

v1.00

Close to Real.

ROBOTIS

Bioloid

Expert Manual

ROBOTIS CO.,LTD. www.robotis.com



Table of Contents

1 . Introduction of the Bioloid Expert Level Education Kit	4
1 - 1 . What is the Bioloid Expert Level Education Kit?	5
1 - 1 - 1 . The Features of the Expert Level Education Kit	5
1 - 1 - 2 . The Different Levels of the Bioloid Kit	5
1 - 2 . The Components of Bioloid Expert Level Education Kit	6
1 - 2 - 1 . The Components of the Expert Level Education Kit	6
1 - 2 - 2 . The Components of Bioloid Robot	7
1 - 2 - 3 . Expert level education kit and CD contents and manual.	8
1 - 2 - 4 . Parts of expert level education kit	10
1 - 3 . Using The Bioloid Expert Level Education Kit	11
1 - 3 - 1 . Usage Overview of the Expert Level Education Kit	11
1 - 3 - 2 . Lesson Contents Using Expert Level Education Kit	13
1 - 3 - 3 . Examples of an Expert Level Application Robot	15
2 . Using the Bioloid Expert Level Educational Kit	16
2 - 1 . CM-5 BASED ROBOT PROGRAMMING	17
2 - 1 - 1 . Installing Development Resources	17
2 - 1 - 2 . CM-5 CONTROL	34
2 - 1 - 3 . Dynamixel Control	49
2 - 1 - 4 . Wireless Data Communication	60
2 - 2 . PC BASED ROBOT PROGRAMMING	69
2 - 2 - 1 . Installing Development Resources	69
2 - 2 - 2 . Dynamixel Unit Control	75
2 - 2 - 3 . Wireless Data Communication	98
2 - 2 - 4 . Image Handling and Recognition	107
3 . Applications of the Bioloid Expert Level Educational Kit	127
3 - 1 . Controlling the Probe Robot Via Remote Control	128
3 - 2 . Controlling Humanoid Robot Via Remote Control	143
3 - 3 . Controlling the Humanoid Robot Via PC	154
3 - 4 . Image Communication and Wireless Control of the Humanoid Robot	162
3 - 5 . Image Recognition and Movement Detection for the Puppy Robot	168
3 - 6 . Image Recognition and Object Tracking with a Pan/Tilt Camera Robot	188
3 - 7 . Image Recognition and a Self-Controlled Wheeled Robot	206
3 - 8 . Image Recognition and Object Tracking with the Humanoid Robot	239

4. APPENDIX	270
4-1. LIST OF THE CM-5 LIBRARY FUNCTIONS	271
4-1-1. LED LIBRARY	271
4-1-2. Button Library	273
4-1-3. UART0 (DYNAMIXEL SERIAL COMMUNICATION) LIBRARY	276
4-1-4. Dynamixel Control Library	279
4-1-5. UART1 (PC SERIAL COMMUNICATION) LIBRARY	290
4-1-6. Wireless Data Communication Library	300
4-1-7. Battery Charge Library	305
4-1-8. Other Essential Library	307
4-2. The List of Window Library Function	313
4-2-1. Dynamixel Control Library	313
4-2-2. Wireless Data Communication Library	324
4-2-3. Image handling and recognition library	329

1 . Introduction of the Bioloid Expert Level Education Kit

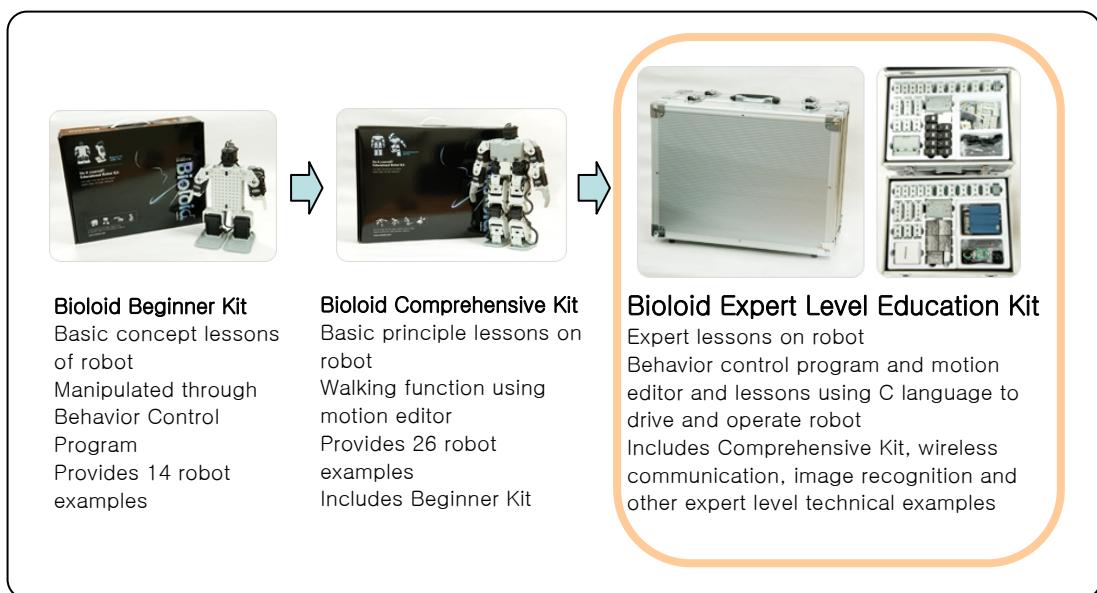
1 – 1 . What Is the Bioloid Expert Level Education Kit?

1 – 1 – 1 . The Features of the Expert Level Education Kit

Compared to the beginner and comprehensive kit, the Bioloid expert level education kit is specifically designed for students who want to acquire an advanced knowledge of robots. The expert level education kit not only provides a Window-based Behavior Control Programmer, Motion Editor, program sources, examples, and libraries for users, but also in-depth explanations on wireless communication, image processing, and image recognition.

1 – 1 – 2 . The Different Levels of the Bioloid Kit

The Bioloid expert level education kit is the most comprehensive version and it includes the contents of both the beginner and comprehensive kit. In order for students to effectively follow the lessons provided in the expert kit, they must know how to assemble robots and understand the core essentials of Behavior Control Programming and the motion editor as explained in the beginner and comprehensive kit. Moreover, to fully understand the contents in this manual, users must have basic understanding of C language and microprocessors.



1 – 2 . The Components of Bioloid Expert Level Education Kit

1 – 2 – 1 . The Components of the Expert Level Education Kit

◎ Bioloid Robot



The basic component of the expert level education kit is the Bioloid robot. The Bioloid robot is composed of the CM-5, AX-12(Dynamixel motor), AX-S1(Dynamixel sensor), and other connection parts; including frames, cables, bolts, and nuts.

◎ Wireless Data Communication Set



Bioloid uses the ZIGBEE method for wireless data communication. As such, a ZIG-100 wireless communication module and ZIG2Serial for PC interface are provided. ZIG-100 is embedded in the CM-5, allowing communication between robots (Remote control communication) and for controlling robots via PC when the ZIG-100 is connected to the ZIG2Serial.

◎ Wireless Camera Set



The wireless camera set is composed of a wireless image transmitter (wireless camera) and a receiver. By receiving an image from the transmitter that is embedded in Bioloid robot, the receiver connected to the PC via USB port can process the image. Additional software is provided for the above operations.

◎ USB2Dynamixel

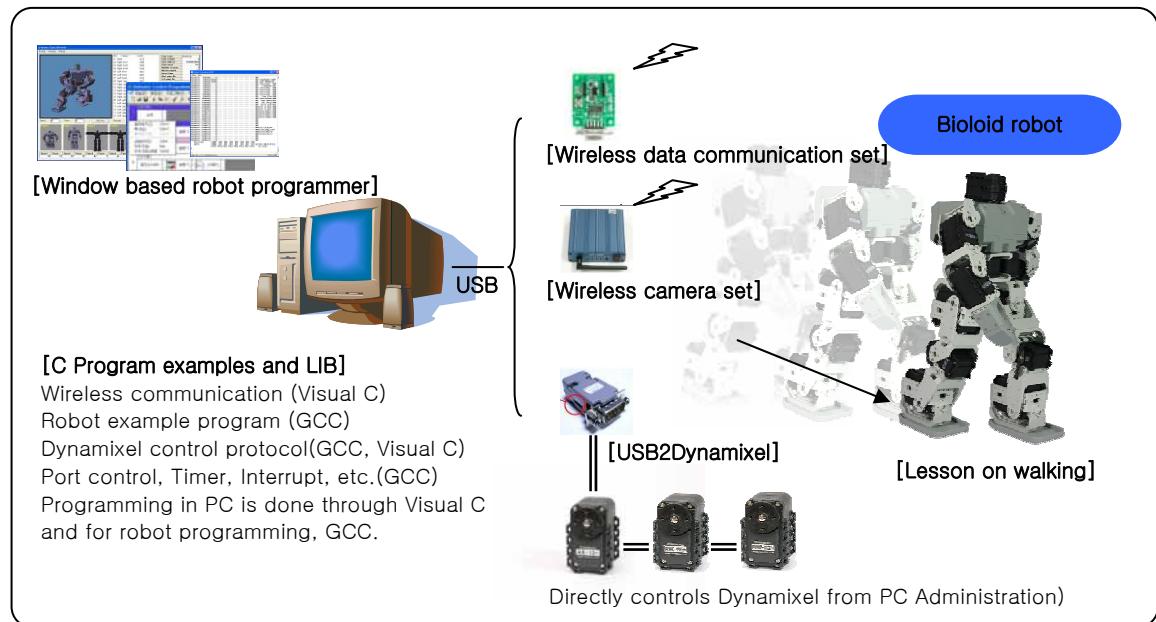


USB2Dynamixel is used when Dynamixels are directly controlled by PC. It is connected to the PC via a USB port. The USB2Dynamixel can also be converted to a serial port for PC that does not have a serial port and connect to the CM-5 of the Bioloid robot.

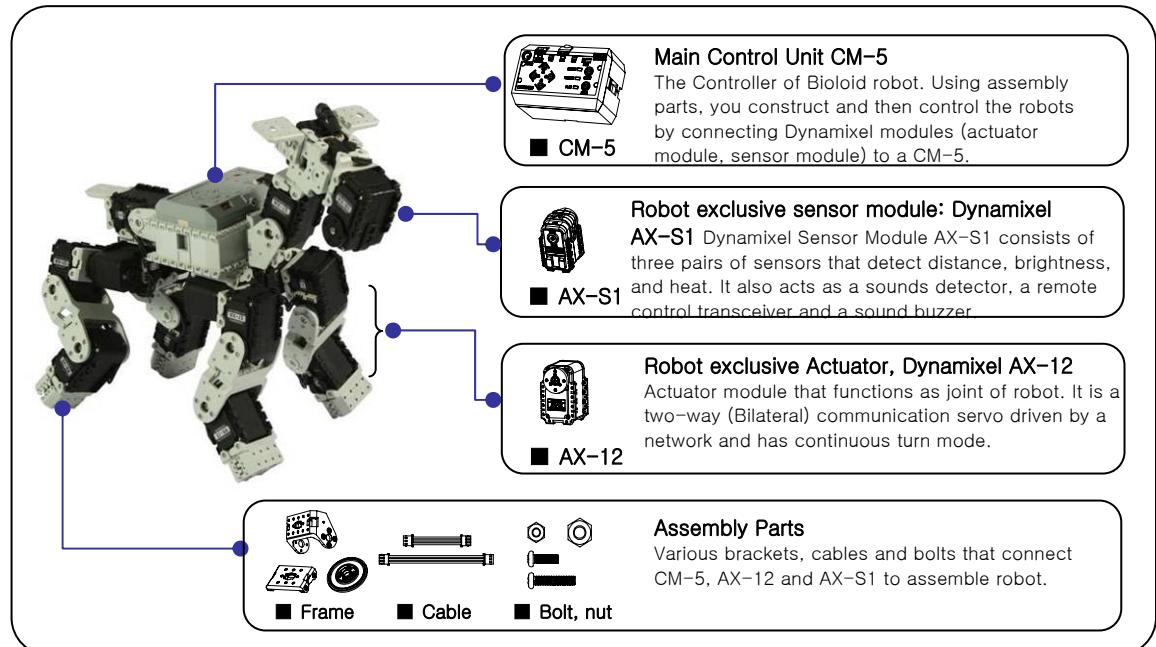
◎ Software



In the basic CD, the Behavior Control Programmer, Motion Editor, and Robot Terminal are provided; the expansion CD provides a GCC compiler for C programs, a development environment, and a library, as well as various Window libraries and example sources are provided for users to control Dynamixel and Bioloid robots via PC.



1 – 2 – 2 . The Components of Bioloid Robot



1 – 2 – 3 . Expert level education kit and CD contents and manual.

◎ Basic CD



- Software : Programs that need to be installed on your PC in order to operate the Bioloid robot.
- Manual : Quick Start, User's Guide, and manuals for the Dynamixel (AX-12, AX-S1).
- Applied : Various programs and video clips of the Bioloid robots.
- robot Examples : Robot program source code that are also explained in the User's Guide.
- Help Files : Help documents and video clips that assist in operating the Bioloid robots.

◎ Expansion CD



- Bioloid SDK
 - CM-5 : C program library and examples for CM-5
 - Windows : C program library and examples for PC Windows
- Manuals : Documents for expert level education
- Install : Installation files for WinAVR, USB2Dynamixel, Wireless Camera Set.
- Help : Help documents and video clips of Bioloid robots.

◎ Manuals

▷ Quick Start



Quick start briefly explains how to assemble and operate robots, and how to download a program. A printed version of the quick start book is provided for your convenience.

▷ User's Guide



The User's Guide explains operating principles and the programming process of the Bioloid robot in detail. The manual contains detailed explanations and examples of the CM-5 and Dynixel modules, as well as how to use software. This guide is essential for users who want to become proficient in assembling the robots. A printed User's Guide is provided for your convenience.

▷ Expert Level Manual

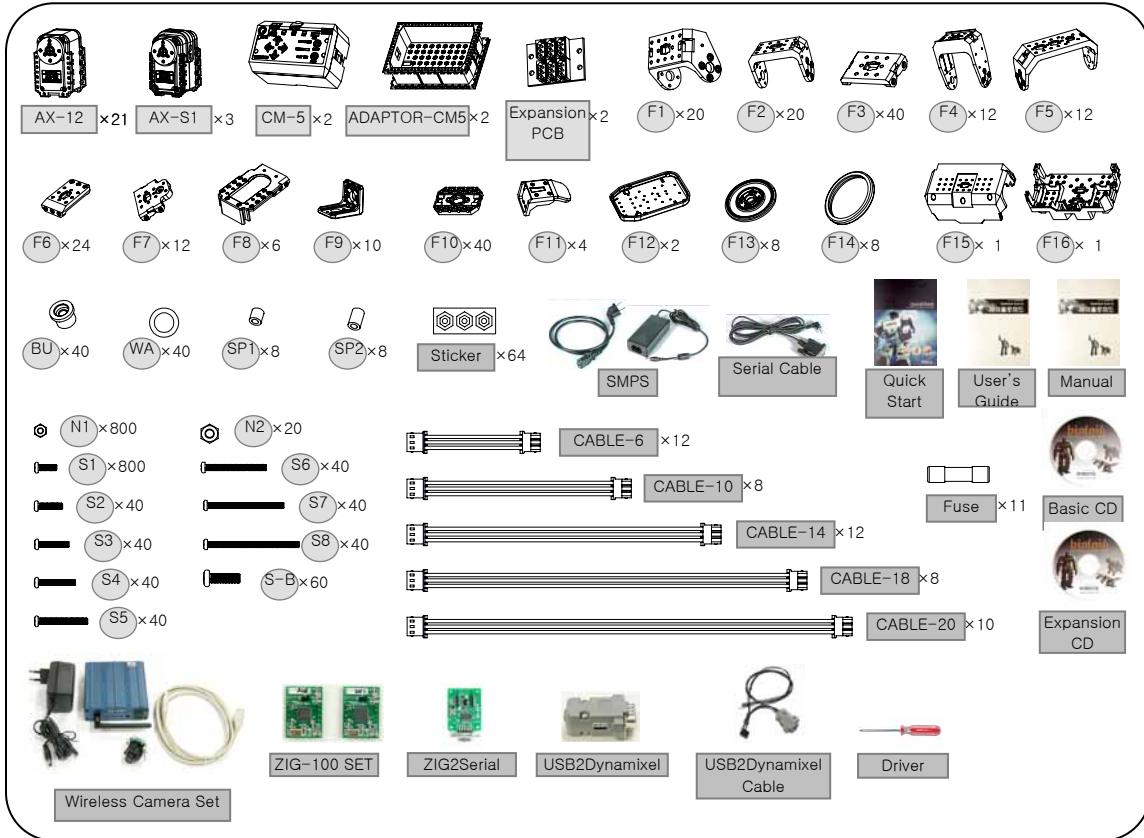


This manual is specifically made for students who want to acquire an advanced knowledge of robots. The manual not only provides detailed explanations on Window-based Behavior Control Programmer and Motion Editor, but also provides program source code, examples and libraries, and details on wireless communication, image recognition, etc. A printed expert manual is provided.

▷ AX-12 & AX-S1 Manual

This manual is generally referenced by the experts in the field. If the users want to learn the advanced functions of AX-12, they can use this manual. A PDF document is included in the CD.

1 - 2 - 4 . Parts of expert level education kit



1 – 3 . Using The Bioloid Expert Level Education Kit

1 – 3 – 1 . Usage Overview of the Expert Level Education Kit

- ◎ Usage overview of existing beginner kit and comprehensive kit

- ▷ Select the robot design

From the Quick Start guide, select which robot to make.

- ▷ Assemble the robot

Assemble the robot referring to the Quick Start guide.

- ▷ Program edit and download

Refer to the Quick Start guide and download robot behavior control program and the motion data examples onto the robot. If necessary, users can also refer to the User's Guide and edit the behavior control program and motion data.

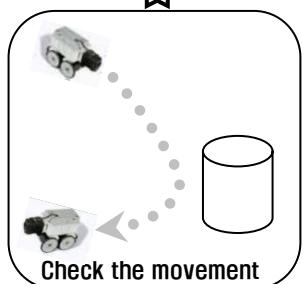
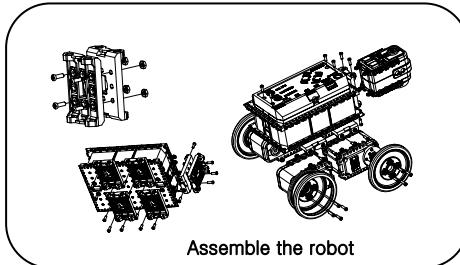
- ▷ Check the movements

Check the movement of robot as explained in the Quick Start guide.

If the user directly programmed the changes, check to see if the robot behaves accordingly.

- ▷ Completion

Usage overview of beginner and comprehensive kit



◎Using the Expert Level Educational Kit

Unlike the beginner and comprehensive kits, users who use expert level education kit can, in addition to downloading and using the behavior control program and motion data, directly program the controller by programming in C language or control the robots via PC by C language programming. Additional modules, including camera and wireless communication, can also be utilized.

▷ Select the robot design

Refer to the Quick Start guide and select a pre-designed robot or design your own robot.

▷ Assemble the robot

Users can either refer to the Quick Start guide or directly assemble the robot as they have designed it.

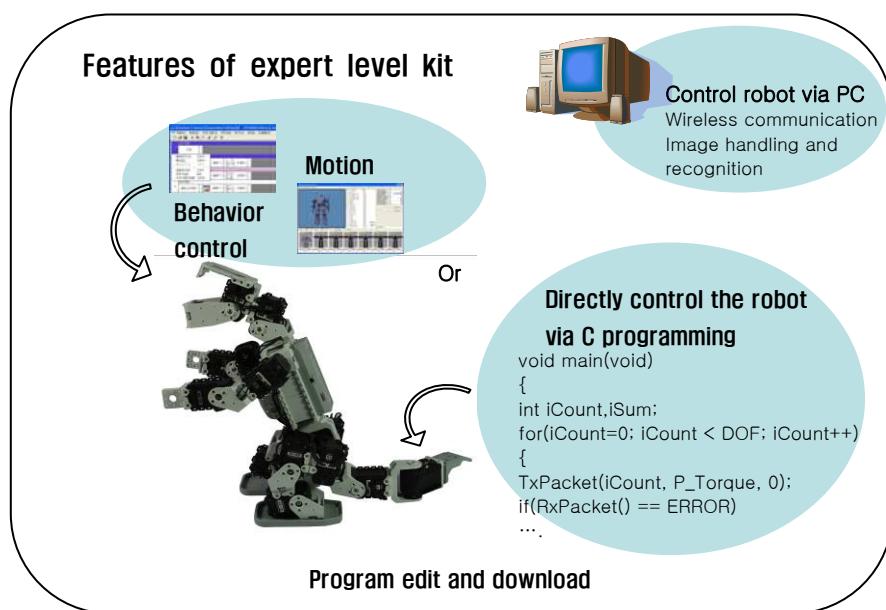
▷ Program edit and download

Users can directly create and edit the behavior control program and motion data, use C language to program the robot, or program under a Windows environment to control the robot via PC.

▷ Check the movements

Check whether the designed program is operating and that the robot is moving properly.

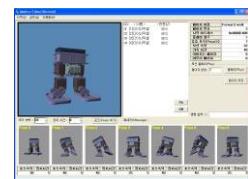
▷ Completion



1 – 3 – 2 . Lesson Contents Using Expert Level Education Kit

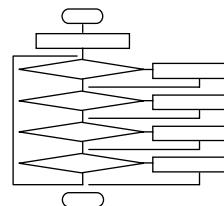
- Motion teaching and playback

Motion teaching and playback for a robot is a method that is frequently used on many industrial sites. Through this lesson, users can understand the concepts involved with the torque, position and speed controls.



- Improvement the logic and artificial intelligence through behavior control programming

Behavior control programming involves a process where users decide on how to execute the robot's functions. The basic components include: the robot's motion, sound and output recognition, object detection, button use, the remote control, temperature, and many others. By using the above elements and designing a visual diagram to execute the process, users can improve their robot's logic and intelligence.

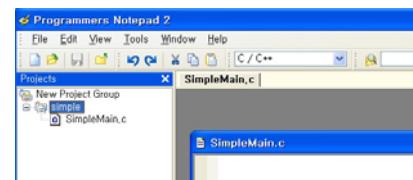


- Overview lesson on robot based microcontroller

In the Bioloid expert level educational kit, there are lessons on how to control the robots through on a microcontroller level. With the above lessons, users can understand the robot's circuit diagram, use the I/O port and A/D converter, and understand the underlying principles of the robot's internal devices and their processes.

- Programming the robot using the C language

In the Bioloid expert level education kit, there are lessons involving downloading and installing Ansi-C (open source program) and using the C language to program to control the robots. From these lessons, users can generate a binary file, learn to execute the file from embedded board, and construct the behavior control program algorithm through C language.



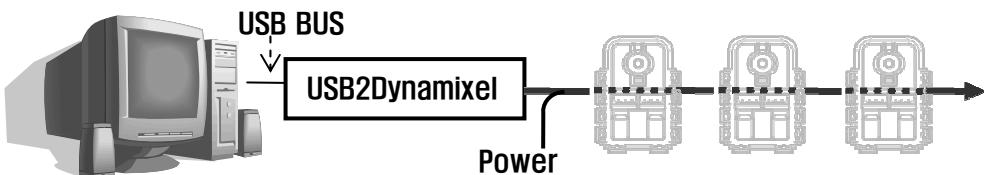
- Controlling the robot using wireless communication

From the Bioloid expert level educational kit, users can learn the basic concepts on PAN (Personal Area Network). By using Zig-100 they can also learn how to control robots via wireless communication from a PC or through remote control.



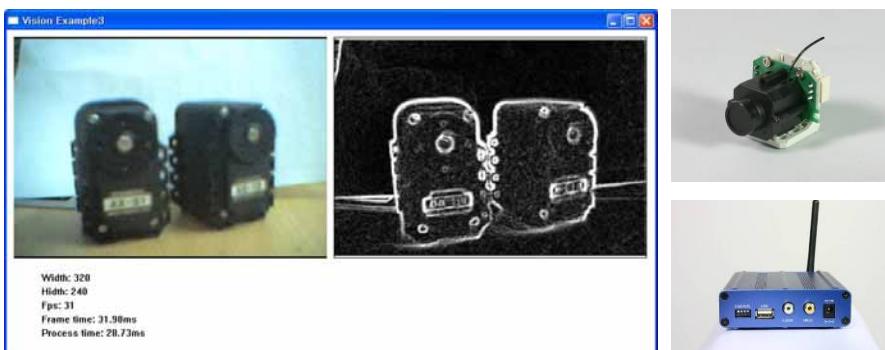
◎ Dynamixel control using a PC

From the Bioloid expert level educational kit, users can learn how to directly control Dynamixels from a PC. The picture below demonstrates how the USB2Dynamixel, a device included in the kit, can be utilized to directly control Dynamixels from PC.



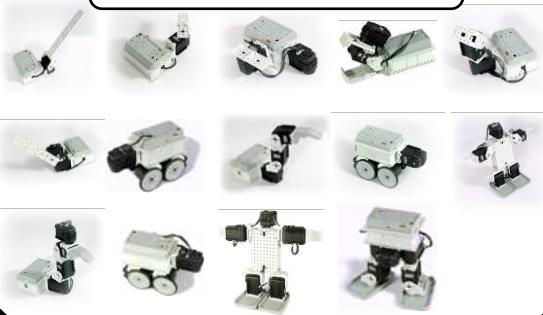
◎ Lessons on Vision

In the Bioloid expert level educational kit, users can learn how to receive an image data from a camera that is embedded in robot and display it in real-time on a PC's monitor through a USB port (image handling). Additionally, users can learn recognition routines (edge detection) by analyzing the received image data.



1 – 3 – 3 . Examples of an Expert Level Application Robot

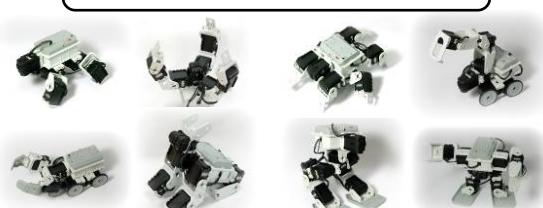
Beginner level robots



Beginner level robots: Robots with 4 or less joints that can be assembled with a Beginner kit.

