theta_i$$
$$\theta(t_f) = \theta_f$$
$$\dot{\theta}(t_i) = 0$$
$$\dot{\theta}(t_f) = 0$$

First derivative of the polynomial equation Eq. 3-1 is

$$\dot{\theta}(t) = c_1 + 2c_2 t + 3c_3 t^2$$

Eq. 3-2

Substituting the intial and final conditions for Eq. 3-1 and Eq. 3-2, we obtain

$$\theta(t_i) = c_0 = \theta_i$$
$$\theta(t_f) = c_0 + c_1 t_f + c_2 t_f^2 + c_3 t_f^3$$
$$\dot{\theta}(t_i) = c_1 = 0$$

$$\dot{\theta}(t_f) = c_1 \quad 2c_2 t_f + 3c_3 t_f^2$$

Eq. 3-3

Solving the four equations simultaneously, the values necessary for the constant can be obtained.

In such a manner, the joint position for a certain period can be calculated, and a controller uses this to control the joint position. A similar procedure has to be individually applied for each joint, and however, it must be operated from beginning to end altogether. Obviously, even though the intial and final velocity are not zero, a

certain given value can be used for this way.

If two or more points are defined and a robot passes continuously through the points, the boundary velocity and position can be used as the initial value for the next motion segment. Such a 3rd-order polynomial equation can also be used for trajectory planning of each motion segment. However, if the acceleration is not continuous even though the position and the velocity is continuous, some problem may occur.

Exercise 3.1

It is desired to have a first joint of a 6-axis robot go from initial angle of 30' to a final angle of 75' in 5 seconds. Using a third-order polynomial, calculate the joint angle at 1, 2 3, and 4 seconds.

☞[Solution]

Substituting the boundary conditions for Eq. 3-3, we obtain

$$\theta(t_i) = c_0 = 30$$

$$\theta(t_f) = c_0 + c_1(5) + c_2(5^2) + c_3(5^3)$$

$$\dot{\theta}(t_i) = c_1 = 0$$

$$\dot{\theta}(t_f) = c_1 \quad 2c_2(5) + 3c_3(5^2)$$

Thus, c_0=30, c_1=0, c_2=5.4, and c_3=-0.72.

Resultantly, this will be 2nd-order equation in terms of the position, velocity, and acceleration, as follows.

$$\theta(t) = 30 + 5.4t^2 - 0.72t^3$$

$$\dot{\theta}(t) = 10.8t - 2.16t^2$$

$$\ddot{\theta}(t) = 10.8 - 4.32t$$

Substituting the desired time interval for the motion equation, we get

$\theta(1) = 34.68'$ $\theta(2) = 45.84'$ $\theta(3)=59.16'$ $\theta(4)=70.32'$

Exercise 3.2

Assume a robot arm in Exercise 3.1 continues moving to the next point, and arrives 105' in 3 seconds. Plot the position, velocity, and acceleration curves.

☞[Solution]

As shown in Exercise 3.1, since we know, from the first motion segment, the joint position and velocity, we use these values as the intial condition for the followings the next segment. We can obtain

$$\theta(t) = c_0+c_1 t + c_2 t^2 + c_3 t^2$$
$$\dot{\theta}(t) = c_1 + 2c_2 t+3c_3 t^2$$
$$\ddot{\theta}(t) = 2c_2+6c_3 t$$

Here, known values for each time are as follows.

$$\theta t_i = 0 \quad \theta_i = 75 \quad \dot{\theta}_i=0$$
$$\theta t_f = 3 \quad \theta_f = 105 \quad \dot{\theta}_f=0$$

Thus, we can obtain

$$c_0 =75, \quad c_1=0, \quad c_2=10, \quad c_3=-2.222$$
$$\theta(t) = 75 +10t^2 - 2.222t^3$$
$$\dot{\theta}(t) = 20t - 6.666t^2$$
$$\ddot{\theta}(t) = 20 - 13.332t$$

ED-7255

Industrial Robot Simulator

1st Published 29 March 2010
1st Printing 29 March 2010

Author ED Corp., R&D Center
Publisher Park, Yong-Hu

Publisher Company: ED Corporation

Headoffice, Factory, and R&D Center
517-15 Sangdaewon-Dong, Joongwon-Gu,
Seongnam, Gyeonggi, Korea
TEL : (031)730-7300 (rep.)
FAX : (031)730-7312
URL : http://www.ed.co.kr

CONTACT : For any inquiry of the trainer or textbook, please contact us.
Trainer Development : R&D Center
Textbook Development : R&D Center
Technical Aid : RnD@ed.co.kr

All Rights Reserved. Duplicating or reproducing all or part of the contents of this textbook, without permission from ED Corporation, shall be prohibited by Copyright Law.