Beginner kit

Intermediate level robots



Intermediate Level Robot: Robots with 8 or less joints that can be assembled with the Comprehensive kit. If you want to assemble intermediate level robots with a Beginner kit. You need to purchase four Dynamixels (AX-12) additionally.

Beginner kit

AX-12, 4 pcs

Comprehensive kit

Or

Beginner kit

AX-12, 14pcs

Frame set

Advanced level robots



Advanced Level Robots: Robot with 18 or less joints can be assembled with the Comprehensive kit. If you want to assemble an advanced level robot with the Beginners kit, you need to purchase a frame set and Dynamixels (AX-12) additionally.

Expert kit

In addition to all above, there are additional functions including wireless communication and image recognition, as well as C language program examples and libraries.

2 . Using the Bioloid Expert Level Educational Kit

2 – 1 . CM-5 BASED ROBOT PROGRAMMING

2 – 1 – 1 . Installing Development Resources

◎ Selecting a Compiler

The CPU on the CM-5 unit is the Atmega128 of the AVR Series from Atmel. There are many different C compilers available for the Atmega128, they are generally expensive. However, a global organization, called GNU, is distributing their compiler called GCC free of charge. For this reason, many research labs and institutions are using this compiler. The CM-5 unit also uses the AVR GCC compiler.

◎ Compiler and Editor

Windows OS users are familiar with compilers that have an editor function built in. However, for compilers that run on a text based OS, such as Linux, they usually have a separate compiler and editor. The GCC is a compiler based on the command line interface and thus does not have a built in editor. Therefore, users have to use a separate editor to develop a program. AVR-Edit and WIN-AVR are two popular editors for GCC. We will be using the WIN-AVR editor in this tutorial. WIN-AVR runs the compiler by internally calling the AVR GCC.

◎ Installing the Compiler

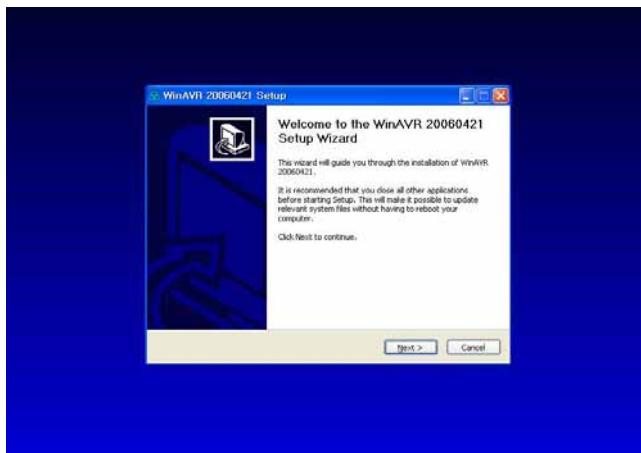
Let's install WinAVR, one of the editors from AVR GCC

Run "Install \ WinAVR \ WinAVR- 20060421-install.exe" from the expansion CD.

In Windows XP, if the warning appears click the "Run" button.



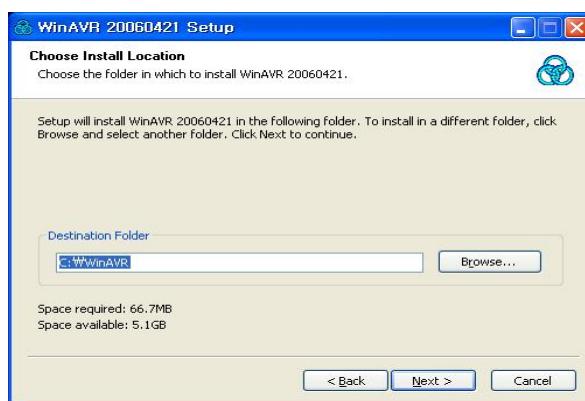
For the installation process, select your language.



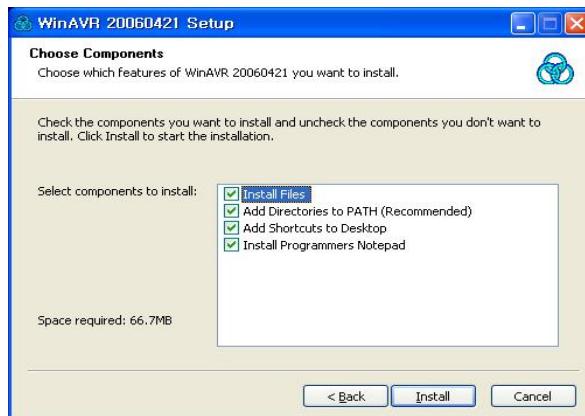
When you see above installation screen, click “Next” to continue.



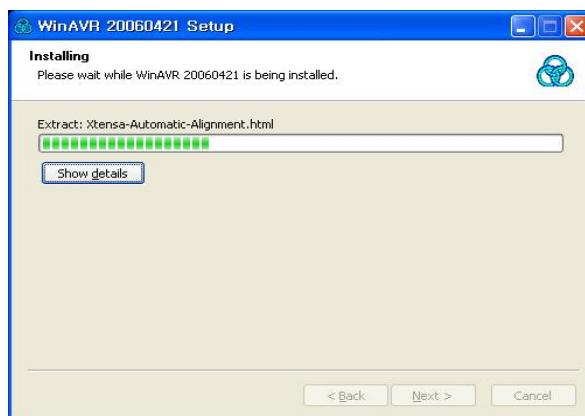
If you want to go on with the installation, you must click “I agree” for the license agreement.



Choose the folder to install WinAVR. You will need approximately 66.7 MB of hard disk space.



Select the components to install. You may choose all and click "Install" to continue.

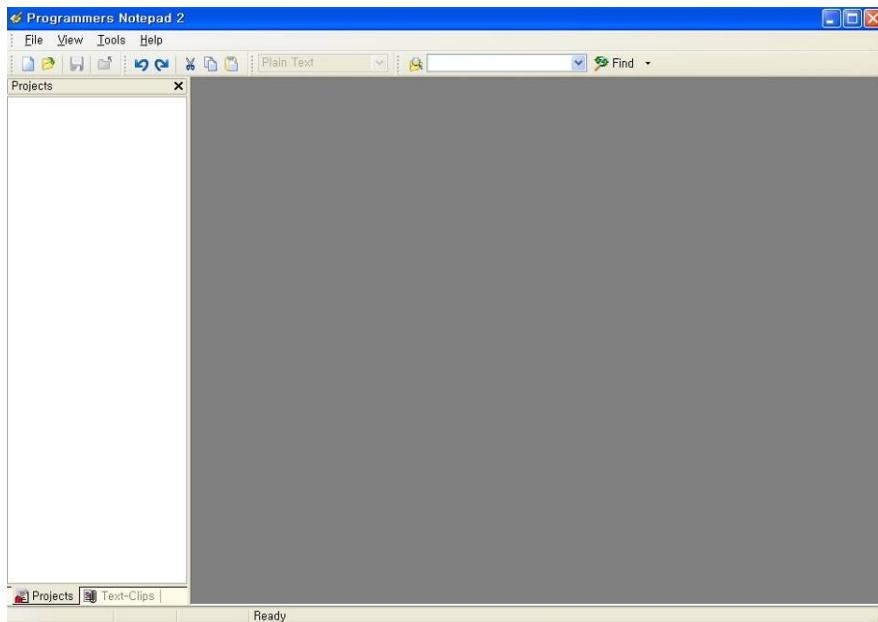
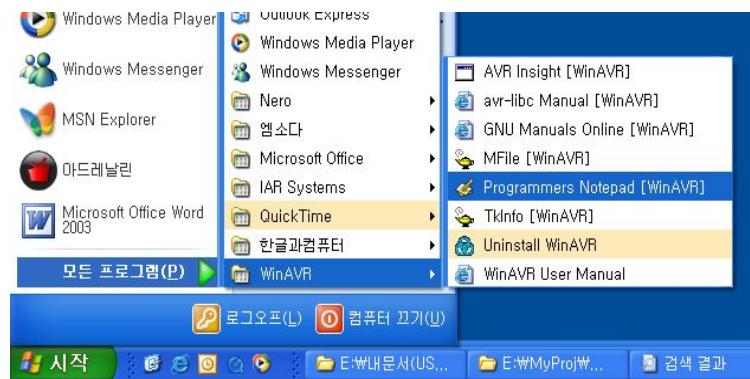


When the installation is complete, you will see the following screen.



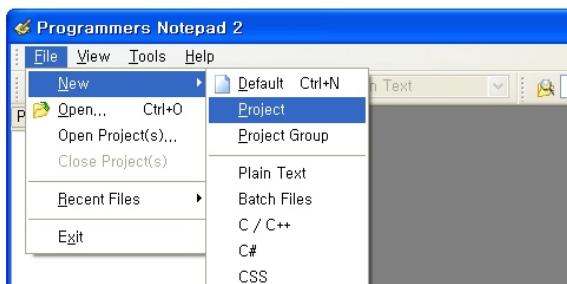
◎ Run the Win AVR

WinAVR is an editor program that calls the code within the GCC AVR Compiler. Let's select and run Programmer's Notepad [Win AVR] from WinAVR.



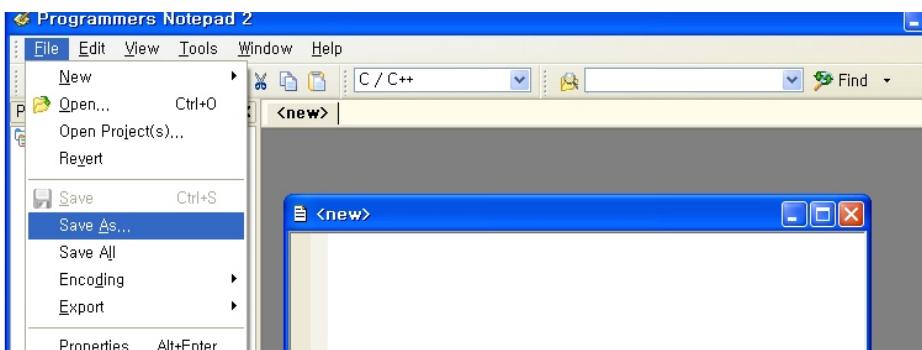
◎ Project File

When writing a large program, it is helpful to structure the program by dividing the source file into a number of smaller files according to its contents. The Project File is a high-level file that contains the list of all of the source files associated with the program that is being developed and includes all of the compile options. Open a new Project File as shown below and give it any name. Here, the project is named “Simple.” Select “Project” from the “New” menu item under the “File” menu.



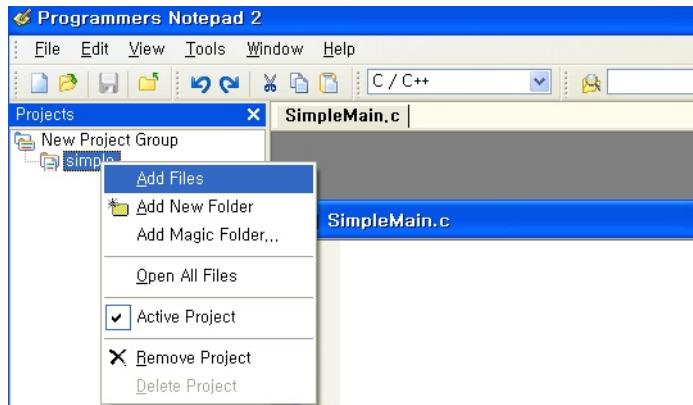
◎ C Source File

Next, we open the C source code file which is a lower-level file. Select “C/C++” from the “New” menu item under the “File” menu.

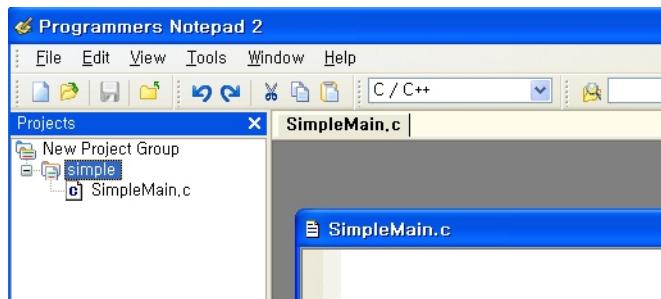


To assign a name, select “Save As...” from the “File” menu and give a name. Here, the C source file is named “SimpleMain.c.”

Next, the source file named “SimpleMain.c” needs to be added to the project file named “simple.” On the left side of the project window, right click on “simple” and click “Add Files” to select “SimpleMain.c”



Now, a project named “simple” is created which includes a source file called “SimpleMain.c.”



◎ Main()

Type the following in the “SimpleMain.c” source code.

```
void main(void)
{
}
```

The above program has no content and is in the form of the most basic structure of a C source code. Now that the source code is complete, the next step is to compile it. To do so, the user has to select the necessary options for compiling it.

◎ Makefile

The Makefile contains the information for the compile options. Sometimes the project file may contain the information for these options. However, since the Win-VAR does not have an internal compiler, it needs a separate Makefile for the GCC. Just like creating a project, the Makefile needs to be created only once and then modified as needed.

◎ Position of the Makefile

The Makefile has to be in the same directory (folder location) as the project file and the main source file that contains the Main function. The name of the file has to be Makefile without an extension and cannot be changed. Thus, the files “Makefile,” “Simple.pnproj,” and “SimpleMain.c” have to be in the same folder.

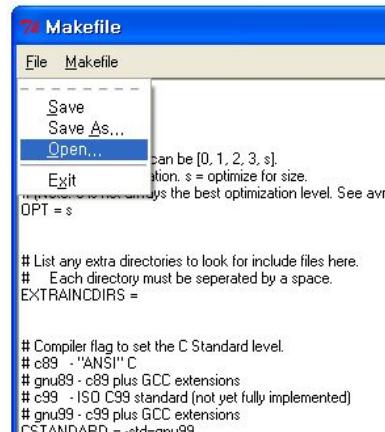
◎ Editing the Makefile

Makefile contains information on opening and compiling the source file and the name of the executable file. Makefile also contains other information, but the important information that the user needs to deal with are the name of the resultant file, the name of the source file, and its directory. This concept will become clearer once we go through the following tutorial. First, from the basic CD that came with the CM-5 unit, copy the Examples\C Program\Example\makefile to the current working directory folder.

◎ Running the mfile

As the picture shown on the right, run the “mfile” program in Win-AVR. This file only contains the editing function for Makefile.

First, open the file that you want to edit. There are two ways of editing the Makefile; the user can directly edit the contents of the Makefile, or the user can use the menu to edit it. To edit it using the menu, the user selects the “Makefile” menu on the right to change the options while the “mfile” is running. To directly edit the contents of the Makefile, the user selects the “Enable Editing of Makefile” under the “Makefile” menu to change options by using the keyboard.



◎ Editing Using the Menu

When a new project is created, two sections have to be modified in the Makefile; one is the main file name section, and the other is the C/C++ Source file(s) section. First set the name of the main file name to “simple.” This is used for the file names the compiler creates. Source codes can be added in the C/C++ Source file section. Edit the two sections of the Makefile as shown below.

```
# MCU name
MCU = atmega128

# Output format. (can be srec, ihex, binary)
FORMAT = ihex

# Target file name (without extension).
TARGET = simple

# List C source files here. (C dependencies are automatically generated.)
SRC = SimpleMain.c

# List Assembler source files here.
# Make them always end in a capital .S. Files ending in a lowercase .s
# will not be considered source files but generated files (assembler
# output from the compiler), and will be deleted upon "make clean"
# Even though the DOS/Win* filesystem matches both .s and .S the same,
# it will preserve the spelling of the filenames, and gcc itself does
# care about how the name is spelled on its command-line.
ASRC =
```

The Makefile can be edited by using the “Notepad” or any text editor.

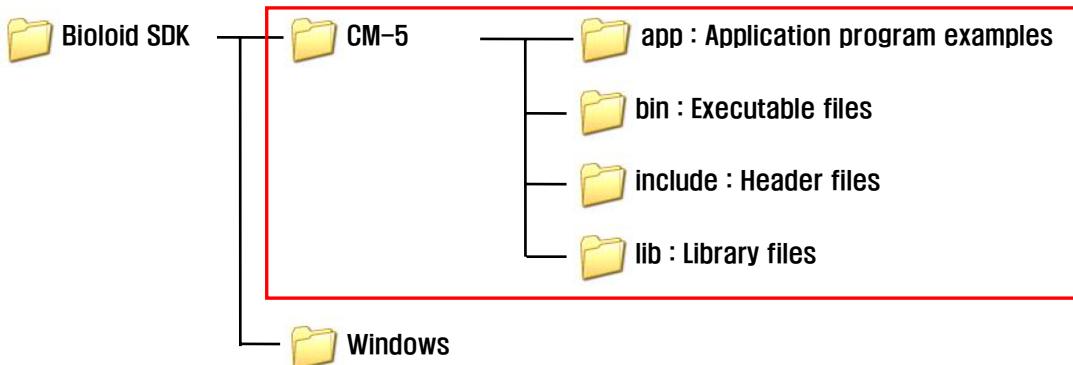
◎ Summary of Makefile

The concept of Makefile can be tricky for those who use GCC for the first time. The Makefile can be summarized with the following two concepts.

1. The Makefile has to be located in the same folder as the project file and source file. The name of the Makefile cannot be changed.
2. Within the Makefile, the source file section (SRC) and the resultant file section (TARGET) are modified as needed.

◎ Installing CM-5 library of the Bioloid SDK

To carry out the CM-5 based robot programming, users have to use CM-5 library. The library is provided as CD\CM-5\lib\libCM-5.a and it can be found in the Bioloid SDK\CM-5 folder of expansion CD.



To install, copy the Bioloid SDK folder from CD to a folder on your PC.

◎ Using the CM-5 library as an application program

First, change the makefile as shown below and link to libCM-5.a, a CM-5 library.

Add following item to #Additional libraries

```
ROBOTIS_LIB = -lCM-5 -L../../lib
```

Add following item to # Linker flags.

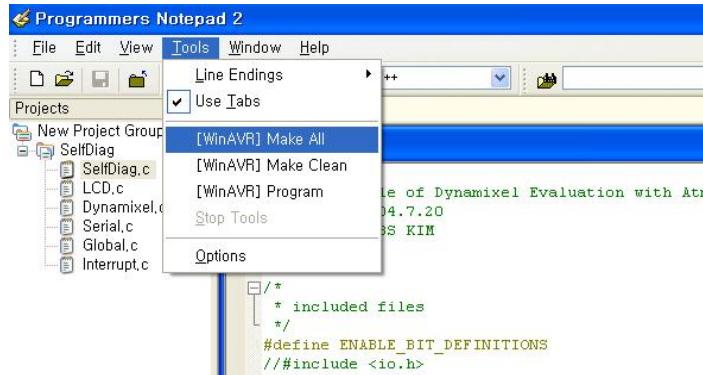
```
LDFLAGS += $(ROBOTIS_LIB)
```

For user program source, as shown below, it must include CM-5 library header file.

```
#include "../../include/libCM-5.h"
```

◎ Running the Compile

Select “Make All” from the “Tools” menu of the Programmers Notepad 2 [WinAVR].



The compile result message will appear at the bottom of the output window. If the compile was successful with no error, the “Errors: none” message will appear.

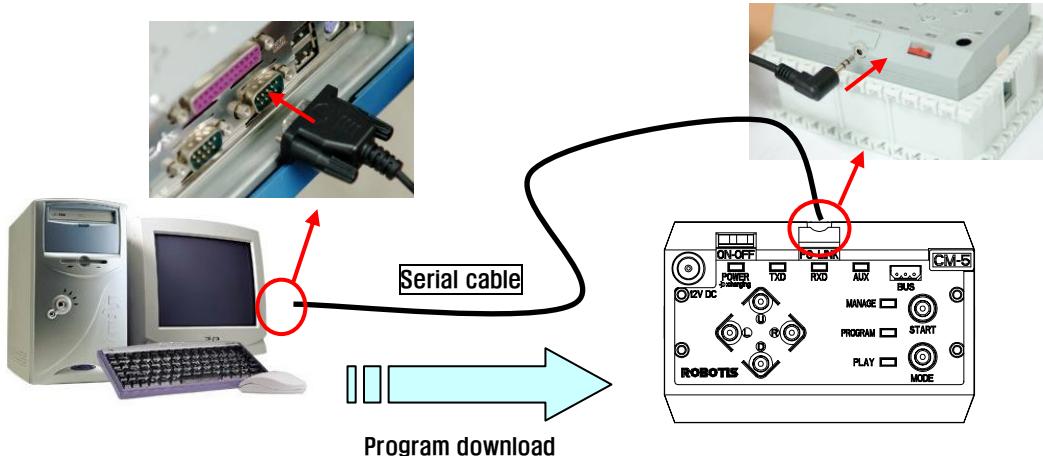
A screenshot of the Programmers Notepad 2 software focusing on the Output window. The window title is "Output". The output text shows the size and address of various sections of the compiled program:

section	size	addr
.data	3382	8388864
.text	7746	0
.bss	705	8392246
.noinit	0	8392951
.eprom	0	8454144
.stab	16716	0
.stabstr	4599	0
Total	33148	

Below this, the message "Errors: none" is displayed in red, followed by "----- end -----". At the very bottom, the message "> Process Exit Code: 0" is shown. The status bar at the bottom indicates "[1:1] : 760" and "Project file: d:\#work\#p".

◎ Setting up the communication environment

To operate a robot, you need to program the robot first. Transferring the robot program from PC to CM-5 is called “Download.”



* if you do not have a serial port, you can purchase USB2Serial and install it or use the USB2Dynamixel that is included in the expert level educational kit. USB2Serial is a device that allows USB port to be used as a serial port and can be purchased at local computer stores.

◎ USB2Dynamixel

USB2Dynamixel is included in the expert level educational kit, allowing users to use The Bioloid from a notebook or in PC that does not have a serial port. In addition to USB2Serial, it also has RS-485 and HDBRT communication functions.

In a case where PC does not have a serial port, USB2Dynamixel can be connected to a USB port and used as a serial port after installing the drivers. Follow the instructions below and set the switch to 232. (It is necessary when connecting PC and CM-5).

< Installation of USB2Dynamixel >

Comm. Mode	Use
TTL	AX Series Dynamixel Control
232	Used as USB2Serial
485	DX, RX Series Dynamixel Control



< Step 1 > When USBtoSerial is connected to PC, the following “New Hardware” dialog window will appear, here, users select “Install from a list of specific location.”



< Step 2 > After selecting search location, select Install/USB2Dynamixel folder from the expansion CD.

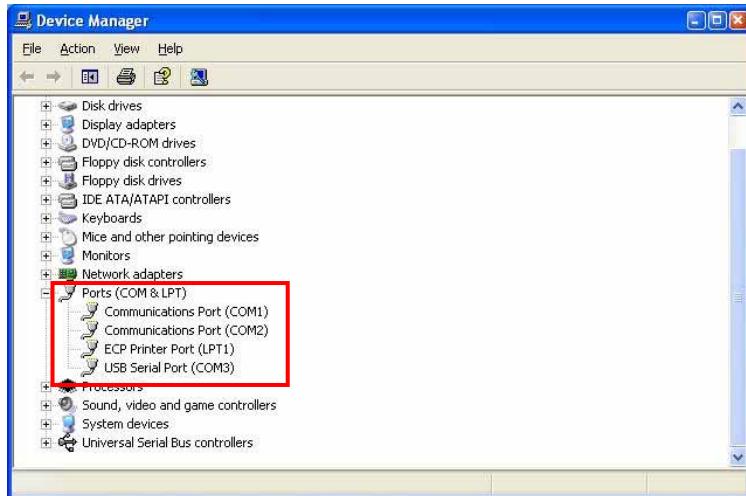
< Step 3 > Complete the installation.



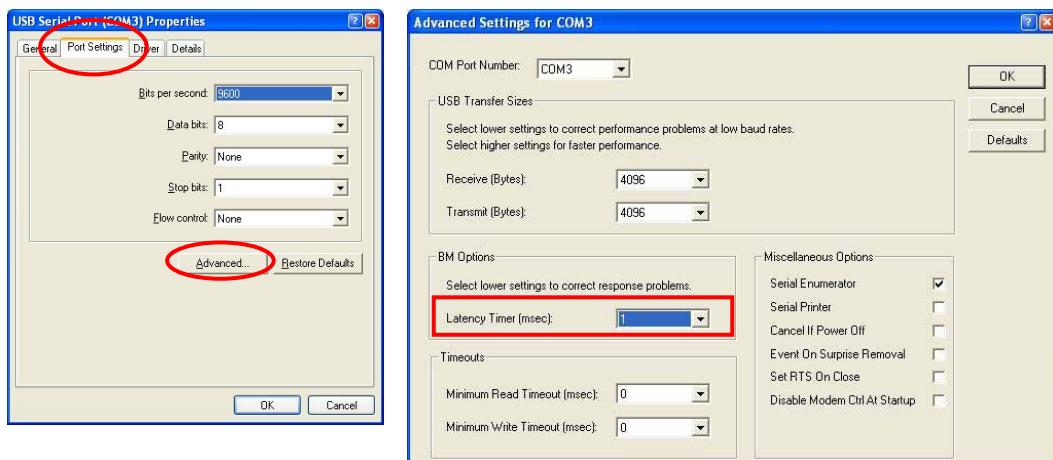
< Step 4 > Another “New Hardware” dialog window appears, repeat the Step 1~3 again.



< Step 5 > From the Device Manager, check where the USBtoSerial driver is installed. For communication connection later on, remember the Port Number (COM x)

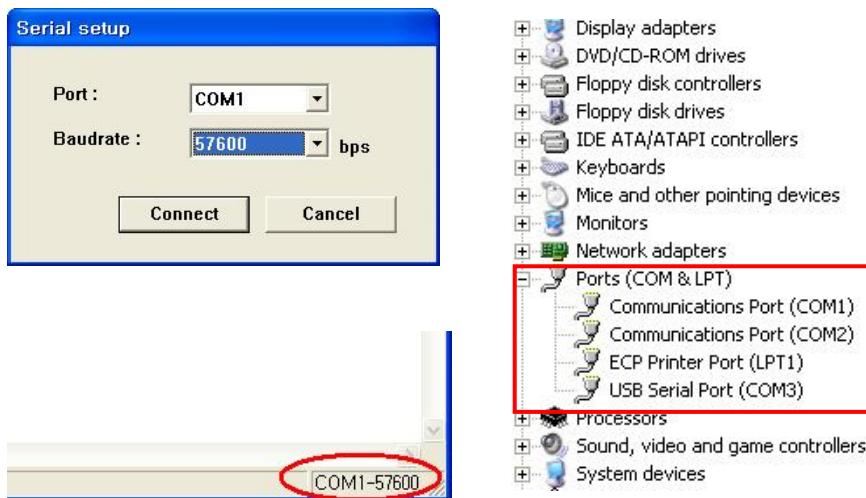


< Step 6 > After checking driver information, change “Latency Timer” to 1ms. This change allows for the fast response time.



◎ Robot Terminal

A terminal is a type of network that connects system with other system and in this case, terminal program is a program that enables connection with the system. A robot terminal is a program that connects CM-5 with PC and it allows users to input and output information using the PC. The information outputted from the CM-5 will go to the PC through the serial cable and then print on screen through the robot terminal. To connect the robot terminal with the CM-5, you must set the port and Baudrate. The port can be checked through Device Manager and Baudrate should remain 57600bps.



◎ Boot Loader

When power is applied to the CM-5 unit, the “CM Boot Loader” program in Reset Vector is executed. The “CM Boot Loader” program does not have as many functions as a PC operating system, but it has the following basic features: uploading and executing user created programs to the CM-5 unit memory, verifying the data in the memory, and downloading programs back to a PC. All numbers are treated in hexadecimal.

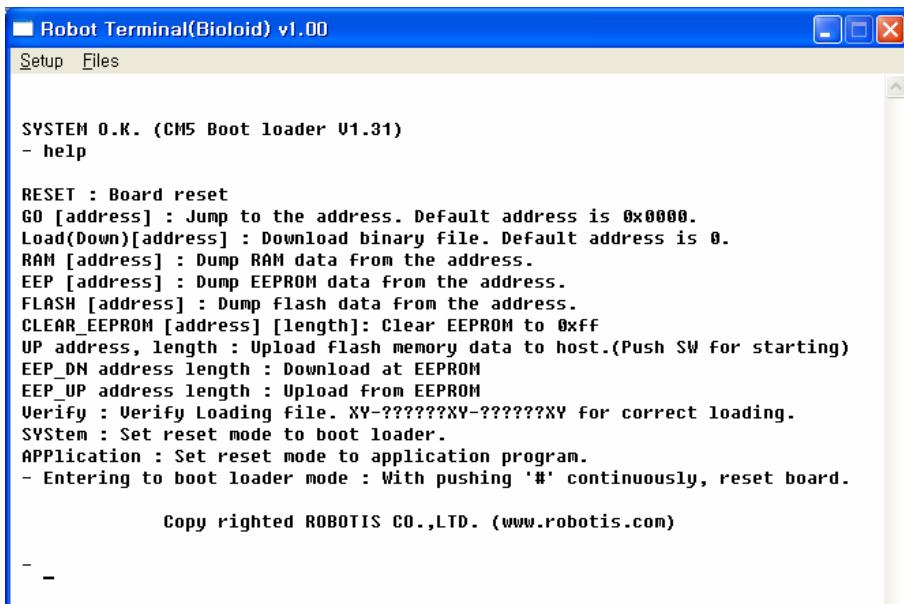
In order for you to download a program they have created, you need to execute the boot loader.

◎ Executing Boot Loader

There are two ways to execute the boot loader

- (1) From the robot terminal, press and hold the # key and turn on the CM-5’ s power switch.
- (2) From the robot terminal, press and hold the # key and press all switches of CM-5.

Next shows when the boot loader has been executed. Here, press Enter and then you will see the next screen.



The screenshot shows a Windows application window titled "Robot Terminal(Bioloid) v1.00". The menu bar has "Setup" and "Files". The main window displays the following text:

```
SYSTEM O.K. (CM5 Boot loader V1.31)
- help

RESET : Board reset
GO [address] : Jump to the address. Default address is 0x0000.
Load(Down)[address] : Download binary file. Default address is 0.
RAM [address] : Dump RAM data from the address.
EEP [address] : Dump EEPROM data from the address.
FLASH [address] : Dump flash data from the address.
CLEAR_EEPROM [address] [length]: Clear EEPROM to 0xFF
UP address, length : Upload flash memory data to host.(Push SW for starting)
EEP_DN address length : Download at EEPROM
EEP_UP address length : Upload from EEPROM
Verify : Verify Loading file. XY-??????XY-??????XY for correct loading.
SYstem : Set reset mode to boot loader.
APPLICATION : Set reset mode to application program.
- Entering to boot loader mode : With pushing '#' continuously, reset board.

Copy righted ROBOTIS CO.,LTD. (www.robotis.com)
```

The window has standard Windows controls (minimize, maximize, close) and a scroll bar on the right side.

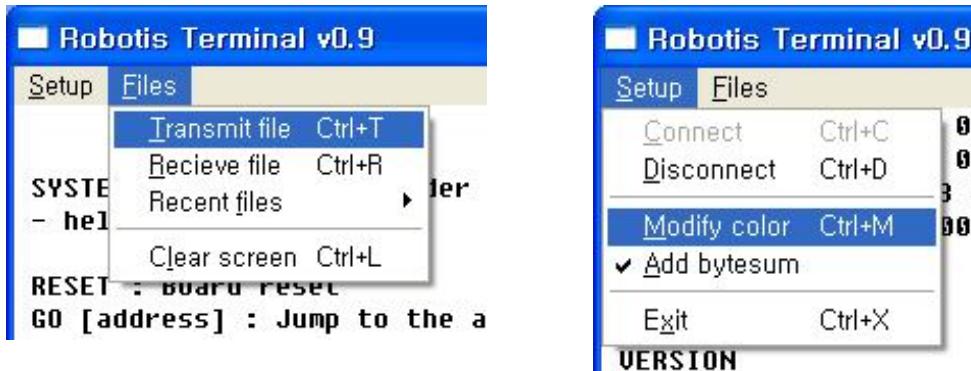
○ Program Download

Let's learn how to download a provided firmware or a program you have created onto the CM-5 unit. Let's try downloading a Bioloid CM-5 firmware.

Type in the command "load." The following message should appear. This message indicates that the data is ready to be written to address 0.

```
- load  
Write Address : 00000000  
Ready..
```

Next, select "Transmit file" in the "Files" menu from the Robot Terminal program as shown below. It is recommended to check the "Add bytesum" menu item in the "Setup" menu as shown below. If not, there will be a "Check Sum" error after download has been completed.



Select "Examples\ROM file\bioloid.hex" on the basic CD as the file to be transmitted. When the serial transmission is completed, as shown below, "Checksum:xx-xx" item will appear. If two numbers displayed are the same, in this case, "91," then there has been no error in serial transmission.

```
- ld  
Write Address : 00000000  
Ready..Success  
Rewriting:0X0006  
Size:0X000094A1 Checksum:91-91  
-
```

○ Run the program

When the "Start" button of the CM-5 is pressed, the downloaded program will execute.

◎ Memory Dump

The CM-5 unit not only has 128 Kbytes of flash memory, but also 4 Kbytes of RAM and 4 Kbytes EEPROM. There is a function in the CM-5 boot loader where you can check the contents in these memory spaces. Type in the memory type as the command followed by the address. The following figure is an example.

```
- ram 8
00000000 : B5 09 00 00 02 F4 37 00 00 94 00 00 00 00 03 00 .....7 .....
00000010 : 10 00 00 00 00 4D 02 2E 01 2E 0A 00 B5 09 1E 00 .....M .....
00000020 : 00 FB DC F8 00 00 00 00 00 00 18 62 00 00 00 2A .....b...*
00000030 : 04 00 00 07 00 07 01 01 01 00 00 00 00 FF FF 0F .....-
00000040 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....-
00000050 : 00 00 00 00 00 00 02 00 00 00 00 00 00 00 4F 09 95 .....0 ..
00000060 : 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 9D .....-
00000070 : 00 F8 FE FF 00 00 00 00 00 00 00 00 20 00 00 00 ......

- eeprom 100
00000100 : FF .....-
00000110 : FF .....-
00000120 : FF .....-
00000130 : FF .....-
00000140 : FF .....-
00000150 : FF .....-
00000160 : FF .....-
00000170 : FF .....-

- Flash 1e000
0001E000 : 0C 94 48 F9 18 95 18 95 18 95 18 95 18 95 18 95 ..H.....
0001E010 : 18 95 18 95 18 95 18 95 18 95 18 95 18 95 18 95 .....-
0001E020 : 18 95 18 95 18 95 18 95 18 95 18 95 18 95 18 95 .....-
0001E030 : 18 95 18 95 18 95 18 95 18 95 18 95 18 95 18 95 .....-
0001E040 : 18 95 18 95 18 95 18 95 18 95 18 95 18 95 18 95 .....-
0001E050 : 18 95 18 95 18 95 18 95 18 95 18 95 18 95 18 95 .....-
0001E060 : 18 95 18 95 18 95 18 95 18 95 18 95 18 95 18 95 .....-
0001E070 : 18 95 18 95 18 95 0C 94 2C F1 18 95 18 95 .....
```

2-1-2. CM-5 CONTROL

◎ Serial Communication

There are two serial ports (UART) on the CM-5, namely “0” and “1.” The “0” port is connected to the HDBRT (Half-Duplex Buffer Rx/Tx) circuit to communicate with the Dynamixel and the “1” port is connected to either the PC or a Zigbee communication module.

In this chapter, we will introduce the library functions that control the serial port of the CM-5.

To use the UART, execute “SerialInitialize” to initialize the serial port and then use the functions provided by the library as required. For a detailed explanation of each function, refer to the CM-5 library reference in the appendix, at the end of this manual.

A function that initializes the serial port is shown below.

```
void SerialInitialize(byte bPort, byte bBaudrate, byte bInterrupt);
```

Functions related to UART 0 are shown below.

```
/* UART 0 Functions */

void TxD485(byte bTxdData);           // Transfer 1 byte to Dynamixel 1 by HDBRT
void TxD0Byte(byte bTxdData);          // Transfer 1 byte by UART 0
byte RxD0Byte(void);                 // Receive data by means of UART 0 (polling)

volatile byte gbpRxInterruptBuffer[256]; // UART 0 Rx Interrupt Buffer
volatile byte gbRxBufferWritePointer;   // Ring-Queue Head
byte gbRxBufferReadPointer;           // Ring-Queue Tail
```

The CM-5 library uses interrupts of RxD as part of UART 0 and UART 1, and uses a successive saving method for data that comes into Ring Queue type buffer. Therefore, even though the RxD0Byte function is provided, in a case when the interrupt is not used, the RxD0Byte function is rarely used.

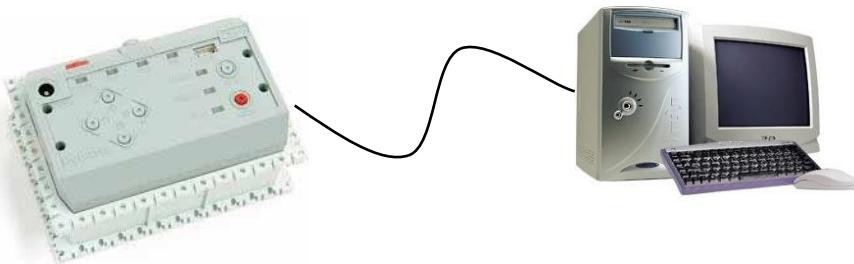
The followings are functions related to UART 1. With UART 1 connected to the PC, there are numerous functions that can check for variables for debugging purpose.

```
/* UART 1 Functions */

void TxD1Byte(byte bTxdData);           // Transfer 1 byte to PC via UART 1.
void TxDByte16(byte bSentData);         // Transfer 1 byte as a hexadecimal.
void TxDByte10(byte bByte);             // Transfer 1 byte as a decimal.
void TxDWord16(word wSentData);         // Transfer 2 bytes as a hexadecimal
void TxDWord10(word wData);              // Transfer 2 bytes as a decimal.
void TxDLong16(long lSentData);          // Transfer 4 bytes as a hexadecimal
void TxDLong10(long lLong);              // Transfer 4 bytes as a decimal.
void TxD8DecHex(byte bByte);             // Transfer 1 byte as a decimal and hexadecimal.
void TxDString(byte *bData);             // Transfer character string
byte RxD8Interrupt(void);               // From UART 1 Interrupt Buffer reads 1 byte.

volatile byte gbpPacketDataBuffer[16+1];   // UART 1 Rx Interrupt Buffer
volatile byte gbPacketWritePointer;        // Ring-Queue Head
volatile byte gbPacketReadPointer;          // Ring-Queue Tail
```

Connect your hardware as shown below.



[Example] Let's construct program that creates echo when the PC sends data.

This example can be found in the "Bioloid SDK \CM-5 \app \example_serialcomm \example_serialcomm.c" part of the expansion CD.

```
#include "../../include/libCM-5.h"

int main(void)
{
    PortInitialize();                                // Port Initialize
    TimerInitialize(TIMER0, 128, 1);                // Timer 0 Initialize
    SerialInitialize(SERIAL_PORT1,DEFAULT_BAUD_RATE,RX_INTERRUPT); //232
    sei();                                         // Enable Interrupt

    char bKeyin;                                    // Input Key variable

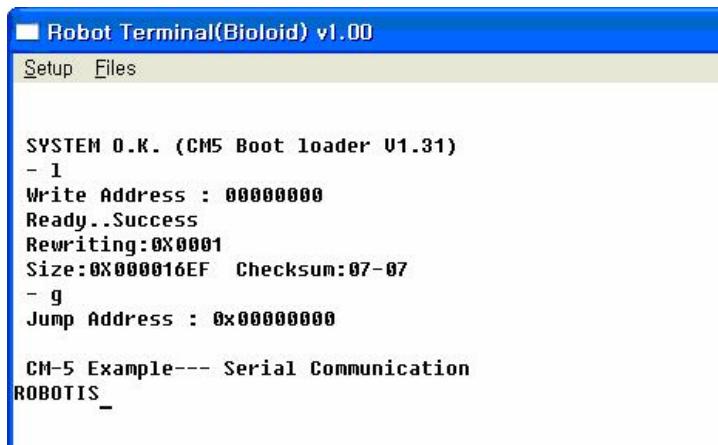
    TxDString("\r\n CM-5 Example--- Serial Communication\r\n");

    while(1)
    {
        bKeyin = RxD8Interrupt();
        TxD1Byte( bKeyin );
    }

    return 0;
}
```

From the “while” loop of main through the RxD8Interrupt, the inputted value of “bKeyin” is retransmitted as a “TxD1Byte.”

When this code executed, text typed on the PC’s keyboard appears on the screen as the echoed data.



◎ Dynamixel Network Communication.

The CM-5 can communicate with Dynamixel units through the HDBRT (Half-Duplex Buffer Rx/Tx) of UART 0. By sending a command packet you can control a Dynamixel unit, and check various values including ID, communication speed, current position, temperature, voltage, and others from the control table. Additionally, you can control motors by changing the moving speed, current position, maximum torque, and others. To do this, you must communicate with Dynamixel units network and by sending packet header, ID, command, parameter, and checksum and it must be set so that it can immediately receive return packet.

Various libraries are provided so that you can easily control Dynamixel units through a network. For a detailed explanation of each function, check CM-5 library reference located in the appendix, which is in the latter part of this manual.

Next is a macro used for controlling the communication direction of HDBRT.

```
#define HDBRT_TXD    cbi(PORTE,3), sbi(PORTE,2)          // HDBRT TxD Mode Macro  
#define HDBRT_RXD    cbi(PORTE,2), sbi(PORTE,3)          // HDBRT RxD Mode Macro
```

By using HDBRT_TXD and HDBRT_RXD macros, you can directly change the HDBRT's communication direction. However, since the included libraries can also change the direction, unless it is specifically required for another purpose, it is recommended that you do not use these macros.

Next is the basic library required for network communication with Dynamixel units.

```
byte RxPacket(byte bRxLength);                                // receives  
return packet  
byte TxPacket(byte bID, byte bInstruction, byte bParameterLength); // creates  
packet, transmits  
void PrintBuffer(byte *bpPrintBuffer, byte bLength); // Outputs the contents of  
buffer in terminal as a hexadecimal  
  
/*volatile*/ byte gbpRxBuffer[180]; // Buffer where return packet is saved  
/*volatile*/ byte gbpTxBuffer[140]; // Buffer where entire packet is saved upon Tx  
Packet creation  
volatile byte gbpParameter[130]; // Buffer where parameter is saved upon Tx  
Packet creation  
byte gbErrorPrint,gbPacketPrint; // Variable that determines the error message  
output
```

The code calculates the header and checksum necessary in the TxPacket, creates the entire packet, and transfers to the HDBRT network, whereas the RxPacket checks for error after receiving a return packet. A PrintBuffer is a function for debugging purposes and distributes the contents of a buffer to terminals.

Assemble the hardware environment as shown below.



[Example] Let's create a program that reads the model numbers from a control table from Dynamixel units having ID 1 and ID 100, and then prints the results on the screen.

This example can be found in "Bioloid SDK \CM-5 \app \ example_networkwithDynamixel \example_networkwithDynamixel.c" of expansion CD.

```
#include "../../include/libCM-5.h"

int main(void)
{
    PortInitialize();
    TimerInitialize(TIMER0, 128, 1); //Interrupt Enable
    SerialInitialize(SERIAL_PORT0,1,RX_INTERRUPT); //485
    SerialInitialize(SERIAL_PORT1,DEFAULT_BAUD_RATE,RX_INTERRUPT); //232
    sei();

    TxDString("\r\n CM-5 Example--- Network Communication with Dynamixel\r\n");
}
```

```
byte bID, bTxPacketLength, bRxPacketLength;

TxDString("\r\n*AX-12 ID 1");
bID = 1;
gbpParameter[0] = P_MODEL_NUMBER_L; //Address of Firmware Version
gbpParameter[1] = 1; //Read Length
bTxPacketLength = TxPacket(bID,INST_READ,2);
bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1]);
TxDString("\r\n TxD:"); PrintBuffer(gbpTxBUFFER,bTxPacketLength);
TxDString("\r\n RxD:"); PrintBuffer(gbpRxBUFFER,bRxPacketLength);
TxDString("\r\n Model Number: ");
TxDByte10( gbpRxBUFFER[5] );

TxDString("\r\n*AX-S1 ID 100");
bID = 100;
gbpParameter[0] = P_MODEL_NUMBER_L; //Address of Firmware Version
gbpParameter[1] = 1; //Read Length
bTxPacketLength = TxPacket(bID,INST_READ,2);
bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1]);
TxDString("\r\n TxD:"); PrintBuffer(gbpTxBUFFER,bTxPacketLength);
TxDString("\r\n RxD:"); PrintBuffer(gbpRxBUFFER,bRxPacketLength);
TxDString("\r\n Model Number: ");
TxDByte10( gbpRxBUFFER[5] );

while(1);
return 0;
}
```

Through this example, users can understand how to use the most basic functions: TxPacket, RxPacket, and PrintBuffer. When the code is executed, as the picture below illustrates, by reading the Model Number of AX-12 and AX-S1, where their ID is 1 and 100 respectively, it prints the details, from start to finish, on the terminal screen.

The screenshot shows the Robot Terminal (Bioloid) v1.00 interface. The window title is "Robot Terminal(Bioloid) v1.00". The menu bar has "Setup" and "Files". The main area displays the following text:

```

SYSTEM O.K. (CMS Boot loader V1.31)
- 1
Write Address : 00000000
Ready..Success
Rewriting:0X0001
Size:0X00017FD Checksum:38-38
- g
Jump Address : 0x00000000

CM-5 Example--- Network Communication with Dynamixel

*AX-12 ID 1
Tx:D:FF FF 01 04 02 00 01 F7 (LEN:008(0x08))
Rx:D:FF FF 01 03 01 0C EE (LEN:007(0x07))
Model Number: 012

*AX-S1 ID 100
Tx:D:FF FF 64 04 02 00 01 94 (LEN:008(0x08))
Rx:D:FF FF 64 03 00 0D 8B (LEN:007(0x07))
Model Number: 013

```

◎ LED Access

The CM-5 has a total of 7 LEDs, including Power, TxD, RxD, AUX, MANAGE, PROGRAM, and PLAY. A library is provided so that you can control the LEDs as needed.

```

void LEDOn(byte bLed);           // Turn on LED (Factor is derived from Bit operation)
void LEDOff(byte bLed);          // Turn off LED (Factor is derived from Bit operation)

```

Factors that are included in the above functions, display 7 LEDs by a bit. It is shown below.

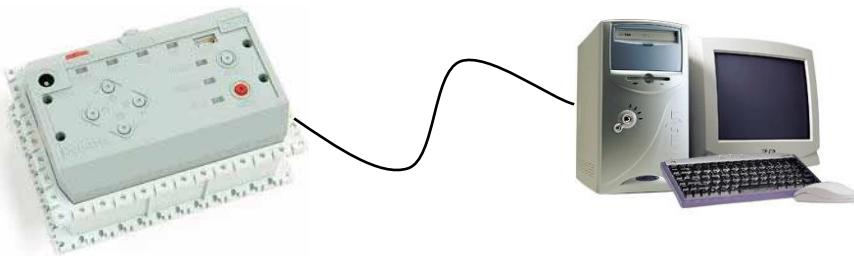
Value	Meaning
BIT_LED_PWR	POWER LED
BIT_LED_TXD	TxD LED
BIT_LED_RXD	RxD LED
BIT_LED_AUX	AUX LED
BIT_LED_MANA	MANAGE LED
BIT_LED_PROG	PROGRAM LED
BIT_LED_PLAY	PLAY LED
BIT_LED_ALL	ALL LED

Combines required LED by OR operation, and as a factor included in the LEDOn/LEDOff function.

(ex) When turning on AUX LED and MANAGE LED: LEDOn (BIT_LED_AUX | BIT_LED_MANA);
When turning off all LED: LEDOff (BIT_LED_ALL);

When TimerInitialize is executed, POWER LED, TXD and RXD LED are automatically controlled in the interrupt service routine. [TXD and RXD instantaneously blink upon data input and output, and the power LED flickers on and off when charging]

Construct the hardware environment as shown below.



[Example] Let's create a program where various LEDs turn on and off alternatively.

This example can be found in the "Bioloid SDK \CM-5 \app \example_led \example_led.c" part of the expansion CD.

```
#include "../../include/libCM-5.h"

int main(void)
{
    PortInitialize();
    SerialInitialize(SERIAL_PORT1,DEFAULT_BAUD_RATE,RX_INTERRUPT); //232
    sei();

    TxDString("\r\n CM-5 Example--- LED Test\r\n");

    while(1)
    {
        LEDOn ( BIT_LED_AUX );
        MiliSec(1000);
        LEDOff( BIT_LED_AUX );

        LEDOn ( BIT_LED_MANA | BIT_LED_PROG | BIT_LED_PLAY );
        MiliSec(1000);
        LEDOff( BIT_LED_MANA | BIT_LED_PROG | BIT_LED_PLAY );

        LEDOn ( BIT_LED_TXD | BIT_LED_RXD | BIT_LED_AUX | BIT_LED_MANA | BIT_LED_PROG
| BIT_LED_PLAY );
    }
}
```

```
MiliSec(1000);  
LEDOFF( BIT_LED_TXD | BIT_LED_RXD | BIT_LED_AUX | BIT_LED_MANA | BIT_LED_PROG  
| BIT_LED_PLAY );  
}  
  
return 0;  
}
```



When this program is executed, the AUX LED will be turned on for 1 second, followed by the MANAGE, PROGRAM, and PLAY LED, and then, except for the POWER LED, 6 LEDs will flicker on and off. [After the program starts, the POWER LED will remain lit]

◎ Button Access

The CM-5 has six buttons: Up, Down, Left, Right direction buttons as well as the Start and Mode Change buttons.

Among the above buttons, the Mode Change button is a reset button and separate programming is not possible. However, through the CM-5 libraries, the five other buttons can be used and programmed.

```
byte GetButtonState(void); // Its current button state is being inputted.  
byte WaitButtonChange(void); // Waits until button is inputted and then returns  
the changed state
```

For the above functions, excluding the Mode button, the five buttons return the current state value. When the button is pressed, the bit for that button becomes 1 and the values for each button are shown as follows.

Value	Meaning
BIT_SW_RT	Right Direction Button
BIT_SW_LF	Left Direction Button
BIT_SW_DN	Down Direction Button
BIT_SW_UP	Up Direction Button
BIT_SW_START	Start Button

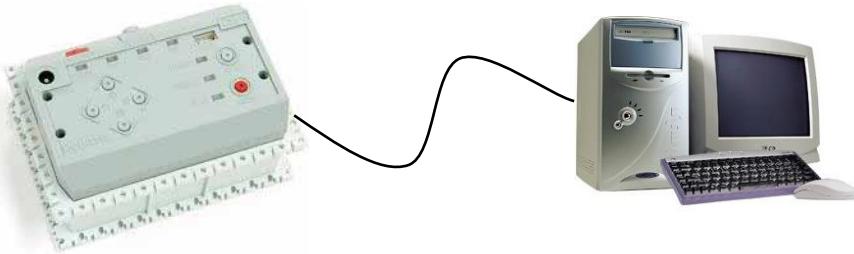
As shown below, simple macros are also possible.

Since a specified button returns the value of 1 when it is pressed, you can use the “If” statement to create conditional programming that depends on whether or not the button is pushed.

```
#define PUSH_SW_START      ( GetButtonState() & BIT_SW_START )  
#define PUSH_SW_UP         ( GetButtonState() & BIT_SW_UP )  
#define PUSH_SW_DN         ( GetButtonState() & BIT_SW_DN )  
#define PUSH_SW_LF         ( GetButtonState() & BIT_SW_LF )  
#define PUSH_SW_RT         ( GetButtonState() & BIT_SW_RT )
```

(ex) When the Start button is pressed, the “if” statement would read: if (PUSH_SW_START) { }

Construct the hardware environment as shown below.



Let's create a program where you can see which button has been pressed by outputting the results on the PC screen through serial communication. This example can be found in the directory "Bioloid SDK \CM-5 \app \example_button \example_button.c" on the expansion CD.

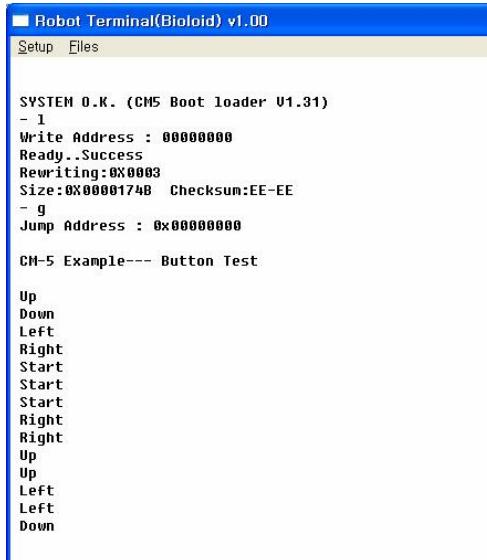
```
#include "../../include/libCM-5.h"

int main(void)
{
    PortInitialize();
    TimerInitialize(TIMER0, 128, 1); //Interrupt Enable
    SerialInitialize(SERIAL_PORT1,DEFAULT_BAUD_RATE,RX_INTERRUPT); //232
    sei();

    TxDString("\r\n CM-5 Example--- Button Test\r\n");

    while(1) {
        switch ( WaitButtonChange() ) {
            case BIT_SW_RT:
                TxDString( "\r\n Right" ); break;
            case BIT_SW_LF:
                TxDString( "\r\n Left" ); break;
            case BIT_SW_DN:
                TxDString( "\r\n Down" ); break;
            case BIT_SW_UP:
                TxDString( "\r\n Up" ); break;
            case BIT_SW_START:
                TxDString( "\r\n Start" ); break;
        }
    }
    return 0;
}
```

As shown in the picture, GetButtonState immediately reads the current state of a button, whereas WaitButtonChange waits for the button to change the state, allowing you to construct a simple program, similar to example programs. If the you download the above examples and execute them, they can immediately check which buttons are pressed.



```
Robot Terminal(Bioloid) v1.00
Setup Files

SYSTEM O.K. (CMS Boot loader V1.31)
- 1
Write Address : 00000000
Ready.Success
Rewriting:0X0003
Size:0X0000174B Checksum:EE-EE
- 9
Jump Address : 0x00000000

CH-5 Example--- Button Test

Up
Down
Left
Right
Start
Start
Start
Right
Right
Up
Up
Left
Left
Down
```

◎ Battery Charge

The CM-5 has the capability to recharge its battery. However, the following conditions apply:

1. You must use nickel metal hydrate (NiMH) batteries.
2. Use 8 AA size, 1.2V in series to total 9.6V.
3. The batteries must be closely adhered to the CM-5 and must be shielded so that they are protected from outside temperature.
4. The power adapter must provide enough current for a constant voltage (SMPS) of 12VDC.

The best setup that satisfies above conditions is when you use CM-5 dedicated battery that is included in the kit. [In other words, if you do not use other batteries or adapters than those provided in the kit]

When the above conditions are met, users can construct a simple charge function by using the charge function provided from the CM-5 library.

```
void ChargeInInterruptInitialize (void); // Initialize charge interrupt
```

Only one function is required and since the charge process is basically done with the interrupt, you only have to initialize the charge interrupt. However, TimerInitialize must be initialized beforehand. Since the charging part uses Timer 1, you do not need to take precaution. [cf. Main timer uses Timer 0 and the charge routine uses Timer 1; and thus, you cannot use Timer 0 and 1 for other purposes].

After initializing the charge interrupt, to carry out the charge, change the variable as shown below.

```
byte gbBatteryChargeMode; // Charged state
```

This variable indicates the current charging state and initially is in the “not charged” state. The following shows the various charge states.

Value	Meaning
BATTERY_CHARGE_MODE_NONE	Not Charged
BATTERY_CHARGE_MODE_ING	Charging (Charge Started)
BATTERY_CHARGE_MODE_ERROR	Error While Charging
BATTERY_CHARGE_MODE_FULL	Charging Completed

Accordingly, to start the charge,

```
gbBatteryChargeMode = BATTERY_CHARGE_MODE_ING;
```

With the simple code above, the charging process is initialized. Once the charging starts, the CM-5 automatically monitors the battery's voltage, temperature, and others, so you do not have to configure anything else. The charging state can be checked through the Power LED. [The power LED's light will blink faster as the charge is near complete. Once it is fully charged, the light will blink very slowly.]

Construct the hardware environment as shown below.



[Example] Let's create a program where users can charge the battery. This example can be found in the director "Bioloid SDK \CM-5 \app \example_batterycharge \example_batterycharge.c" in the expansion CD.

```
#include "../../include/libCM-5.h"

int main(void)
{
    PortInitialize();
    TimerInitialize(TIMER0, 128, 1); //Interrupt Enable
    SerialInitialize(SERIAL_PORT1,DEFAULT_BAUD_RATE,RX_INTERRUPT); //232
    ChargeInInterruptInitialize();
```

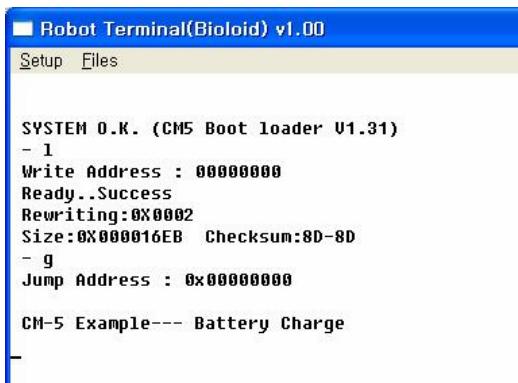
```
sei();

TxDString( "\r\n CM-5 Example--- Battery Charge\r\n" );

gbBatteryChargeMode = BATTERY_CHARGE_MODE_ING;
while(1);

return 0;
}
```

As shown above, the TimerInitialize function must be executed before running ChargeInInterruptInitialize. By simply changing the gbBatteryChargeMode to BATTERY_CHARGE_MODE_ING, you can begin the charging process.



2 – 1 – 3. Dynamixel Control

◎ Using AX-12

As mentioned previously, the CM-5, through the HDBRT (Half-Duplex Buffer Rx/Tx) circuit of UART 0, communicates with Dynamixel units and can read various Control Table values, such as a Dynamixel unit's ID, communication speed, current position, temperature, and voltage, in addition to controlling the motor by adjusting the moving speed, current position, maximum torque, etc.

This kit provides various libraries so that you can freely use Dynamixel units by communicating with the Dynamixel network. Previously, through the examples, you learned the details of TxPacket and RxPacket, but here, you will use more advanced functions to control the Dynamixel units, in particular, AX-12, an actuator module.

The following are the various libraries that can be used to control the Dynamixel units. The most basic are the TxPacket, RxPacket, and functions that were created using them, which are introduced next.

```
byte TxRxPacket(byte bID, byte bInst, byte bTxParaLen);
                                // Function that combined TxPacket and
RxPacket

byte ReadByte(byte bID, byte bAddress);           // Reading 1 byte using TxRxPacket
word ReadWord(byte bID, byte bAddress);          // Reading 2 bytes using TxRxPacket

byte WriteByte(byte bID, byte bAddress, byte bData); // Writing 1 byte using
TxRxPacket
byte WriteWord(byte bID, byte bAddress, word wData); // Writing 2 bytes using
TxRxPacket
```

TxRxPacket is a combined TxPacket and RxPacket function that integrates the process of sending command packets and return packets into one function. The method to use this function is the same as the TxPacket and allows you more convenience than using the TxPacket and RxPacket separately.

Moreover, by using the TxRxPacket, the ReadByte, ReadWord, WriteByte, and WriteWord, can read and write 1 or 2 bytes data from the Control Table, making it ideal for sending and receiving data that is less than 2 bytes.

Let's see what's inside the Dynamixel unit's Control Table.

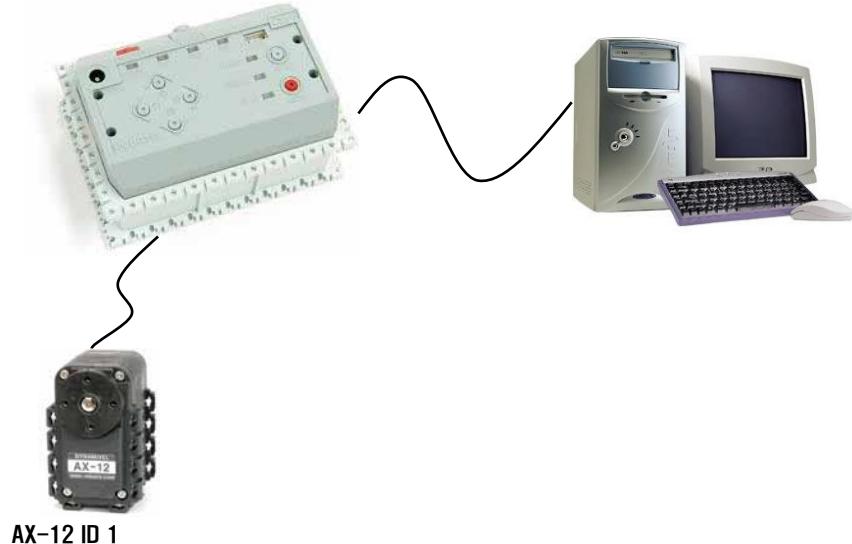
The Control Table is divided into EEPROM and RAM. Values saved in the EEPROM range will remain, even when the power is turned off and on. Conversely, in the RAM range, some values become their initial value and other values become copied values of the EEPROM range. For more details, refer to either AX-12 or AX-S1 manual.

The following table shows the elements of the Control Table.

		Value	Meaning
EEPROM Range	Common	P_MODEL_NUMBER_L	Low byte of model number
		P_MODEL_NUMBER_H	High byte of model number
		P_VERSION	Firmware version information
		P_ID	Dynamixel ID
		P_BAUD_RATE	Dynamixel comm. speed(Baud rate)
		P_RETURN_DELAY_TIME	Return delay time
		P_LIMIT_TEMPERATURE	Internal limit temperature
		P_DOWN_LIMIT_VOLTAGE	Minimum limit voltage
		P_UP_LIMIT_VOLTAGE	Maximum limit voltage
		P_RETURN_LEVEL	Return level
AX-12	AX-12	P_CW_ANGLE_LIMIT_L	Low byte of clockwise angle limit value
		P_CW_ANGLE_LIMIT_H	High byte of clockwise angle limit value
		P_CCW_ANGLE_LIMIT_L	Low byte of counter-clockwise angle limit value
		P_CCW_ANGLE_LIMIT_H	High byte of counter-clockwise angle limit value
	AX-S1	P_MAX_TORQUE_L	High byte of torque limit value
		P_MAX_TORQUE_H	Low byte of torque limit value
	AX-S1	P_ALARM_LED	Alarm LED function
		P_ALARM_SHUTDOWN	Alarm (Shut down) function
RAM Range	Common	P_EEPROM_IR_DETECT_COMPARE	IR sensor comparison value
		P_EEPROM_LIGHT_DETECT_COMPARE	Light sensor comparison value
		P_PRESENT_VOLTAGE	Present voltage
		P_PRESENT_TEMPERATURE	Present temperature
	AX-12	P_REGISTERED_INSTRUCTION	Question on instruction registration
		P_LOCK	EEPROM lock
		P_TORQUE_ENABLE	Torque On/Off
		P_LED	LED On/Off
		P_CW_COMPLIANCE_MARGIN	CW compliance margin
		P_CCW_COMPLIANCE_MARGIN	CCW compliance margin
	AX-12	P_CW_COMPLIANCE_SLOPE	CW compliance slope
		P_CCW_COMPLIANCE_SLOPE	CCW compliance slope
		P_GOAL_POSITION_L	Low byte of goal position value
		P_GOAL_POSITION_H	High byte of goal position value
		P_GOAL_SPEED_L	Low byte of goal speed value
		P_GOAL_SPEED_H	High byte of goal speed value
		P_TORQUE_LIMIT_L	Low byte of torque limit value
		P_TORQUE_LIMIT_H	High byte of torque limit value

		Value	Meaning
AX-12	RAM Range	P_PRESENT_POSITION_L	Low byte of present position value
		P_PRESENT_POSITION_H	High byte of present position value
		P_PRESENT_SPEED_L	Low byte of present speed value
		P_PRESENT_SPEED_H	High byte of present speed value
		P_PRESENT_LOAD_L	Low byte of present load value
		P_PRESENT_LOAD_H	High byte of present load value
		P_MOVING	Question on movement
		P_PUNCH_L	Low byte of punch value
		P_PUNCH_H	High byte of punch value
		P_IR_LEFT_FIRE_DATA	Left IR sensor data
AX-S1	RAM Range	P_IR_CENTER_FIRE_DATA	Center IR sensor data
		P_IR_RIGHT_FIRE_DATA	Right IR sensor data
		P_LIGHT_LEFT_DATA	Left light sensor data
		P_LIGHT_CENTER_DATA	Center light sensor data
		P_LIGHT_RIGHT_DATA	Right light sensor data
		P_IR_OBSTACLE_DETECTED	IR sensor detection
		P_LIGHT_DETECTED	Light sensor detection
		P_SOUND_DATA	Sound sensor data
		P_SOUND_DATA_MAX_HOLD	Maximum value of sound sensor data
		P_SOUND_DETECTED_COUNT	Number of sound detected
		P_SOUND_DETECTED_TIME_L	Low byte of sound detected time
		P_SOUND_DETECTED_TIME_H	High byte of sound detected time
		P_BUZZER_DATA0	Buzzer data 0
		P_BUZZER_DATA1	Buzzer data 1
		P_REMOCON_ARRIVED	IR remocon data arrival
		P_REMOCON_RXDATA0	Low byte of IR remocon received data
		P_REMOCON_RXDATA1	High byte of IR remocon received data
		P_REMOCON_TXDATA0	Low byte of IR remocon sent data
		P_REMOCON_TXDATA1	High byte of IR remocon sent data
		P_IR_DETECT_COMPARE	IR detection comparison value of RAM range
		P_LIGHT_DETECT_COMPARE	Light detection comparison value of RAM range

Construct the hardware environment as shown below.



Now, let's use the examples to learn how to use the functions of Dynamixel units.

[Example1] Let's create a program that writes 0 and 1023 alternately every two seconds to the goal positions from the Control Table for an AX-12 that has an ID of 1. This example can be found in the directory: “Bioloid SDK \CM-5 \app \example_control_AX12_write \example_control_AX12_write.c” in the expansion CD.

```
#include "../../include/libCM-5.h"

int main(void)
{
    PortInitialize();
    TimerInitialize(TIMER0, 128, 1); //Interrupt Enable
    SerialInitialize(SERIAL_PORT0,1,RX_INTERRUPT); //485
    SerialInitialize(SERIAL_PORT1,DEFAULT_BAUD_RATE,RX_INTERRUPT); //232
    sei();

    TxDString("\r\n CM-5 Example--- Control AX-12-Write\r\n");

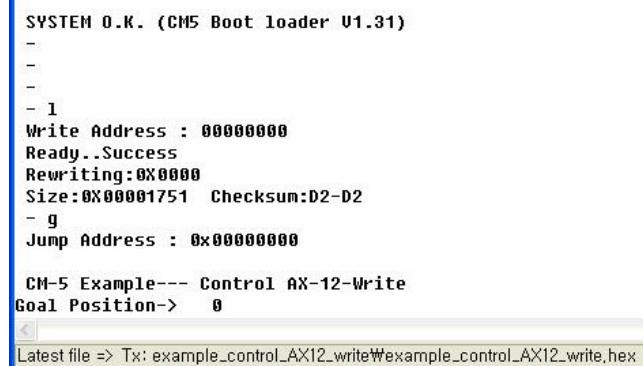
    byte bID;
    bID = 1;
    while(1) {
```

```
TxDString( "\rGoal Position-> 0");
WriteWord( bID, P_GOAL_POSITION_L, 0 );
MiliSec(2000);

TxDString( "\rGoal Position->1023");
WriteWord( bID, P_GOAL_POSITION_L, 1023 );
MiliSec(2000);
}

return 0;
}
```

Since the Goal Position uses 2 bytes in the control table, to write the goal position value, you must use a WriteWord function. When this program is executed, the motor moves back and forth between the 0 and 1023 positions (both extremes of the range of motion).



[Example 2] Let's create a program that writes 0 and 1023 back and forth to the goal positions in the Control Table on a scheduled interval, and then reads the present position and outputs the data on the screen for an AX-12 that has an ID 1. This example can be found in the directory: "Bioloid SDK \CM-5 \app \example_control_AX12_read \example_control_AX12_read.c" on the expansion CD.

```
#include "../../include/libCM-5.h"

int main(void)
{
    PortInitialize();
    TimerInitialize(TIMER0, 128, 1); //Interrupt Enable
    SerialInitialize(SERIAL_PORT0,1,RX_INTERRUPT); //485
```

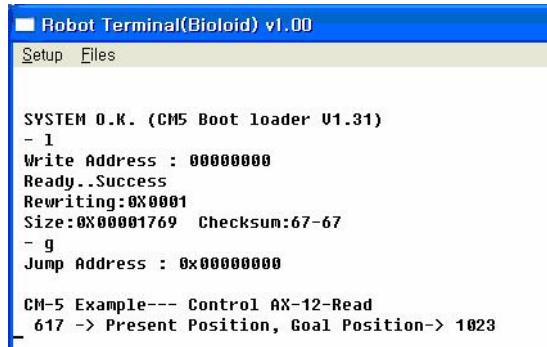
```
SerialInitialize(SERIAL_PORT1,DEFAULT_BAUD_RATE,RX_INTERRUPT); //232
sei();
TxDSString ("\r\n CM-5 Example--- Control AX-12-Read\r\n");

byte bID;
word wGoal, wCount;
bID = 1;
wGoal = 0x3ff;
WriteWord( bID, P_GOAL_SPEED_L, 0x100 );

while(1) {
    wGoal = 0x3ff-wGoal;
    WriteWord( bID, P_GOAL_POSITION_L, wGoal );
    TxDSString ("\r      -> Present Position, Goal Position->");
    TxDWord10( wGoal );

    for( wCount=0; wCount<2500; wCount++ ) {
        TxD1Byte ('\r');
        TxDWord10( ReadWord( bID, P_PRESENT_POSITION_L ) );
    }
}
return 0;
}
```

Since the goal position uses 2 bytes in the control table, to write the goal position, you have to use a WriteWord function. Similarly, since the present position uses 2 bytes, you must use a ReadWord function to read the present position. In the example, 0x100 was written for the goal speed, so that the user can check the goal position and present position at slow rate to verify the changes. When the program is executed, you can see that the present position steadily approaches the goal position. Moreover, through the control table concept, you can see firsthand that writing and reading parameters even when the motor is in motion is possible with the Dynamixel units.



The screenshot shows a terminal window titled "Robot Terminal(Bioloid) v1.00". The window has a blue header bar and a white main area. In the main area, there are two tabs: "Setup" and "Files", with "Setup" being the active tab. The terminal output is displayed in the following lines:

```
SYSTEM O.K. (CM5 Boot loader V1.31)
- 1
Write Address : 00000000
Ready..Success
Rewriting:0X0001
Size:0X00001769 Checksum:67-67
- g
Jump Address : 0x00000000

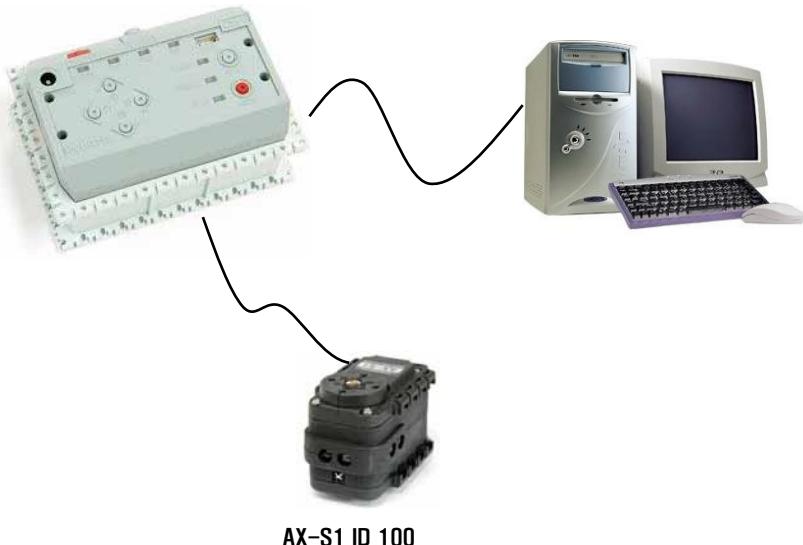
CM-5 Example--- Control AX-12-Read
- 617 -> Present Position, Goal Position-> 1023
```

◎ Using AX-S1

The AX-S1 is a dedicated sensor module for the Bioloid that has a infrared ray distance sensor, infrared data communication (communication between Bioloids), and sound detection capabilities, allowing Bioloid robots to respond to external stimuli through the behavior control program.

The AX-S1 generally has an ID of 100 but can be changed as needed so several AX-S1s can be used at the same time. Although the control method of the AX-S1 is the same as the AX-12 motor, its control table contents are different. Refer to the table and explanation of the control table in the previous chapter..

Construct the hardware environment as shown below.



Let's learn how to read specific values from the control table of the AX-S1 by looking at the next example.

[Example 1] Let's print the center infrared's sensor value of the AX-S1 on the PC screen through serial communication. This example can be found in "Bioloid SDK \CM-5 \app \example_control_AXS1_read \example_control_AXS1_read.c" on the expansion CD.

```
#include "../../include/libCM-5.h"

int main(void)
{
    PortInitialize();
    TimerInitialize(TIMER0, 128, 1); //Interrupt Enable
    SerialInitialize(SERIAL_PORT0,1,RX_INTERRUPT);//485
    SerialInitialize(SERIAL_PORT1,DEFAULT_BAUD_RATE,RX_INTERRUPT); //232
    sei();

    TxDString("\r\n CM-5 Example--- Control AX-S1-Read\r\n");

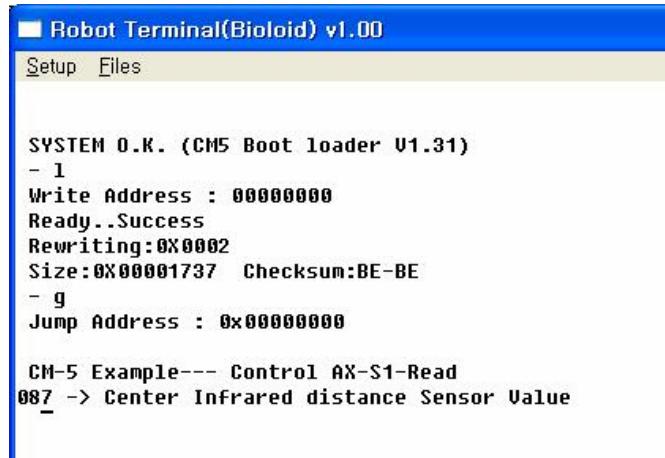
    byte bID;

    bID = 100;

    TxDString("\r      -> Center Infrared distance Sensor Value\r" );
    while(1)
    {
        TxDByte10( ReadByte( bID, P_IR_CENTER_FIRE_DATA ) );
        TxD1Byte( '\r' );
    }

    return 0;
}
```

As you can see, it is very simple. By using a ReadByte function and reading P_IR_CENTER_FIRE_DATA, the program can print the sensor's value on the screen. When the program is executed, it should display what is shown below. The sensor value continually changes depending on the distance of an object in front of the sensor.



[Example 2] Let's use AX-S1's buzzer to make musical notes. This example can be found in "Bioloid SDK \CM-5 \app \example_control_AXS1_write \example_control_AXS1_write.c" on the expansion CD.

```
#include "../../include/libCM-5.h"

int main(void)
{
    PortInitialize();
    TimerInitialize(TIMER0, 128, 1); //Interrupt Enable
    SerialInitialize(SERIAL_PORT0,1,RX_INTERRUPT);//485
    SerialInitialize(SERIAL_PORT1,DEFAULT_BAUD_RATE,RX_INTERRUPT); //232
    sei();

    TxDString("\r\n CM-5 Example--- Control AX-S1-Write\r\n");
    byte bID, bNote;
    bID = 100;
    bNote = 0;

    TxDString("\r      -> Buzzer Data 0 (musical note)\r" );
    while(1) {
        WriteByte( bID, P_BUZZER_DATA0, bNote );
        TxD1Byte( '\r' );
        TxDByte10( bNote );
        MiliSec(500);
        bNote = (bNote+1)%52;
    }
}
```

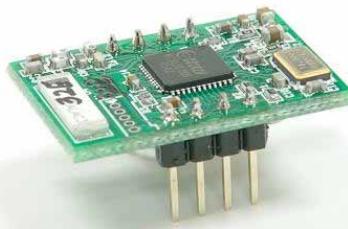
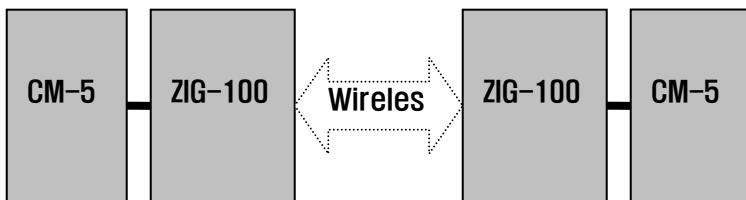
```
    }  
    return 0;  
}
```

To sound the buzzer, you have to write a desired musical note on P_BUZZER_DATA0 from the control table of the AX-S1. When this example is executed, for every half second, the buzzer sounds and goes up a half note. Refer to the AX-S1 manual for details on the musical note chart.

```
SYSTEM O.K. (CM5 Boot loader V1.31)  
- 1  
Write Address : 00000000  
Ready..Success  
Rewriting:0X0000  
Size:0X00001773 Checksum:67-67  
- g  
Jump Address : 0x00000000  
  
CM-5 Example--- Control AX-S1-Write  
010 -> Buzzer Data 0 (musical note)
```

2 – 1 – 4 . Wireless Data Communication

Zigbee, just like Bluetooth, is a frequently used PAN (Personal Area Network) communication technology. The Zigbee module, ZIG-100, is a communication module for the Bioloid and as such, is a communication technology that enables the Bioloid to transmit data and controls the robot in various ways.



○ Embedding a ZIG-100 unit onto the Bioloid

By default, the CM-5, the central control device of Bioloid, does not have a ZIG-100 unit. Thus, for the IR wireless communication between Bioloids, you need to have a ZIG-100 unit attached to the CM-5. In order to attach it, you need to disassemble the CM-5 and solder the ZIG-100 unit on the Zigbee circuit board. You need at least two sets of CM-5s and ZIG-100 modules to send and receive data.



◎ ZIGBEE ID

ZIG-100 modules each have their own unique IDs. Following that, in order to communicate between each others, they need to know the IDs of the respective devices. In general, with known IDs, devices can communicate from one to another and additionally, you can send broadcasting messages to all ZIG-100s. For further details, you can check out the ZIG-100 manual. You can change the communication mode setup through the behavior control program.

◎ Changing the setting

The CM-5 can determine which side the UART 1 will be connected to. Although it can be connected to either a PC or ZIG-100 unit, in order to communicate with the ZIG-100 unit, the UART must be connected to the ZIG-100 module. (In this case, communication with PC is not possible. Take note of this matter when programming).

Next is a macro that selects UART 1. Using this library, you can determine the direction of the UART.

```
#define RXD_ZIGBEE      cbi(PORTD,5), sbi(PORTD,6)
#define RXD_PC          cbi(PORTD,6), sbi(PORTD,5)
```

To change the setting of the ZIG-100 or to receive wireless data from the ZIG-100, you must execute RXD_ZIGBEE and connect UART 1 to the ZIG-100.

The followings are libraries related to the settings for ZIGBEE.

```
word GetHexWord(void); // Receives hexadecimal word type by text form and converts
it to data
byte ZigbeeSet(byte Function, word newDest, byte newWait); //Zigbee setting

word gwMyZigbeeID, gwYourZigbeeID; // Variable where both its own Zigbee ID and
opposing ID are saved.
```

ZigbeeSet, a function that reads or changes settings, can set the opposing ID and Wait Mode. For necessary movement, include the values as shown below to the first parameter function.

Function Value	Meaning
0	Reads the setting
1	Sets the opposing ID
2	Sets the WaitMode
3	Sets the opposing ID and the WaitMode

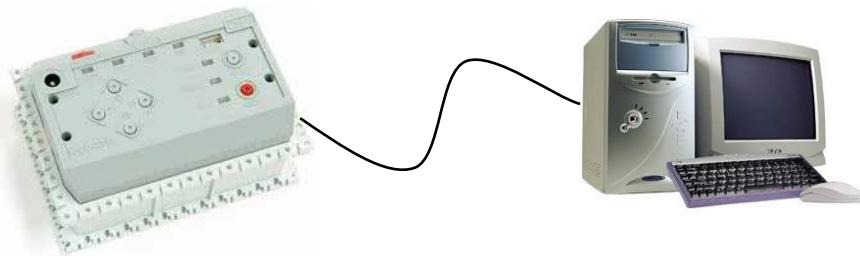
Since the PC and Zigbee will always use 57600bps, the baud rate must forced to 57600bps.

Broadcasting mode, unlike the one to one mode, sends data to all ZIG-100s that have been configured to receive broadcasting data. The opposing IDs are set to 65535 (0xFFFF). Additionally, with one remote control, more than one Bioloid can be controlled. For more details, refer to the ZIG-100 manual.

Let's look at a simple example that uses the ZigbeeSet function.

```
(ex1) When changing to Broadcasting mode -> ZigbeeSet (1, 0xFFFF, 0);
(ex2) When changing opposing ID to 560 -> ZigbeeSet (1, 560, 0);
(ex3) When changing opposing ID to 250 and Wait Mode to No-> ZigbeeSet (3, 560, 0);
(ex4) When changing Wait Mode to Yes -> ZigbeeSet (2, 0, 1);
```

Construct the hardware environment as shown below.



ZIG-100 embedded CM-5

[Example] Let's create a program that reads the settings of a ZIG-100 unit and then sets the opposing IDs as needed. This example can be found in "Bioloid SDK \CM-5 \app \example_zigbee_setting \example_zigbee_setting.c" in the expansion CD.

```
#include "../../include/libCM-5.h"
```

```
int main(void)
{
    PortInitialize();
    TimerInitialize(TIMER0, 128, 1); //Interrupt Enable
    SerialInitialize(SERIAL_PORT0,1,RX_INTERRUPT);//485
    SerialInitialize(SERIAL_PORT1,DEFAULT_BAUD_RATE,RX_INTERRUPT); //232
    sei();

    word wNewDestID;

    TxDString("\r\n CM-5 Example--- Zigbee TxD/RxD Test\r\n");

    while(1)
    {
        ZigbeeSet( 0,0,0 );
        TxDString("\r\n\r\n    Zigbee ID : 0x");
        TxDWord16( gwMyZigbeeID );
        TxD1Byte ('(');
        TxDWord10( gwMyZigbeeID );
        TxD1Byte (')');
        TxDString("\r\nDestination ID : 0x");
        TxDWord16( gwYourZigbeeID );
        TxD1Byte ('(');
        TxDWord10( gwYourZigbeeID );
        TxD1Byte (')');
        TxDString("\r\nChange Destination ID->Press 'D' Key");

        while( RxD8Interrupt()!='D' );

        TxDString("\r\nInput New Dest. ID(Hexadecimal)");
        wNewDestID = GetHexWord();
        ZigbeeSet(3, wNewDestID, 0 );
    }

    return 0;
}
```

When the program is executed, it reads the current settings of the ZIG-100 module and prints out the values on the screen. By pressing on the D button, the program sets the opposing IDs, and when a 4-digit (word type) value is inputted hexadecimally on a keyboard, its ID changes.

```
- 1
Write Address : 00000000
Ready..Success
Rewriting:0X0003
Size:0X000017F1 Checksum:A8-A8
- g
Jump Address : 0x00000000

CM-5 Example--- Zigbee TxD/RxD Test

Zigbee ID : 0x02BC( 700)
Destination ID : 0x02B2( 690)
Change Destination ID->Press 'D' Key
Input New Dest. ID(Hexadecimal)

Zigbee ID : 0x02BC( 700)
Destination ID : 0xFFFF(65535)
Change Destination ID->Press 'D' Key_
< Latest file => Tx: example_zigbee_setting\example_zigbee_setting.hex
```

◎ Communication send/receive

Next, by using the CM-5 that is connected to a ZIG-100 unit, it enables communication; to prevent the data loss or fault as a result of radio wave or frequency interferences, data is encapsulated into a specific format and becomes a packet before it is transmitted. With one packet, 2 bytes (word type) can be transmitted in a frame format that is shown below.

FF	55	Data_L	~Data_L	Data_H	~Data_H	*~ means inverse
----	----	--------	---------	--------	---------	------------------

A library function is provided so that data can be encapsulated in the above format and be ready to sending and receiving.

```
void ZigbeePacketCheck(void);           // Checks UART Buffer and receives wireless  
data  
void ZigbeePacketTransfer(word wData);  // creates packet and sends wireless data  
  
word gwZigbeeRxData;                  // Newly received wireless data  
byte gbNewPacket;                    // Checks whether there is new wireless data
```

The ZigbeePacketTransfer is a function that sends and receives 2 bytes of data after encapsulation.

ZigbeePacketCheck checks the UART buffer from the ZIG-100 unit, and when there is new data, it writes 1 to gbNewPacket and saves the new data to gwZigbeeRxData. Since this function can only analyze a packet when a Zigbee packet is completely received, you must include and execute loop routines.

To test the wireless data communication that uses ZIGBEE, it is necessary to have two CM-5 units that have a ZIG-100 unit. To make the communication possible, destination IDs must be set as shown in the previous example.



ZIG-100 embedded CM-5



ZIG-100 embedded CM-5

[Example] Let's create a program where when new data arrives, the AUX LED turns on and the new data is printed on a screen via the serial port and that sends specified data to the ZIGBEE when the CM-5 button is pressed. This example can be found in "Bioloid SDK \CM-5 \app \example_zigbee_txdrxd \example_zigbee_txdrxd.c" in the expansion CD.

```
#include "../../include/libCM-5.h"

int main(void)
{
    PortInitialize();
    TimerInitialize(TIMER0, 128, 1); //Interrupt Enable
    SerialInitialize(SERIAL_PORT0,1,RX_INTERRUPT); //485
    SerialInitialize(SERIAL_PORT1,DEFAULT_BAUD_RATE,RX_INTERRUPT); //232
    sei();

    TxDString("\r\n CM-5 Example--- Zigbee TxD/RxD Test\r\n");

    RXD_ZIGBEE;

    while(1)
    {
        ZigbeePacketCheck();
        if( gbNewPacket )
        {
            LEDOn( BIT_LED_AUX );
            TxDString(" New Zigbee Data Received: ");
            TxDWord10( gwZigbeeRxData );
            TxDString("\r\n");
            gbNewPacket = 0;
        }
    }
}
```

```
if( GetButtonState()==BIT_SW_UP )
{
    TxDString("\r Zigbee Data Transfer: ");
    TxDWord10( 12345 );
    TxDString("\r\n");
    ZigbeePacketTransfer( 12345 );
}
else
if( GetButtonState()==BIT_SW_DN )
{
    TxDString("\r Zigbee Data Transfer: ");
    TxDWord10( 321 );
    TxDString("\r\n");
    ZigbeePacketTransfer( 321 );
}
MiliSec(50);
LEDOff( BIT_LED_AUX );
}

return 0;
}
```

From looking at the program source code, you can check to see whether there is a new packet by running the ZigbeePacketCheck in the main loop. When there is a packet, the AUX LED is turned on and the data prints out via the serial communications. In addition, when the UP button is pressed, “12345” is transferred, whereas, when the DN button is pressed, “321” is transferred.

When this program is downloaded to two ZIG-100 embedded CM-5s that are set to communicate with one another and executed, the opposing CM-5’s AUX LED will turn on whenever the UP or DN button is pressed, and for the side that receives wireless data, it will prints out received data via the serial port.

```
- L  
Write Address : 00000000  
Ready..Success  
Rewriting:0X0000  
Size:0X000017B9 Checksum:2A-2A  
- G  
Jump Address : 0x00000000  
  
CM-5 Example--- Zigbee TxD/RxD Test  
New Zigbee Data Received: 12345  
New Zigbee Data Received: 12345  
New Zigbee Data Received: 12345  
New Zigbee Data Received: 321  
New Zigbee Data Received: 321
```

Latest file => Tx: example_zigbee_txdrxd\example_zigbee_txdrxd.hex

2 – 2 . PC BASED ROBOT PROGRAMMING

2 – 2 – 1 . Installing Development Resources

In this chapter, we introduce PC based robot programming using a Bioloid SDK.

○ Preparing the compile environment

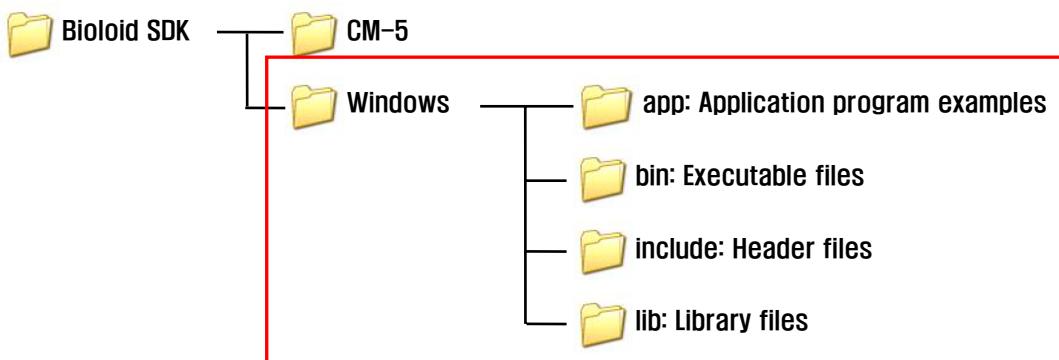
The development environment for using the Window related library of the Bioloid SDK is as follows:

–OS: Windows 2000/XP

– Compiler: Other compilers that support Visual C++ 6.0 and above, or Win32 programming.

A PC based robot program that uses the Bioloid SDK must develop via a Win32 programming method. Accordingly, developers must have sufficient background knowledge in Win32 programming. This manual, does not provide any lessons on Win32 programming. For users that need lessons on Win32, we recommend that they study Win32 related materials.

The contents in the Bioloid SDK/Windows folder are required for creating a PC based robot program. Copy the Bioloid SDK folder and save it to a folder of your choice.



◎ Preparing a device

Before starting PC based robot programming, you need to install devices provided in the expert educational kit to your PC.

< Installing the USB2Dynamixel>

Comm. Mode	Use as a USB2Serial
TTL	AX Series Dynamixel Control
232	Use as USB2Serial
485	DX.RX Series Dynamixel Control



Pin No.	Signal	Pin Figure
1	Data (TTL)	
2	RXD (RS-232)	
3	TXD (RS-232)	
4	D+ (RS-485)	
5	GND	
6	D- (RS-485)	
7	Short 8th pin	
8	Short 7th pin	
9	USB power (5V)	

< Step 1 > When the USBtoSerial is connected to the PC, the following “New Hardware” dialog window will appear. Here, select “Install from a list of specific location.”



< Step 2 > After selecting “Install from a list or specific location”, select Install/USB2Dynamixel folder from the expansion CD

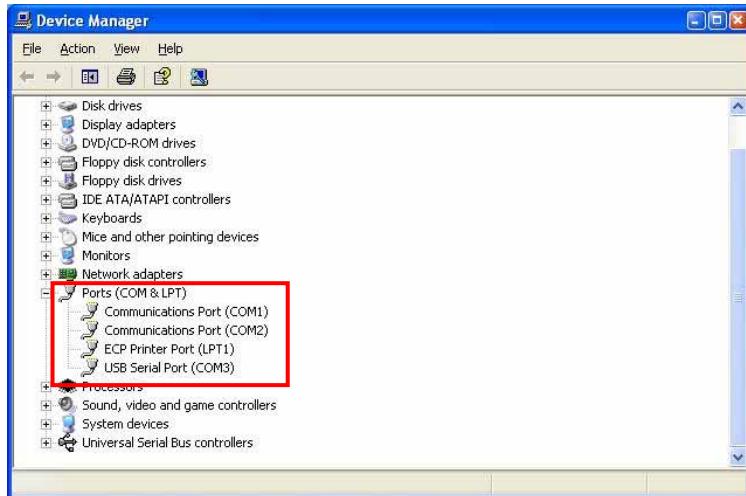
< Step 3 > Complete the installation.



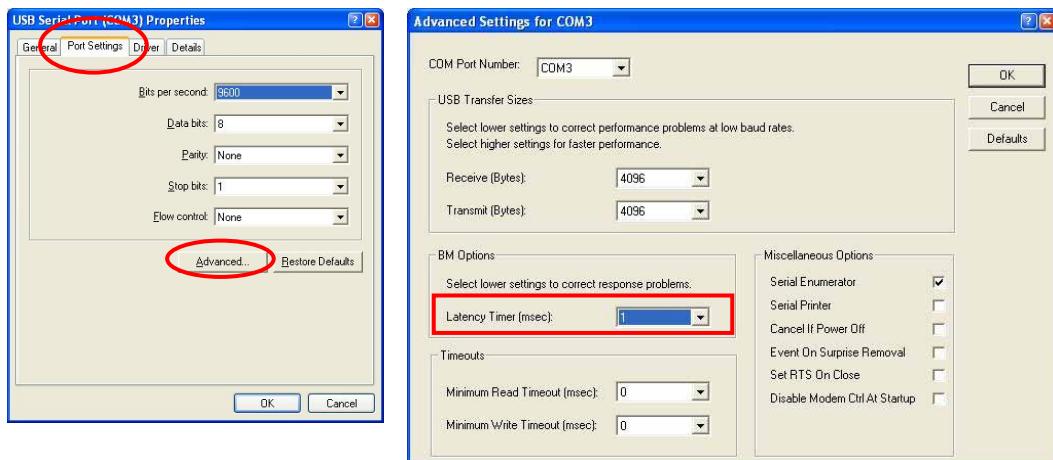
< Step 4 > Another “New Hardware” dialog window appears, repeat the Step 1~3 again.



< Step 5 > From the Device Manager, check to see where the USBtoSerial driver was installed. For a communication connection later on, remember the Port Number (COM x).



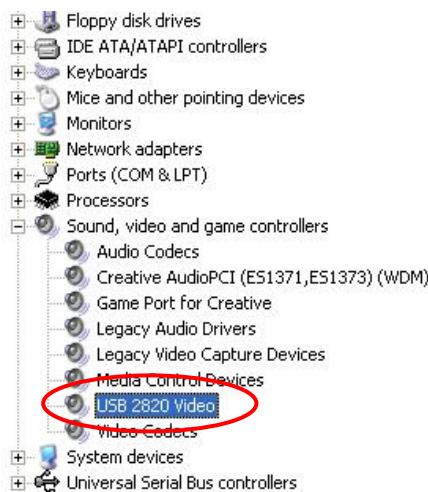
< Step 6 > After checking the driver information, change the “Latency Timer” to 1ms. This change allows for a fast response time.



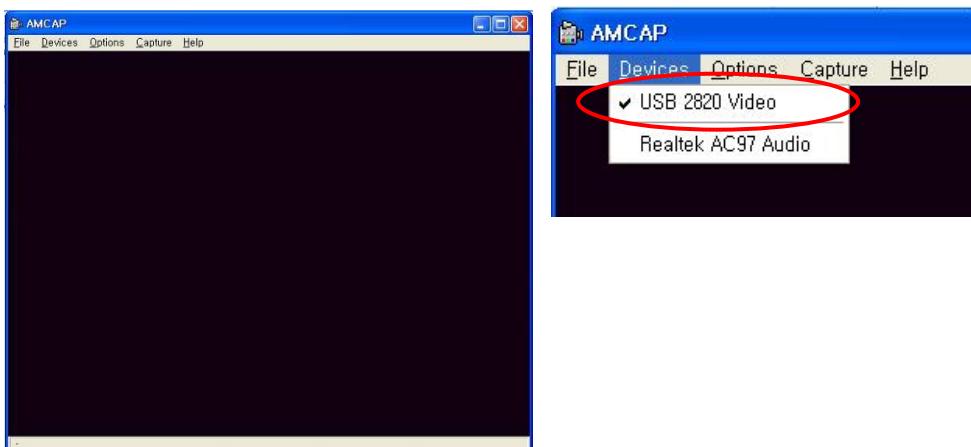
< Installing the wireless camera >

[Driver installation process]

- Turn on the wireless receiver's power switch
- Connect the wireless receiver to the PC via a USB port
- A "New Hardware" installation menu will appear
- Specify the driver's folder path (Select Install/Wireless Camera folder of expansion CD).
- Install the driver.
- Confirm the installation through PC's Hardware Manager



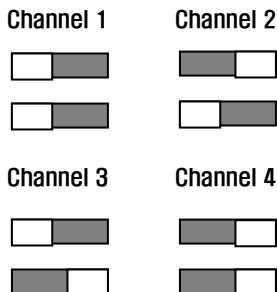
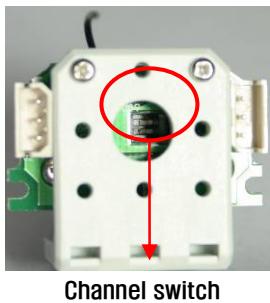
After the installation, run the Amcap.exe program in the Bioloid SDK/Windows/bin folder and check whether the image appears correctly.



[Channel select method]

- If using more than two wireless cameras, you cannot use the same channel for both cameras.
- In a case where wireless communication is carried out simultaneously using a Zig-100 unit, the similar frequency signals may create signal confusion. Thus, set in the channels as shown below.

Channel 3 >> Channel 2 >> Channel 4 >> Channel 1



Channel select button

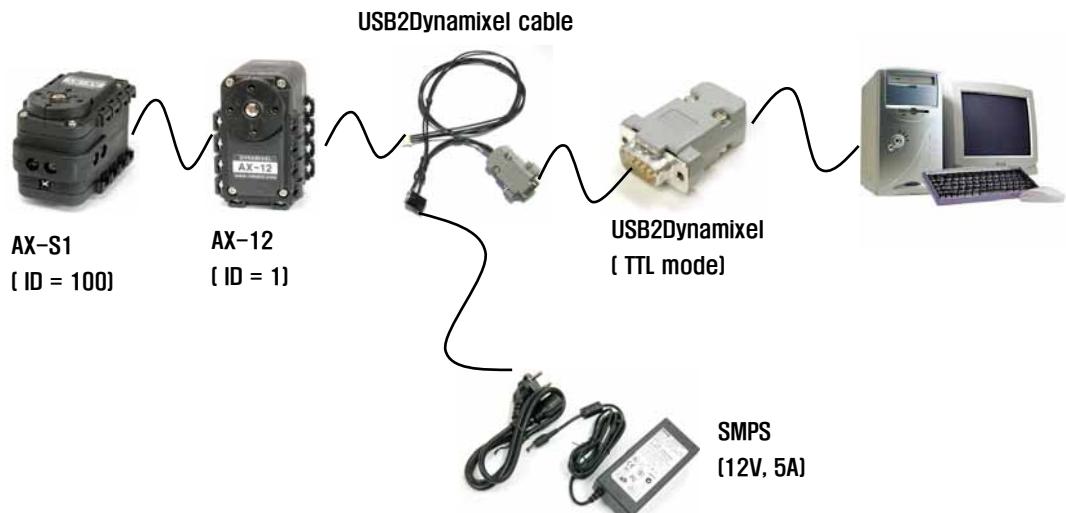


Fixed channel select dip switch

2 - 2 - 2 . Dynamixel Unit Control

- ◎ Reading a Dynamixel Unit's model number

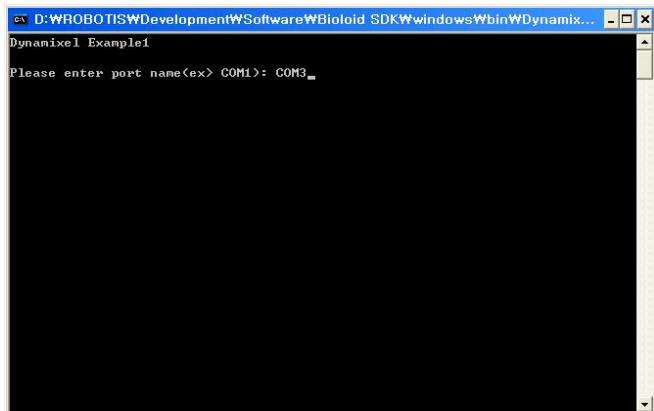
Construct the test environment as shown below.



*Caution

You must connect a Dynamixel unit and SMPS to the USB2Dynamixel before connecting to the USB2Dynamixel. If not, communication may not be possible. In this case, disconnect the USB2Dynamixel from the PC and connect it again.

Execute the Dynamixel Example1.exe from Bioloid SDK/windows/bin.



Input the port name connected to the USB2Dynamixel (e.g. COM1, COM2...), and press the “Enter” key.

```
Dynamixel Example1

Please enter port name<ex> COM1>: COM3
COM3, Baud number is 1<1000000bps>

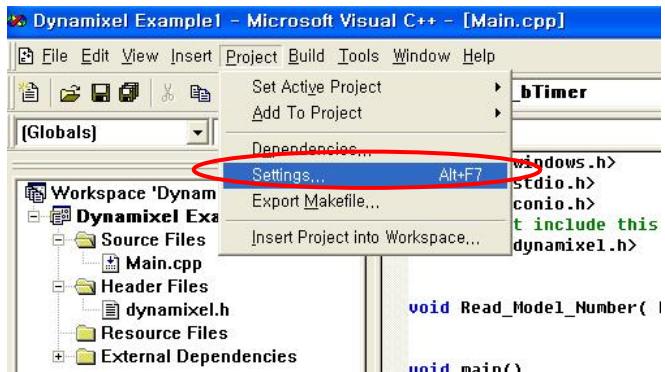
Successed communication!
Model Number of ID:1 is 12

Successed communication!
Model Number of ID:100 is 13
```

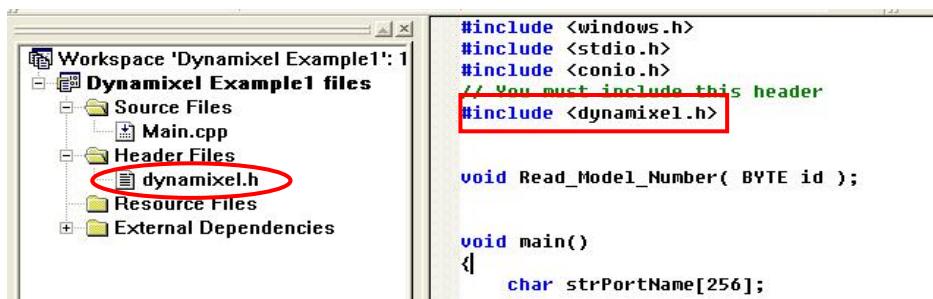
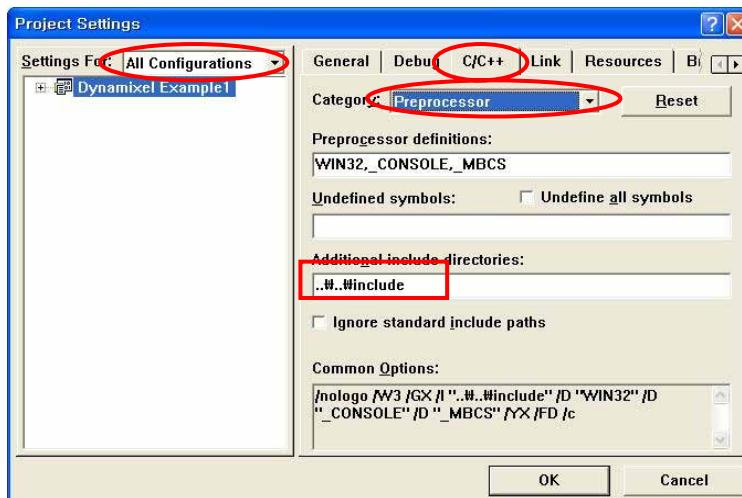
The model numbers of the AX-12 that has an ID 1 and the AX-S1 that has an ID 100 appear on the screen. Press any key and the program terminates.

First, use Visual C++ 6.0 and open the project file, "Bioloid SDK/windows/app/Dynamixel Example1/Dynamixel Example1.dsw." To use the Bioloid SDK, you must include applicable header files and libraries.

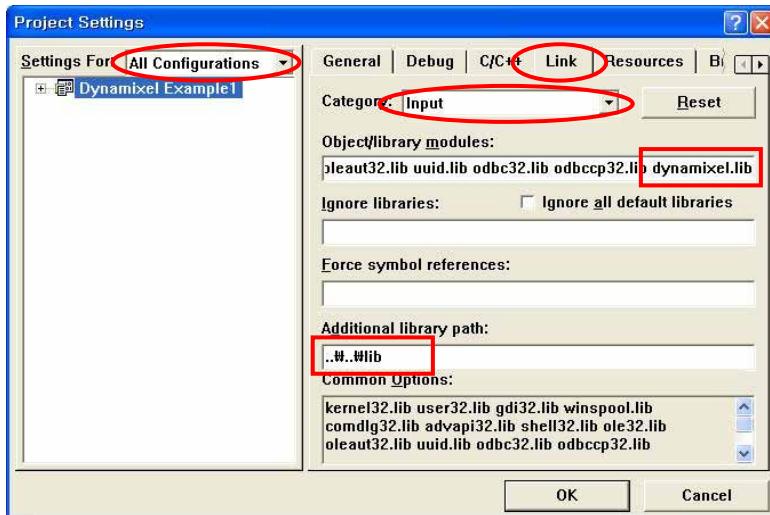
From the menu, select Project → Settings...



Include the header files from the Dynamixel's library and specify the path.



Include the library files from the Dynamixel's library and specify the path.



Let's analyze Main.cpp.

From the main function, you must first initialize the USB2Dynamixel using dxl_open.

```
printf( "Please enter port name(ex> COM1): " );
scanf( "%s", strPortName );
if( dxl_open( strPortName, 1 ) == false )
{
    printf( "Fail to open dynamixel libarary!\n" );
    goto END_PROGRAM;
}
printf( "%s, Baud number is 1(1000000bps)\n\n", strPortName );
```

The dxl_open function receives a port name and baud rate as a parameter. The port name is the same as the PC's communication port name, such as "COM1" or "COM2." Dynamixel Example1 receives the port name inputted strPortName by a keyboard. For the baud rate's number, refer to the AX-12 manual. In this example, the baud rate's number is 1 and the actual speed is 1000000bps. If the initialization of library is successful, it will return "true" and if not, "false."

This program reads the model number of the AX-12 and AX-S and outputs the results on a screen. In the actual main function, Read_Model_Number function is used to execute the program.

```
void Read_Model_Number( BYTE id );
```

Read_Model_Number receives the Dynamixel's ID as a parameter. The connected AX-12's ID is 1, and the AX-S1's ID is 100 and are used as follows.

```
Read_Model_Number( 1 );  
  
printf( "\n\n" );  
  
Read_Model_Number( 100 );
```

The Read_Model_Number function communicates with Dynamixel units and finds out the model number. To control the Dynamixel units, you must know the basics of instruction packets and status packets. For more details on status packets, refer to the AX-12/S1 manual.

```
dxl_instruction InstPacket;  
  
InstPacket.id          = id;  
InstPacket.instruction = INST_READ;  
InstPacket.address     = P_MODEL_NUMBER_L;  
InstPacket.parameter[0] = 2;  
InstPacket.length      = 4;
```

The instruction packet is declared as a dxl_instruction structure and among the member variable, ID is a variable received as a parameter from the Read_Model_Number function. The instruction inputs INST_READ, a read command. The address assigns P_MODEL_NUMBER_L to a model number from the control table of Dynamixel. To find out more about the control table, refer to the AX-12/S1 manual. For parameter, when INST_READ is read, the number of bytes becomes the input and in the case of model number, as it is 2 bytes, the input is "2." The length is the length of instruction packet and the calculation method is as follows.

Length = the number of parameters + 3 ("3" is a value where 1 byte of ID, instruction and address have been added)

When the instruction packet is ready, by using a dxl_control function, it can communicate with the Dynamixel. The dxl_control function receives the instruction packet as a parameter, sends the status packet to a output and informs the user of the error code as a return value.

```
dxl_instruction InstPacket;  
dxl_status StatusPacket;  
int ErrorCode;  
  
ErrorCode = dxl_control( &InstPacket, &StatusPacket, true );  
if( ErrorCode == ERR_CODE_SUCCESS )
```

```

{
    printf( "Successed communication!\n" );
}
else if( ErrorCode == ERR_CODE_SEND_FAIL )
{
    printf( "Fail to send Instruction Packet!\n" );
}
else if( ErrorCode == ERR_CODE_PACKET_LOST )
{
    printf( "Lost Status Packet!\n" );
}
else if( ErrorCode == ERR_CODE_TIMEOUT )
{
    printf( "Timeout occured\n" );
}

```

The ErrorCode, a return value of dxl_control function, must be checked before using the status packet.

Value	Meaning
ERR_CODE_SUCCESS	Communication success.
ERR_CODE_SEND_FAIL	Not able to send instruction packet from PC.
ERR_CODE_PACKET_LOST	Parts of status packet is lost.
ERR_CODE_TIMEOUT	Status packet not delivered

Only when it is ERR_CODE_SUCCESS can the contents of the status packet be used.

The dxl_control function can decide whether or not to wait for the status packet as a parameter. Generally, by making a parameter “true,” it waits for the status packet, but in a special case, by making it “false” it does not have to wait for the status packet. By using this parameter, when the status packet has no significance, it can minimize the wait time, increasing the efficiency of the program.

```

dxl_instruction InstPacket;
dxl_status StatusPacket;
WORD Model_Number;

if( StatusPacket.id != InstPacket.id )
    printf( "Incorrect ID of Status Packet!\n" );
else if( StatusPacket.error == ERROR_NONE )
{
    Model_Number = MAKE_WORD(StatusPacket.parameter[0], StatusPacket.parameter[1]);
    printf( "Model Number of ID:%d is %d\n", InstPacket.id, Model_Number );
}

```

The status packet is declared as a dxl_status structure. If communication is successful, you have to check whether the ID of status packet and the ID of instruction packet match in order to check to see if the received status packet is a response to the sent instruction. Moreover, the status packet has an error member variable, allowing you to check for errors in Dynamixel units.

Value	Meaning
ERROR_NONE	There is no error.
ERROR_VOLTAGE	Error in input voltage.
ERROR_ANGLE	Angle value out of range.
ERROR_OVERHEAT	Temperature value out of range.
ERROR_RANGE	Parameter value out of valid range.
ERROR_CHECKSUM	Error checksum in instruction packet.
ERROR_OVERLOAD	Not able to control current load with the maximum torque.
ERROR_INSTRUCTION	Error in instruction code.

To check the return value of error code of dxl_control function from the actual source code, use the following method.

```
dxl_status StatusPacket;

if( StatusPacket.error != ERROR_NONE )
{
    if( StatusPacket.error & ERROR_VOLTAGE )
        printf( "Voltage error!" );
    else if( StatusPacket.error & ERROR_ANGLE )
        printf( "Angle error!" );
    else if( StatusPacket.error & ERROR_OVERHEAT )
        printf( "Overheat error!" );
    else if( StatusPacket.error & ERROR_RANGE )
        printf( "Range error!" );
    else if( StatusPacket.error & ERROR_CHECKSUM )
        printf( "Checksum error!" );
    else if( StatusPacket.error & ERROR_OVERLOAD )
        printf( "Overload error!" );
    else if( StatusPacket.error & ERROR_INSTRUCTION )
        printf( "Instruction error!" );
}
```

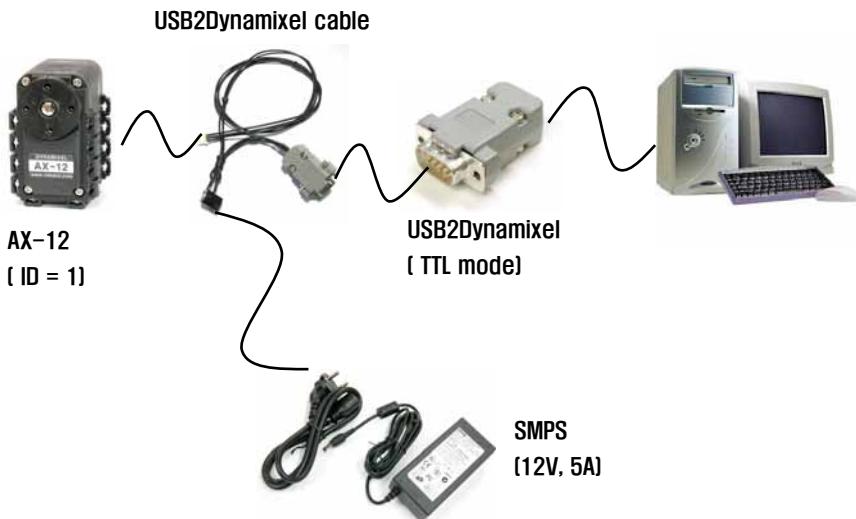
Data can be checked only after the program has been confirmed that there is no error. In a status packet, there will be 2 bytes in the model number. Among the member variables of dxl_status structure, there will be 2 bytes for the model number in the same order as the parameters. MAKE_WORD is a macro that combines high and low bytes into a 2 byte “word type” variable. Through this macro, you can easily check the model number.

Before closing the application program, always terminate USB2Dynamixel by using dxl_close.

```
dxl_close();
```

◎ Controlling the AX-12's position and speed

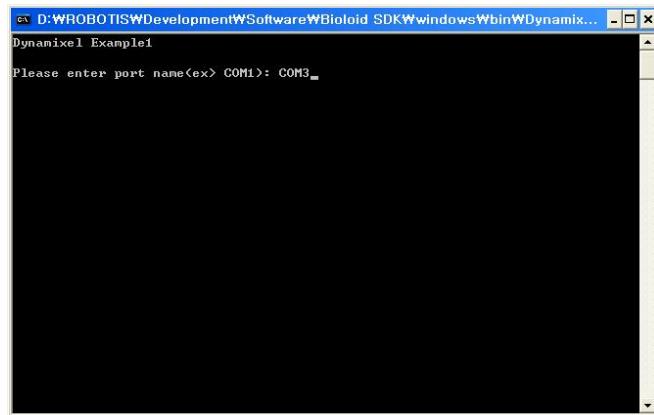
Construct the test environment as shown below.



※Caution

You must connect Dynamixel units and SMPS to the USB2Dynamixel cable before connecting to the USB2Dynamixel. If not, communication may not be possible. In this case, disconnect the USB2Dynamixel from the PC and connect it again.

Execute the Dynamixel Example2.exe from Bioloid SDK/windows/bin.



Input the port name connected to the USB2Dynamixel (e.g. COM1, COM2...), and press the "Enter" key.

```
Dynamixel Example2
Please enter port name(ex> COM1): COM3
COM3, Baud number is 1(1000000bps)

<1/5>
Successed communication!
Goal Position: 100, Goal Speed: 600
Successed communication!
Goal Position: 1000, Goal Speed: 600
<2/5>
Successed communication!
Goal Position: 100, Goal Speed: 600
Successed communication!
Goal Position: 1000, Goal Speed: 600
<3/5>
Successed communication!
Goal Position: 100, Goal Speed: 600
Successed communication!
Goal Position: 1000, Goal Speed: 600
<4/5>
Successed communication!
Goal Position: 100, Goal Speed: 600
Successed communication!
Goal Position: 1000, Goal Speed: 600
<5/5>
Successed communication!
Goal Position: 100, Goal Speed: 600
Successed communication!
Goal Position: 1000, Goal Speed: 600
Successed communication!
Goal Position: 512, Goal Speed: 300
End
```

AX-12 will move from 100 to 1000, repeating five times. The final position will be 512. Press any key and the program terminates.

First, use Visual C++ 6.0 and open the project file, “Bioloid SDK/windows/app/Dynamixel Example2/Dynamixel Example2.dsw.” For basic conditions in making an application program and using the Dynamixel libraries, refer to the Dynamixel Example1 explanation.

Let's analyze the source of Main.cpp. This example shows how to control the Dynamixel units by giving the location and speed for the AX-12. In this example, a new function Motor_Move is introduced.

```
void Motor_Move( BYTE id, WORD pos, WORD speed );
```

This function moves the appropriate AX-12 unit by giving the ID of the AX-12, the position value that it will move to, and speed as a parameter. The main function will program the AX-12 with the ID 1 will move from the 100 to the 1000 position at the speed of 600, repeating five times.

```
for( i=0; i<5; i++ )
{
    printf( "<%d/5>\n", i+1 );
    Motor_Move( 1, 100, 600 );
    Sleep( 1500 );

    Motor_Move( 1, 1000, 600 );
    Sleep( 1500 );
}
Motor_Move( 1, 512, 300 );
```

The reason why the Sleep function was used to delay the program was so that it waits for the AX-12 to arrive at the goal position. The number “1500” indicates 1500msec and it denotes a delay time. Lastly, the program makes the Dynamixel unit move to the 512 position at the speed of 300.

Here, let's analyze the Motor_Move function and see how the instruction packet is created.

```
dxl_instruction InstPacket;

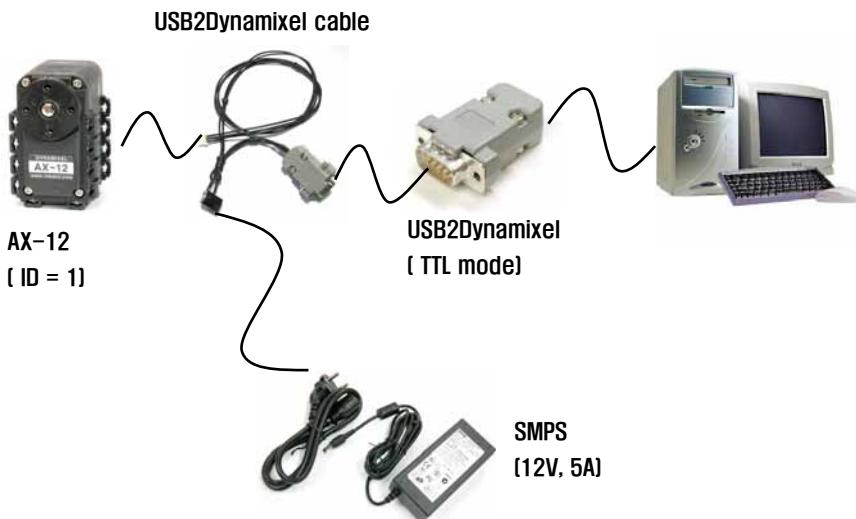
InstPacket.id          = id;
InstPacket.instruction = INST_WRITE;
InstPacket.address     = P_GOAL_POSITION_L;
InstPacket.parameter[0] = LOW_BYT(pos);
InstPacket.parameter[1] = HIGH_BYT(pos);
InstPacket.parameter[2] = LOW_BYT(speed);
InstPacket.parameter[3] = HIGH_BYT(speed);
InstPacket.length       = 7;
```

Among the dxl_instruction structure value, ID uses the value that is transferred as a parameter. The instruction will use INST_WRITE so that it can write to the control table. The address P_GOAL_POSITION, is the same as the goal position of the control table. The program inserts data that will be used in the correct order for the parameter. The goal position and the goal speed are listed in the control table successively in the order of: low byte of goal position, high byte of goal position, low byte of goal speed, and high byte of goal speed. The low byte and high byte are macros that divide bytes into either a high or low byte. The calculation of the length is the number of parameters + 3, so it becomes $4 + 3 = 7$.

Therefore, if you create instruction packets and use the dxl_control function to send packets to Dynamixel units, the Dynamixel unit with the appropriate ID will move to the position at the specified speed.

◎ Reading the Ax-12's current position value

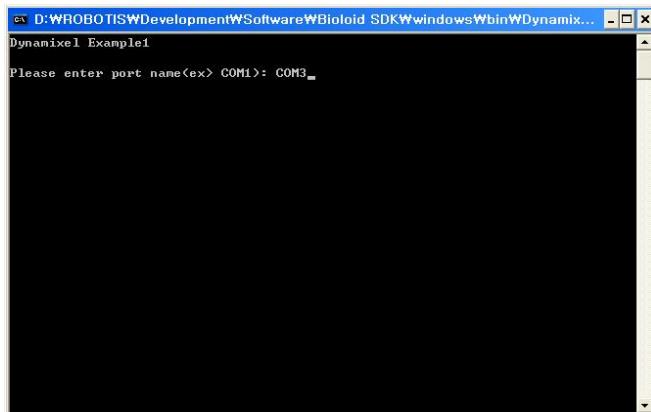
Construct the test environment as shown below.



*Caution

You must connect Dynamixel units and the SMPS to the USB2Dynamixel cable before connecting to the USB2Dynamixel. If not, communication may not be possible. In this case, disconnect the USB2Dynamixel from the PC and connect it again.

Execute the Dynamixel Example3.exe from Bioloid SDK/windows/bin.



Input the port name connected to the USB2Dynamixel (e.g. COM1, COM2...), and press the "Enter" key.

```
Dynamixel Example3

Please enter port name(ex> COM1): COM3
COM3, Baud number is 1(1000000bps)

<1/5>
Successed communication!
Goal Position: 100, Goal Speed: 600
Present Position: 100
Successed communication!
Goal Position: 1000, Goal Speed: 600
Present Position: 1000
<2/5>
Successed communication!
Goal Position: 100, Goal Speed: 600
Present Position: 100
Successed communication!
Goal Position: 1000, Goal Speed: 600
Present Position: 1000
<3/5>
Successed communication!
Goal Position: 100, Goal Speed: 600
Present Position: 100
Successed communication!
Goal Position: 1000, Goal Speed: 600
Present Position: 1000
<4/5>
Successed communication!
Goal Position: 100, Goal Speed: 600
Present Position: 100
Successed communication!
Goal Position: 1000, Goal Speed: 600
Present Position: 1000
```

The program has the same basic principle as the Dynamixel Example2, but when the AX-12 moves, you can see the position value update itself in real time.
Press any key and the program terminates.

First, use Visual C++ 6.0 and open the project file, “Bioloid SDK/windows/app/Dynamixel Example3/Dynamixel Example3.dsw.” For basic conditions in making an application program and using the Dynamixel libraries, refer to the “◎ Reading a Dynamixel Unit”’s model number.”

Let’s analyze the source of Main.cpp. This example shows how to control the AX-12 through Motor_Move and simultaneously lets you know the current position value. For this example, a new function Motor_ReadPos is introduced.

```
void Motor_ReadPos( BYTE id );
```

After the Dynamixel unit’ s ID is assigned, this function reads the current position value and outputs it on a screen. The main function is the same as the Dynamixel Example2, but after the Dynamixel unit is moved by Motor_Move, you can see the entire process by using the Motor_ReadPos function. In the Dynamixel Example2, the Sleep function was used to delay the process while the AX-12 was moving, but in this example, the program repeats 1500 times reading and outputting the current position value.

```
for( i=0; i<5; i++ )
{
    printf( "<%d/5>\n", i+1 );
    Motor_Move( 1, 100, 600 );
    for( j=0; j<1500; j++ )
    {
        Motor_ReadPos( 1 );
    }
    printf( "\n" );

    Motor_Move( 1, 1000, 600 );
    for( j=0; j<1500; j++ )
    {
        Motor_ReadPos( 1 );
    }
    printf( "\n" );
}
Motor_Move( 1, 512, 300 );
```

Here, let’s analyze Motor_ReadPos function and see how the instruction packet is created.

```
dxl_instruction InstPacket;

InstPacket.id      = id;
InstPacket.instruction = INST_READ;
```

```
InstPacket.address      = P_PRESENT_POSITION_L;
InstPacket.parameter[0] = 2;
InstPacket.length       = 4;
```

Among the dxl_instruction structure value, ID uses the value that is transferred as a parameter. The instruction will use INST_READ so that it can read from the control table. The address is the P_PRESENT_POSITION_L, which is the present position of the control table. Since the current position value is 2 bytes, the parameter for the byte read is “2”. For calculating the length; since the number of parameters is 3, the length is $1 + 3 = 4$. By using the dxl_control function, it transfers the instruction packet and checks to see if the status packet was received without an error. For more details on the communication problems, refer to the “[○ Reading a Dynamixel Unit’s model number.](#)”

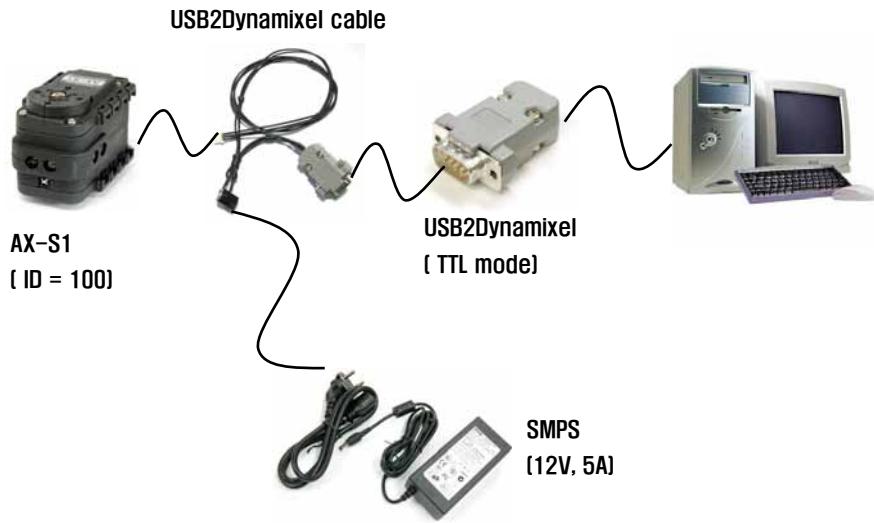
```
dxl_instruction InstPacket;
dxl_status StatusPacket;
int ErrorCode;
WORD position;

ErrorCode = dxl_control( &InstPacket, &StatusPacket, true );
if( ErrorCode == ERR_CODE_SUCCESS )
{
    if( StatusPacket.error == ERROR_NONE )
    {
        position = MAKE_WORD(StatusPacket.parameter[0], StatusPacket.parameter[1]);
        printf( "Present Position: %d\r", position );
    }
}
```

When there is no communication problem, the current position value is included in the member variable parameter of the dxl_status structure in the order of low byte and high byte. You can change the low and high byte into word type data by using the MAKE_WORD macro.

◎ Playing musical notes from the AX-S1's buzzer

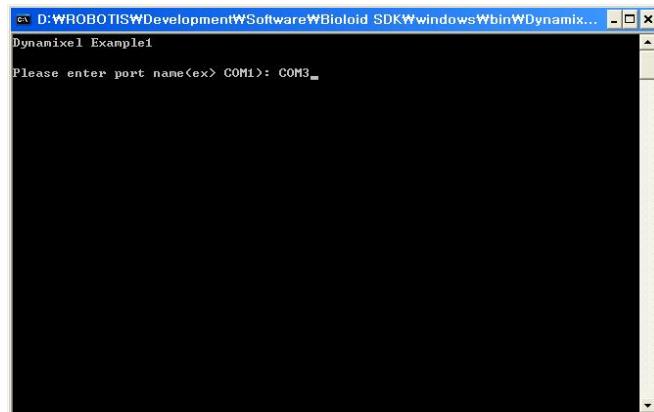
Construct the test environment as shown below.



※Caution

You must connect Dynamixel units and the SMPS to the USB2Dynamixel cable before connecting to the USB2Dynamixel. If not, communication may not be possible. In this case, disconnect the USB2Dynamixel from the PC and connect it again.

Execute the Dynamixel Example4.exe from Bioloid SDK/windows/bin.



Input the port name connected to the USB2Dynamixel (e.g. COM1, COM2...), and press the "Enter" key.

```
Dynamixel Example4

Please enter port name(ex> COM1): COM3
COM3, Baud number is 1(1000000bps)

Successed communication!
Sound index: 0, Sound length: 0
Successed communication!
Sound index: 1, Sound length: 0
Successed communication!
Sound index: 2, Sound length: 0
Successed communication!
Sound index: 3, Sound length: 0
Successed communication!
Sound index: 4, Sound length: 0
Successed communication!
Sound index: 5, Sound length: 0
Successed communication!
Sound index: 6, Sound length: 0
Successed communication!
Sound index: 7, Sound length: 0
Successed communication!
Sound index: 8, Sound length: 0
Successed communication!
Sound index: 9, Sound length: 0
Successed communication!
Sound index: 10, Sound length: 0
Successed communication!
Sound index: 11, Sound length: 0
Successed communication!
Sound index: 12, Sound length: 0
Successed communication!
Sound index: 13, Sound length: 0
```

As the message appears on a screen, the AX-S1 will make a sound.

The sound index indicates the musical scale of the AX-31. The program will make a sound from 0 to 51 before it stops.

Press any key and the program terminates.

First, use Visual C++ 6.0 and open the project file, Bioloid SDK/windows/app/Dynamixel Example4/Dynamixel Example4.dsw. For basic conditions in making an application program and using the Dynamixel libraries, refer to the “◎ Reading a Dynamixel Unit’s model number.”

Let’s analyze the source of Main.cpp. This example shows how to play a musical note by using the buzzer of the AX-S1. For this example, a new function Buzzer_sound is introduced.

```
void Buzzer_sound( BYTE id, BYTE index, BYTE length );
```

This function transforms the ID of the AX-S1, the musical note, and the playing time as parameters and uses the buzzer to make a sound. For more details on playing musical notes on the buzzer, refer to the AX-S1 manual. There are total of 52 available musical notes, with playing times in increments of 10msec with a minimum play time of 300ms. If the length is “10,” the playing time is $100 \times 10 = 1000$ msec. If the length is 0~2, then the playing time is 300msec, not 0~200msec. If you look at the main function, an AX-S1 that has an ID of 100 plays each musical note from 0 to 51 for 300msec.

```
for( i=0; i<52; i++ )
{
    Buzzer_sound( 100, i, 0 );
    Sleep(500);
}
```

Here, by using the Sleep function, it delays the program until the musical notes finishes sounding. Since the playing time is 300msec, the Sleep delay was purposely set to 500msec to leave a margin for error.

Let’s see how instruction packet is created by analyzing the Buzzer_sound function..

```
dxl_instruction InstPacket;

InstPacket.id          = id;
InstPacket.instruction = INST_WRITE;
InstPacket.address     = P_BUZZER_DATA0;
InstPacket.parameter[0] = index;
InstPacket.parameter[1] = length;
InstPacket.length       = 5;
```

Among the dxl_instruction structure values, ID uses the value that is handed over as a parameter. The instruction will use INST_WRITE so that it can write to the control table. The address P_BUZZER_DATA0 is buzzer sound (Buzzer data0) in the control table. The program inserts data that will be used in the proper order for the parameters. The buzzer uses 2 bytes: melody type and playing time. Therefore, for parameter[0], insert the musical note type, and in parameter[1], insert the playing time. For the calculating of length, since the number of

parameters is +3, the length is $2 + 3 = 5$. By using the dxl_control function, it transfers the instruction packet and checks to see if the status packet was received without an error. For more details on the communication errors, refer to the “[○ Reading a Dynamixel Unit’s model number.](#)”

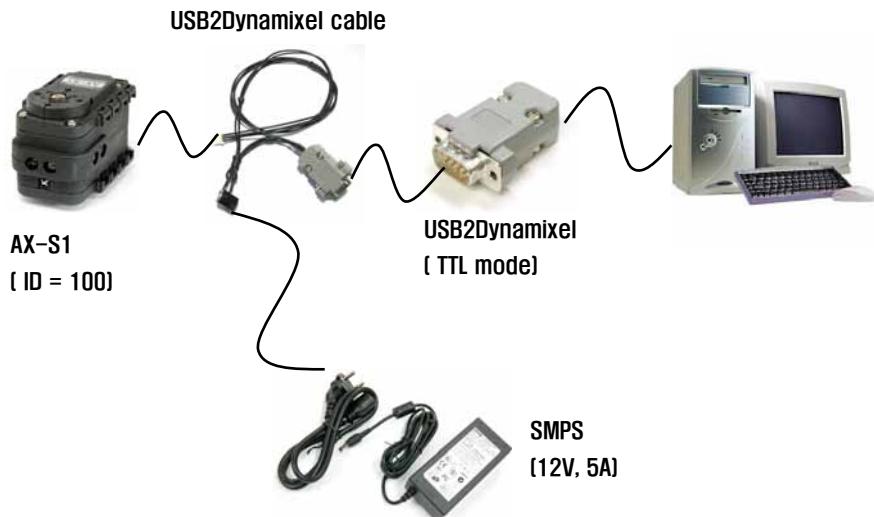
```
dxl_instruction InstPacket;
dxl_status StatusPacket;
int ErrorCode;

ErrorCode = dxl_control( &InstPacket, &StatusPacket, true );
if( ErrorCode == ERR_CODE_SUCCESS )
{
    printf( "Successed communication!\n" );

    if( StatusPacket.id != InstPacket.id )
        printf( "Incorrect ID of Status Packet!\n" );
    else if( StatusPacket.error == ERROR_NONE )
    {
        printf( "Sound index: %d, Sound length: %d\n", index, length );
    }
}
```

◎ Reading the AX-S1's IR sensor value

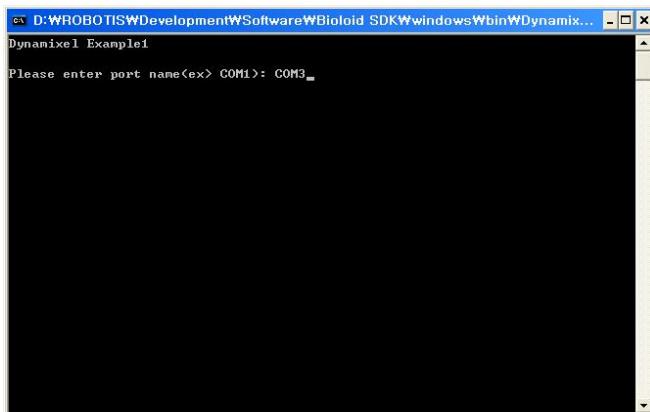
Construct the test environment as shown below.



※Caution

You must connect the Dynamixel and SMPS to the USB2Dynamixel cable before connecting to the USB2Dynamixel. If not, communication may not be possible. In this case, disconnect the USB2Dynamixel from the PC and connect it again.

Execute the Dynamixel Example5.exe from Bioloid SDK/windows/bin.



Input the port name connected to the USB2Dynamixel (e.g. COM1, COM2...), and press the “Enter” key.

Dynamixel Example5
Please enter port name(ex> COM1): COM3
COM3, Baud number is 1<1000000bps>
If you want to quit, press 'q' key!
Left IR: 15, Center IR: 255, Right IR: 0

When you cover the IR sensor of the AX-S1 with your hands, you will see the values changing.

When “q” is pressed on the keyboard, the IR sensor detection closes.

Press any key and the program

First, open the “Bioloid SDK/windows/app/Dynamixel Example5/Dynamixel Example5.dsw” file with Visual C++ 6.0. For basic conditions in making an application program and using the Dynamixel libraries, refer to the “◎ Reading a Dynamixel Unit’s model number.”

Let’s analyze the source of Main.cpp. This example shows how to read data that detected by the IR (infrared ray) sensor. For this example, a new function IR_read is introduced.

```
void IR_read( BYTE id );
```

Once the Dynamixel unit has an assigned ID, this function reads the left, center, and right IR sensor values and outputs them on a screen. By using this function in the main function, it continually reads the IR sensor value of the AX-S1. When “q” on the keyboard is pressed, the program terminates.

```
bool run = true;

while( run )
{
    IR_read( 100 );
    if( _kbhit() != 0 )
    {
        if( getch() == 'q' )
            run = false;
    }
}
```

Here, let’s analyze the IR_read function and see how the instruction packet is created.

```
dxl_instruction InstPacket;

InstPacket.id      = id;
InstPacket.instruction = INST_READ;
InstPacket.address   = P_IR_LEFT_FIRE_DATA;
InstPacket.parameter[0] = 3;
InstPacket.length     = 4;
```

Among the dxl_instruction structure value, ID uses the value that is handed over as a parameter. The instruction will use INST_READ so that it can write to the control table. The address P_IR_LEFT_FIRE_DATA the left IR sensor data of the control table. The function inserts the number of bytes that it will read into a parameter and since it has left, center and right IR sensor data, it requires 3 bytes. For calculating length, since the number of parameters is +3, the length is 1 + 3 = 4. By using the dxl_control function, it transfers the instruction packet and checks to see if the

status packet was received without error. For more details on the communication problems, refer to the “[○ Reading a Dynamixel Unit’s model number](#)”.

```
dxl_instruction InstPacket;
dxl_status StatusPacket;
int ErrorCode;
BYTE left, center, right;

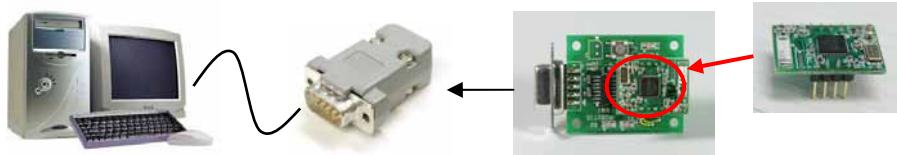
ErrorCode = dxl_control( &InstPacket, &StatusPacket, true );
if( ErrorCode == ERR_CODE_SUCCESS ) // Communication success
{
    if( StatusPacket.error == ERROR_NONE )
    {
        left = StatusPacket.parameter[0];
        center = StatusPacket.parameter[1];
        right = StatusPacket.parameter[2];
        printf( "Left IR: %d, Center IR: %d, Right IR: %d      \r"
                ,left, center, right );
    }
}
```

When there is no error in the status packet, the left, center, and right IR sensor data are included in order in the parameter array, a member variable of `dxl_status` structure. The `IR_read` function outputs this data on the screen.

2-2-3. Wireless Data Communication

◎ Communicating with the behavior control program

Construct the test environment as shown below.



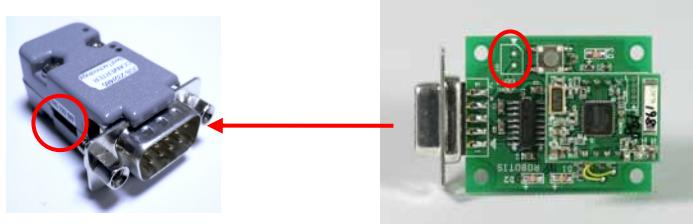
USB2Dynamixel
[RS-232 mode]



Embed the Zig-100 and download the behavior control program for communication



The Zig2Serial must provide 5V power supply separately. If it is used with the USB2Dynamixel, it can be used without requiring power supply. For this case, the USB2Dynamixel's communication mode must be configured to RS-232.



[Zig-100 1:1 communication set using Zig2Serial]

Among the Bioloid software installed on the PC, you can set the Zig-100 communication mode that utilizes Zig2Serial by using the robot terminal. The following commands are used to set the Zig-100 communication mode.

Command	Action
i	Checks current Zig-100's communication set value
d	Sets ID of targeted Zig-100
b	Set current Zig-100's Baud Rate
w	Set current wait mode of Zig-100
e	Terminate communication set mode

When you press the reset button of the Zig2Serial while "!"(Shift+1) key is pressed, the Zig-100 will go into communication set mode.

```
[Robotis Zigbee Monitor]
Baudrate :0F -> 57600
My ID. :02E6
Dest.Addr:02BC
Wait Mode:No
Command- (I)nfo, (D)est. Addr/(B)aud, (W)ait mode, (E)xit
!?
Command- (I)nfo, (D)est. Addr/(B)aud, (W)ait mode, (E)xit
?!
```

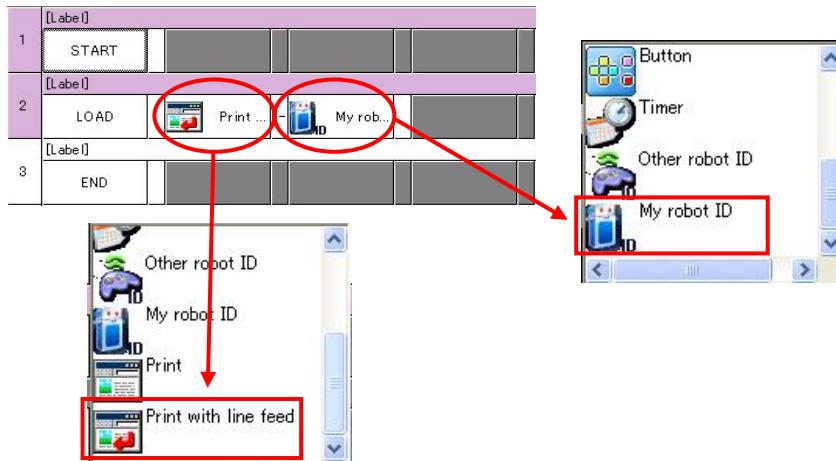
Use the communication set command to set the ID of the other. (Hexadecimal is used to configure the communication setting)

Baudrate :0F -> 57600	Baud rate is always set to 0F.
My ID. :02E6	Set the ID of Zig-100 that will 1:1 communicate in Dest.Addr.
Dest.Addr:02BC	
Wait Mode:No	Wait Mode is always in "No" mode.

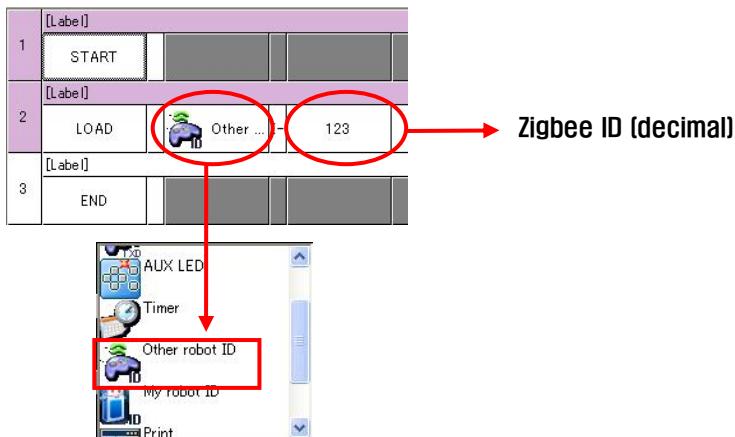
[The Zig-100 1:1 communication is set using a behavior control program]

The Zig-100 embedded on the CM-5 also requires configuration for communication. The CM-5 can set the ID of the opposing Zig-100 by using the behavior control program. For more details, refer to the Bioloid user's guide.

By using the "Print with line feed" and "My robot ID" of the CM-5, you can find the ID of embedded the Zig-100 as a decimal number..



Set the ID of the Zig-100 in decimal in "Other robot ID" item of the CM-5.



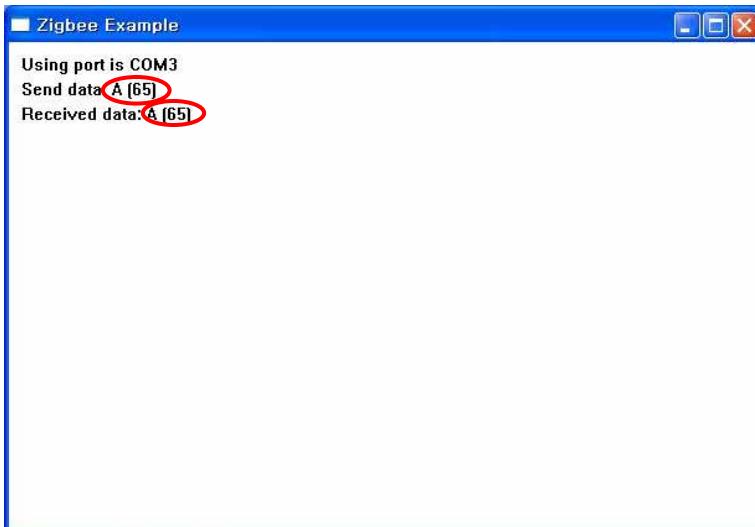
From the Bioloid SDK/windows/bin folder, download the Zigbee Example.bpg file to the CM-5 and execute it.

From the Bioloid SDK/windows/bin folder, open Zigbee Example.bat in a Notepad and change "COM3" to the name of connected port name of your PC. For example, if the port name is "COM5," it will become "Zigbee Example.exe COM5." When editing is finished, save and close the Notepad program.

From the Bioloid SDK/windows/bin folder, execute Zigbee Example.bat.

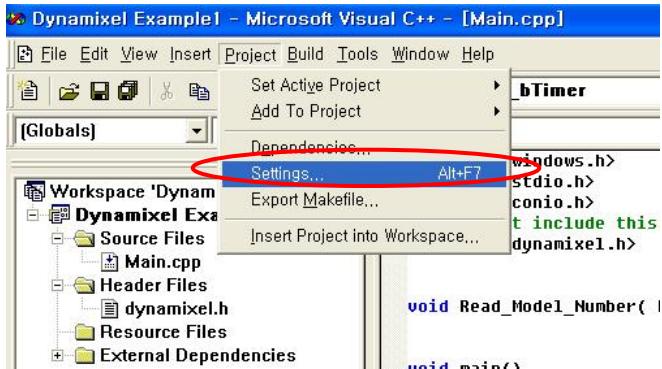
When a key is pressed, the current key value appears in Send data and the same key value is outputted in Received data.

This is because the received key values in the behavior control program of the CM-5 "echo" the pressed keys.

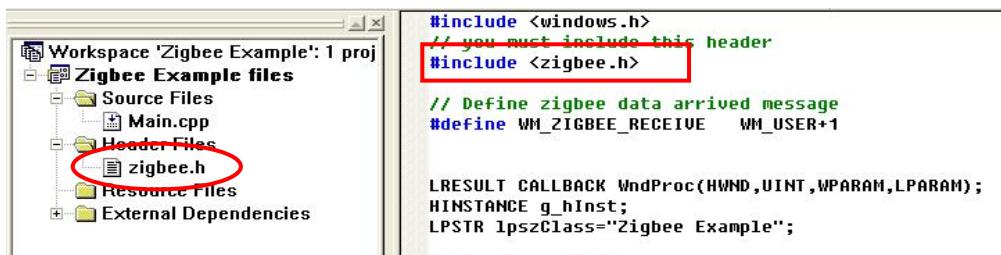
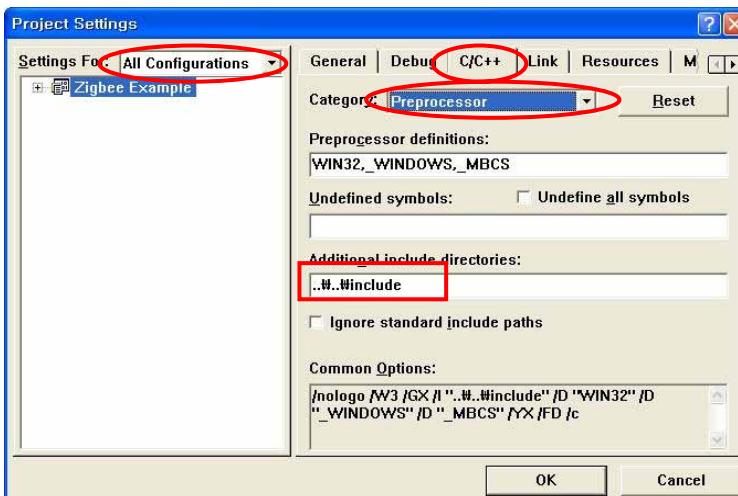


First, use Visual C++ 6.0 and open the project file, "Bioloid SDK/windows/app/Zigbee Example/Zigbee Example.dsw. To use the Bioloid SDK, you must include applicable header files and libraries. You can do this from this example.

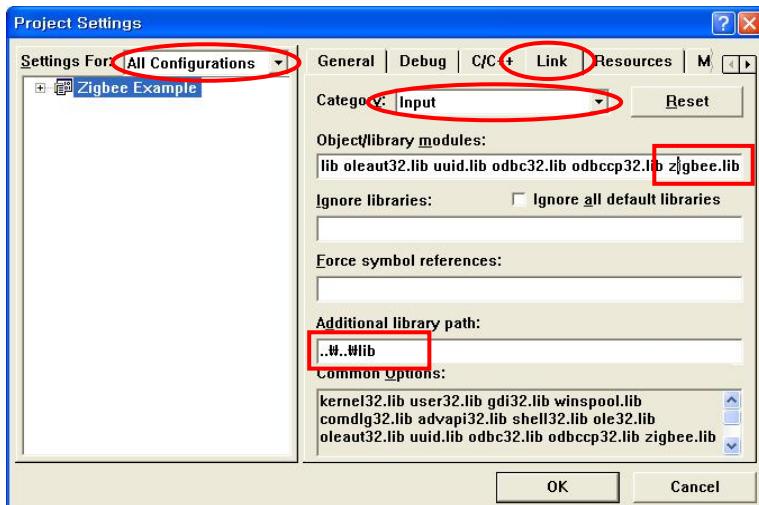
From menu, select Project → Settings...



Include the header file of the Zigbee library and specify the path.



Include the library file of the Zigbee library and specify the path.



Let's analyze the source of Main.cpp. Since the Zigbee Example is a win32 application program, you must have background knowledge with win32 programs. To find out the details on win32 programming, refer to the related books.

The first thing you must do in the main function is initialize the Zig2Serial by using zigbee_open.

```
#define WM_ZIGBEE_RECEIVE WM_USER+1

if( zigbee_open( PortName, hWnd, WM_ZIGBEE_RECEIVE ) == false )
{
    MessageBox( hWnd, "Fail to open Zigbee!", "Error" , MB_OK );
    PostQuitMessage(0);
}
```

As a parameter, zigbee_open function receives the name of port connected with Zig2Serial B/D, window handle (hWnd) for the message of received data and the message (WM_ZIGBEE_RECEIVE) used for giving notice of received data. The message of received data is able to be newly declared by user. If the initialization of library is successful, it will return "true" and if not, "false."

```
if( zigbee_clear() == false )
{
    MessageBox( hWnd, "Fail to clear buffer of Zigbee!", "Error" , MB_OK );
}
```

When the Zigbee library is initialized, you can clear the communication buffer by using the zigbee_clear function. Deleting unnecessary data improves the dependability.

```
SendData = (WORD)wParam;
if( zigbee_send( SendData ) == false )
{
    MessageBox( hWnd, "Fail to send data!", "Error" , MB_OK );
}
```

This is a routine that receives data from keyboard input and sends the data using the zigbee_send function. The zigbee_send function sends word type data to the CM-5 through Zigbee communication. It returns true when the transfer is successful and false if not.

```
case WM_ZIGBEE_RECEIVE:
    RcvData = (WORD)wParam;

    InvalidateRect( hWnd, NULL, TRUE );
    return 0;
```

Receiving data sent by the CM-5 to the PC is very simple. From a Windows procedure function that processes a message, you can check to see if a message assigned by the zigbee_open function has arrived. As shown in the above source, if the declared WM_ZIGBEE_RECEIVE message arrives, you can read WPARAM, a Window message parameter. When the wParam variable, a WPARAM type, is converted to a word type, it becomes Zigbee received data and can be used immediately.

```
case WM_PAINT:
    hdc = BeginPaint( hWnd, &ps );

    wsprintf( strText, "Using port is %s", PortName );
    TextOut( hdc, 10, 10, strText, strlen(strText) );
    wsprintf( strText, "Send data: %c (%d)", sendData, sendData );
    TextOut( hdc, 10, 30, strText, strlen(strText) );
    wsprintf( strText, "Received data: %c (%d)", RcvData, RcvData );
    TextOut( hdc, 10, 50, strText, strlen(strText) );

    EndPaint( hWnd, &ps );
    return 0;
```

The routine in charge of outputting data to the screen is currently using the Zigbee library to display the values of sendData, sent data, RcvData, and received data.

```
case WM_DESTROY:  
    zigbee_close();  
    PostQuitMessage(0);  
    return 0;
```

When the program closes, use the `zigbee_close` function to terminate the Zig2Serial.

◎ N:N Communication Setup method

Zig-100 units supports both 1:1 and N:N communication. 1:1 communication mode enables assigned IDs to communicate on a one on one basis, whereas, in N:N communication, any Zig-100 unit on a channel, can send and receive data. To enable N:N communication, use the following steps.

```
[Robotis Zigbee Monitor]
Baudrate :0F -> 57600
My ID. :02E6
Dest.Addr:FFFF
Command- (I)nfo, (D)est. Addr/(B)aud, (W)ait mode, (E)xit
!_
```

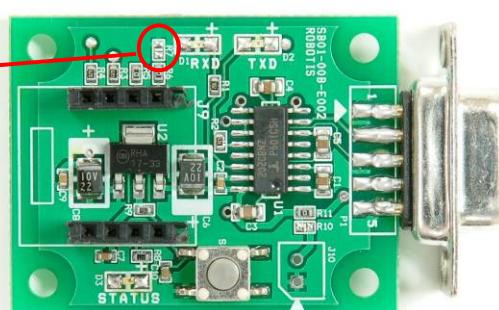
To enable N:N communication, the ID of opposing Zig-100 units must be set to FFFF in hexadecimal. When using robot terminal to set the Zig2Serial B/D, as shown above, Dest.Addr must be set to FFFF.

The Zig-100 attached to the CM-5 can set to N:N communication mode through the behavior control program. In the behavior control program, you can set the ID of opposing robots in decimal form, 65535 (in hexadecimal, FFFF).



When N:N communication is not possible after assigning the Zig-100 ID, check to see if the channel resistance of the Zig2Serial B/D is configured as the picture below shows.

Check to see if the R7
resistance is shorted.
If not, short it by
soldering

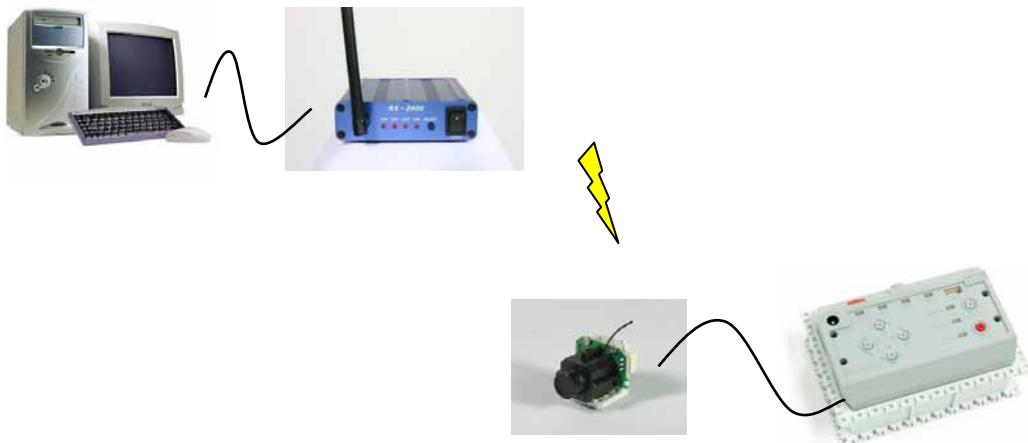


If you want the configuration so that only a specified Zig-100 unit can communicate with others in N:N communication, you can do so by assigning it an ID as a word type using send/receive data, which will distinguish it in the application or program.

2-2-4. Image Handling and Recognition

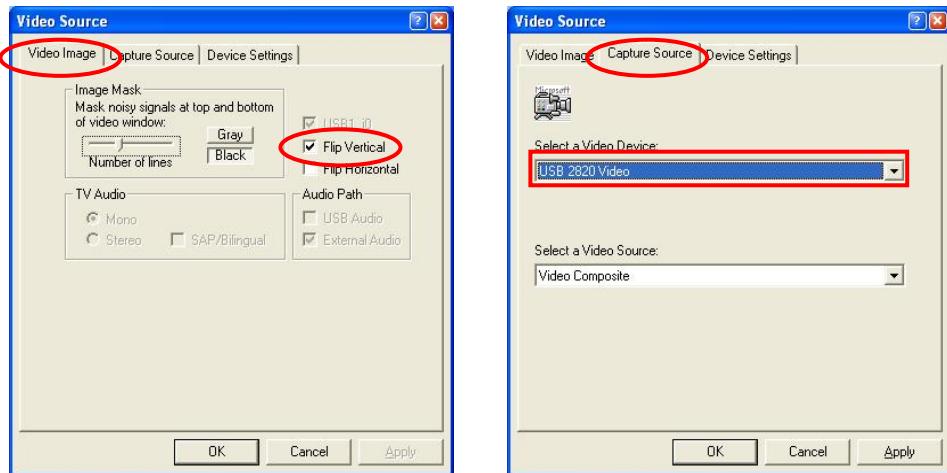
- ◎ Outputting the image to the screen

Construct the test environment as shown below.



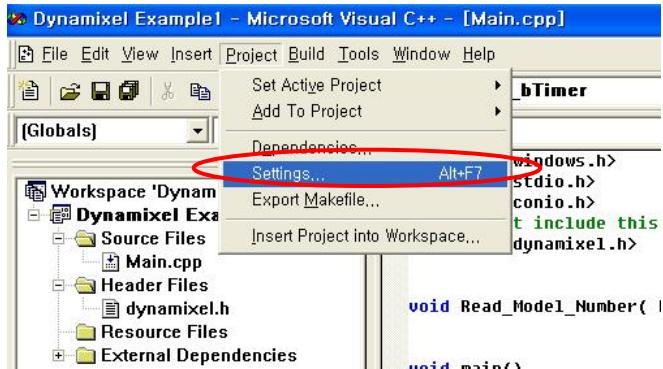
If the image does not appear on the monitor, check to see the image receiver and transmitter are using the same channel.

Execute Vision Example1.exe from the Bioloid SDK/windows/bin folder. When the following dialog box appears, configure the setting as follows.

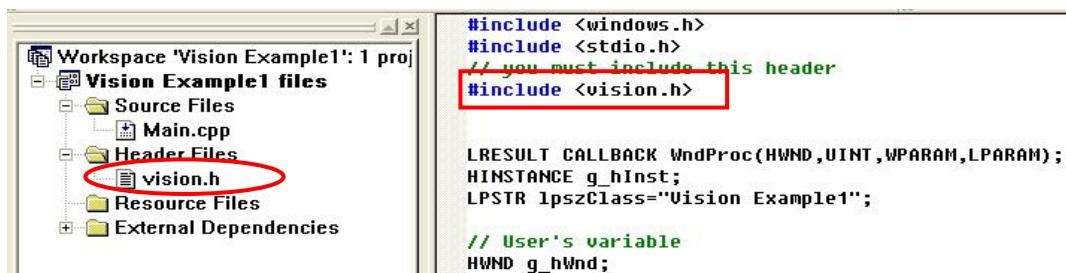
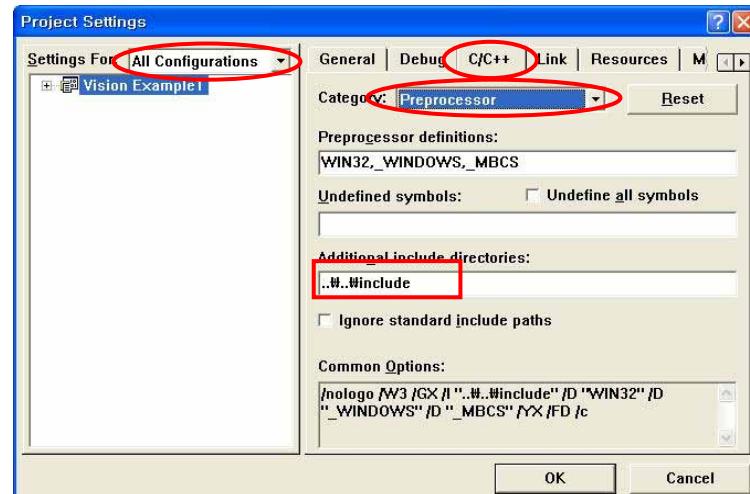


First, use Visual C++ 6.0 and open the project file, “Bioloid SDK/windows/app/Vision Example/Vision Example1.dsw.” To use the Bioloid SDK, you must include applicable header files and libraries.

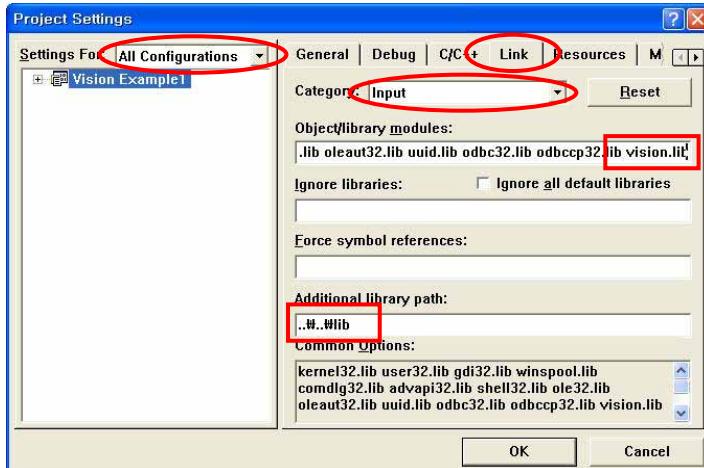
From the menu, select Project → Settings...



Include the header files from the Image Handling and Recognition's library and specify the path.



Include the library files from the Image Handling and Recognition's library and specify the path.



Include vision.lib

Let's analyze the source of Main.cpp.

Since the Vision Example 1 is a win32 application program, you must have background knowledge on win32 programs. To find out details on win32 programming, refer to applicable books.

```
VISION_PARAM VisionParam;
void ImageProcess( unsigned char* pImageData, DWORD dwDataSize );

VisionParam.width = 320;
VisionParam.height = 240;
VisionParam.pProcessFunc = ImageProcess;

if( vision_open( &VisionParam ) == false )
{
    MessageBox( hWnd, "Fail to open Vision!", "Error" , MB_OK );
    PostQuitMessage(0);
}

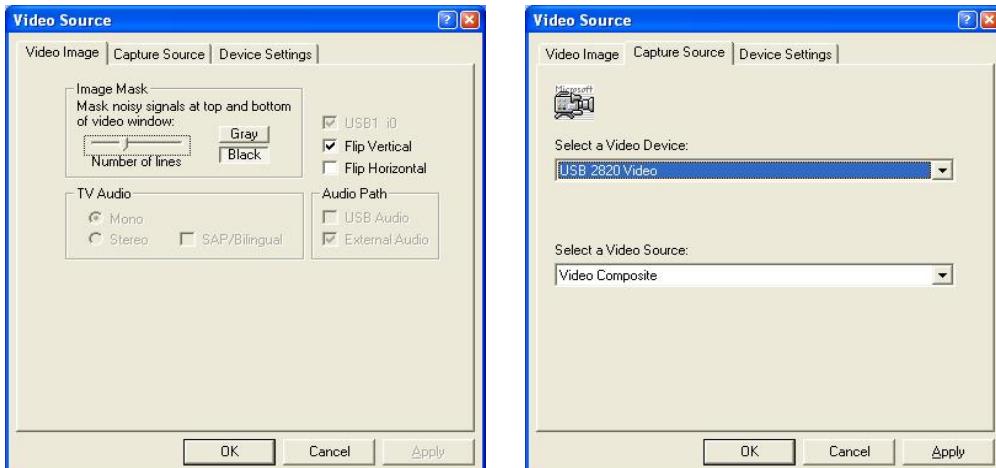
if( vision_start() == false )
{
    MessageBox( hWnd, "Fail to start Vision!", "Error" , MB_OK );
}
```

To use the wireless image receiver, you must initialize it using the `vision_open` function. Since the `vision_open` function uses the content of the `VISION_PARAM` structure to initialize the wireless image receiver, you first must include the appropriate value in `VISION_PARAM` structure. In the `VISION_PARAM` structure there are three values that can be set as shown below.

Member variable	Meaning
<code>width</code>	Width size of captured image data
<code>height</code>	Height size of captured image data
<code>pProcessFunc</code>	User function that will process image handling

This example captures a 320 x 240 size image and processes the image in the `ImageProcess` function.

When the `VISION_PARAM` structure is set in the `Vision_open` function, it returns either true or false depending on the success. When the `Vision_open` function is executed, the wireless image receiver dialog box appears, allowing you to set many configurations.



When initialization is successful, image capture must begin with the `vision_start` function. When this function is called, the `ImageProcess` function is called periodically and image processing is performed. Let's analyze the most important `ImageProcess` function.

```
void ImageProcess( unsigned char* pImageData, DWORD dwDataSize )
{
    HDC hdc;
    HBITMAP hBitmap;
    char strText[128];
    VISION_STATE VisionState;

    // Ready
    hdc = GetDC( g_hWnd );
    
```

```

// Display state
VisionState = vision_get_state();

sprintf( strText, "Width: %d", VisionState.width );
TextOut( hdc, 500, 30, strText, strlen(strText) );
sprintf( strText, "Hidth: %d", VisionState.height );
TextOut( hdc, 500, 45, strText, strlen(strText) );
sprintf( strText, "Fps: %d ", VisionState.fps );
TextOut( hdc, 500, 60, strText, strlen(strText) );
sprintf( strText, "Frame time: %.2fms ", VisionState.capTime_ms );
TextOut( hdc, 500, 75, strText, strlen(strText) );
sprintf( strText, "Process time: %.2fms ", VisionState.procTime_ms );
TextOut( hdc, 500, 90, strText, strlen(strText) );

// Display image
hBitmap = vision_RGB24toDDB( hdc, pImageData );
vision_DrawDDB( hdc, 10, 10, 450, 320, hBitmap );

InvalidateRect( g_hWnd, NULL, FALSE );

// Release
DeleteObject( hBitmap );
ReleaseDC( g_hWnd, hdc );
}

```

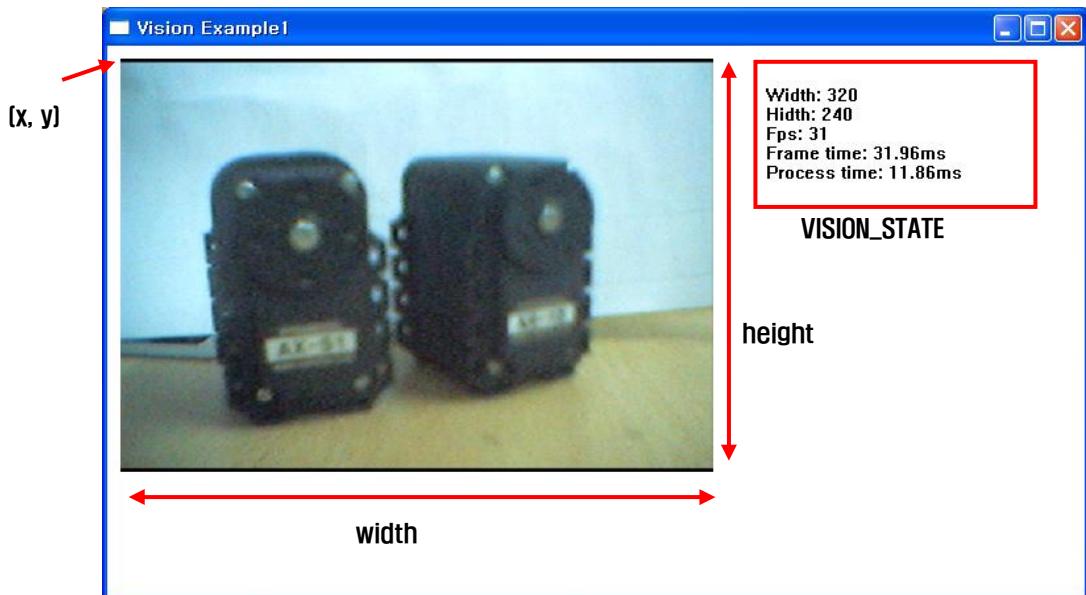
Callback is a function that receives Image data, **pImageData**, that is captured with the **ImageProcess** parameter with a **dwDataSize**, size of data (byte number), and is periodically called to handle the process.

You can use the **vision_get_state** function and check the state of the vision library. The **Vision_get_state** function includes required information in the **VISION_STATE** structure and returns it to the users.

Member variable	Meaning
width	Width size of captured image data
height	Height size of captured image data
fps	FPS(Frame Per Second)
capTime_ms	Captured time interval(msec)
procTime_ms	User function handling time(msec)

Here, in the case of **procTime_ms**, it is same as with **ImageProcess** function in regard to processing time. In the Vision Example1 case, the content of the **VISION_STATE** structure is outputted on the screen and the image from camera appears as is.

To output the captured image data on a screen, you must use a GDI related function from a Win32 program. The image processing and recognition library allows the GDI function to be used in a simple fashion and converts captured image data to HBITMAP type. The Vision_RGB24toDDB function receives HDC, a DC (Device Context) handle, and image data that will be converted as a parameter and converts to HBITMAP. Vision_DrawDDB function is used to actually display on the screen. The vision_DrawDDB function uses the HDC and range coordinates and HBITMAP is transferred.

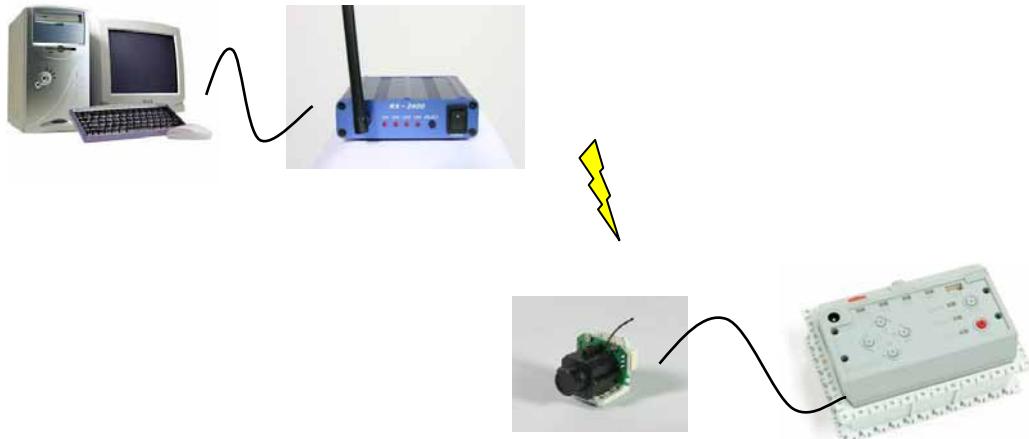


When using Vision_DrawDDB to output the image and to make the results appear on a monitor, you must also use InvalidateRect, a screen update Win32 API function. When closing the application program, use the vision_close function to terminate the wireless image receiver.

```
case WM_DESTROY:  
    vision_close();  
    PostQuitMessage(0);  
    return 0;
```

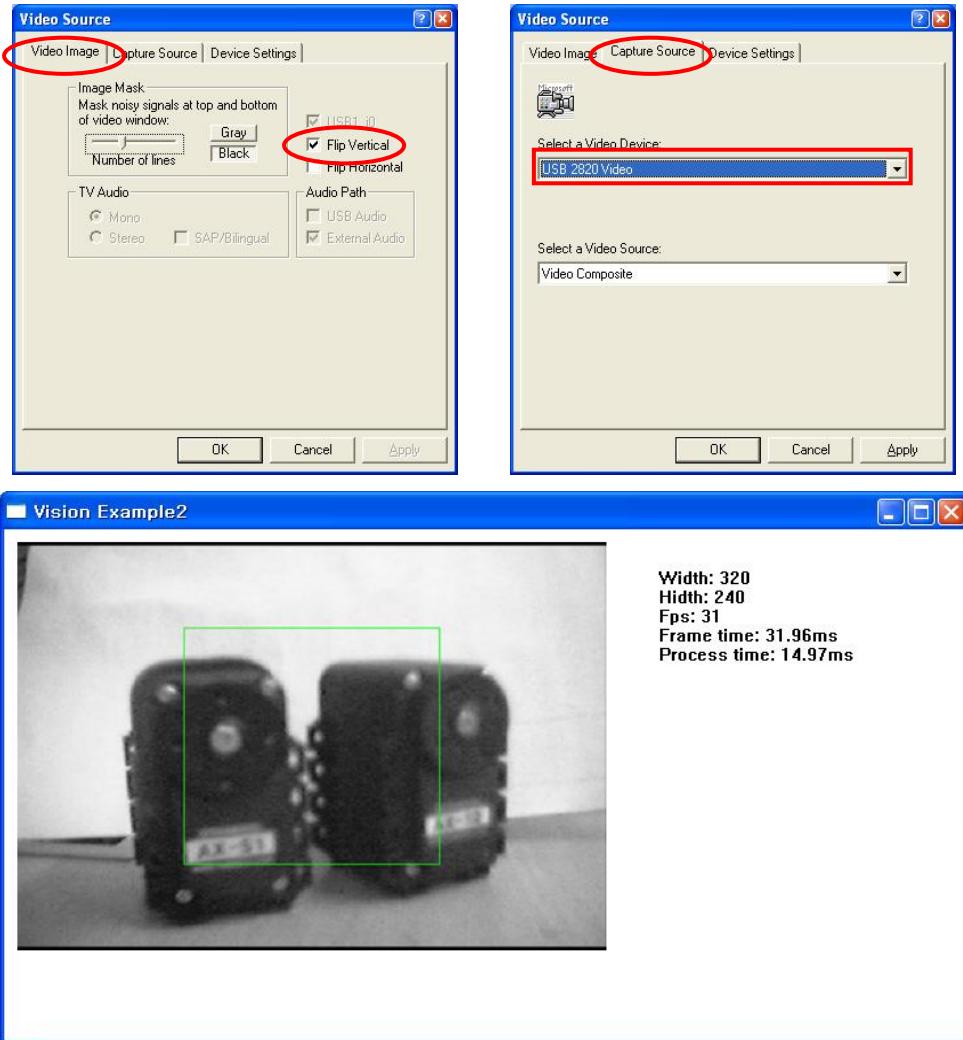
◎ Changing and outputting the image

Construct the test environment as shown below.



If the image does not appear on the monitor, check to see if the image receiver and transmitter are using the same channel.

Execute the Vision Example2.exe from the Bioloid SDK/windows/bin folder. When the following dialog box appears, configure the setting as follows.



First, use Visual C++ 6.0 and open the project file, “Bioloid SDK/windows/app/Vision Example2/Vision Example2.dsw.” For the basics of creating application program using the Vision library, refer to “Outputting the image to the screen.”

This example shows how you can process the captured image data and output the results on the screen.

Let's analyze the content of the ImageProcess function of Main.cpp.

```
void ImageProcess( unsigned char* pImageData, DWORD dwDataSize )
{
    HDC hdc;
    HDC hMemDC;
    HBITMAP hBitmap, hOldBitmap;
    HBRUSH hBrush;
    RECT rtFrame;
    char strText[128];
    VISION_STATE VisionState;

    VisionState = vision_get_state();

    RGB24toGray( pGrayImage, pImageData, VisionState.width, VisionState.height );

    // Ready
    hdc = GetDC( g_hWnd );
    hMemDC = CreateCompatibleDC( hdc );
    hBrush = CreateSolidBrush( RGB(0,255,0) );

    // Display state
    sprintf( strText, "Width: %d", VisionState.width );
    TextOut( hdc, 500, 30, strText, strlen(strText) );
    sprintf( strText, "Height: %d", VisionState.height );
    TextOut( hdc, 500, 45, strText, strlen(strText) );
    sprintf( strText, "Fps: %d ", VisionState.fps );
    TextOut( hdc, 500, 60, strText, strlen(strText) );
    sprintf( strText, "Frame time: %.2fms ", VisionState.capTime_ms );
    TextOut( hdc, 500, 75, strText, strlen(strText) );
    sprintf( strText, "Process time: %.2fms ", VisionState.procTime_ms );
    TextOut( hdc, 500, 90, strText, strlen(strText) );

    // Display image
    // Step1. Convert ImageData to DDB
    hBitmap = vision_GraytoDDB( hdc, pGrayImage );
```

```
// Step2. Draw image using win32 GDI
hOldBitmap = (HBITMAP)SelectObject( hMemDC, hBitmap );

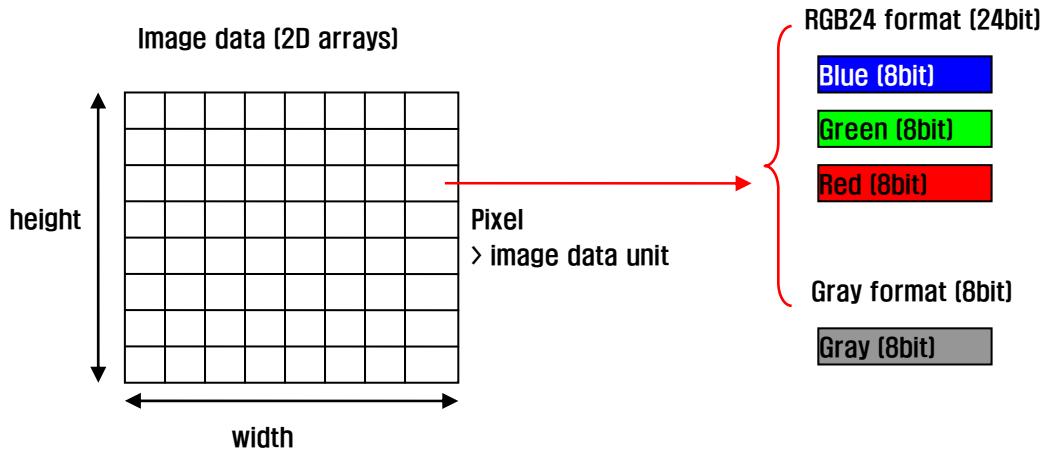
rtFrame.left = VisionState.width/2 - 70;
rtFrame.right = VisionState.width/2 + 70;
rtFrame.top = VisionState.height/2 - 70;
rtFrame.bottom = VisionState.height/2 + 70;
FrameRect( hMemDC, &rtFrame, hBrush );

SelectObject( hMemDC, hOldBitmap );
// Step3. Draw DDB
vision_DrawDDB( hdc, 10, 10, 450, 320, hBitmap );

InvalidateRect( g_hWnd, NULL, FALSE ); // Draw result

// Release
DeleteObject( hBitmap );
DeleteObject( hBrush );
ReleaseDC( g_hWnd, hMemDC );
ReleaseDC( g_hWnd, hdc );
}
```

The **RGB24toGray** function executes image processing by changing a color image to a grayscale image.



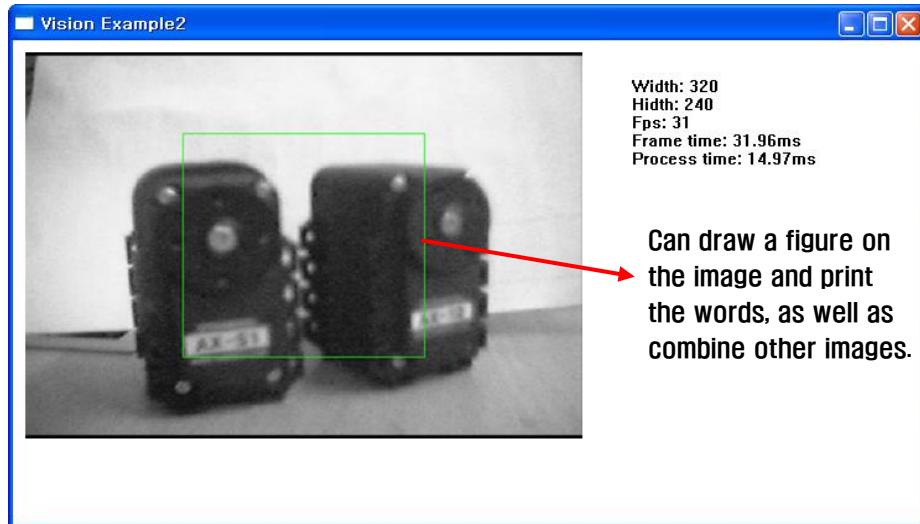
```
void RGB24toGray( unsigned char* pGray, unsigned char* pRGB24
                  ,int width, int height )
{
    int x, y, index;
    unsigned char r, g, b, gray;

    for( y=0; y<height; y++ )
    {
        for( x=0; x<width; x++ )
        {
            index = y * width + x;

            b = pRGB24[3*index];
            g = pRGB24[3*index+1];
            r = pRGB24[3*index+2];

            gray = (unsigned char)((r + g + b) / 3);
            pGray[index] = gray;
        }
    }
}
```

When the Vision_GraytoDDB function is used, Gray format image data can also be converted to HBITMAP. In this example, after changing the converted Gray format image to HBITMAP, the FrameRect function, a GDI function, draws a green square box.



```
HDC hdc;
HDC hMemDC;
HBITMAP hBitmap, hOldBitmap;
HBRUSH hBrush;
RECT rtFrame;

hdc = GetDC( g_hWnd );
hMemDC = CreateCompatibleDC( hdc );
hBrush = CreateSolidBrush( RGB(0,255,0) );

hBitmap = vision_GraytoDDB( hdc, pGrayImage );

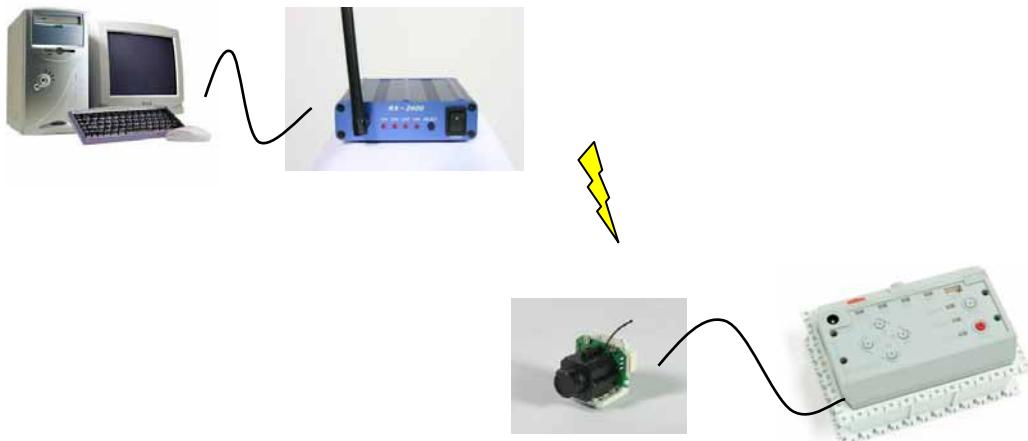
hOldBitmap = (HBITMAP)SelectObject( hMemDC, hBitmap );

rtFrame.left = VisionState.width/2 - 70;
rtFrame.right = VisionState.width/2 + 70;
rtFrame.top = VisionState.height/2 - 70;
rtFrame.bottom = VisionState.height/2 + 70;
FrameRect( hMemDC, &rtFrame, hBrush );

SelectObject( hMemDC, hOldBitmap );
```

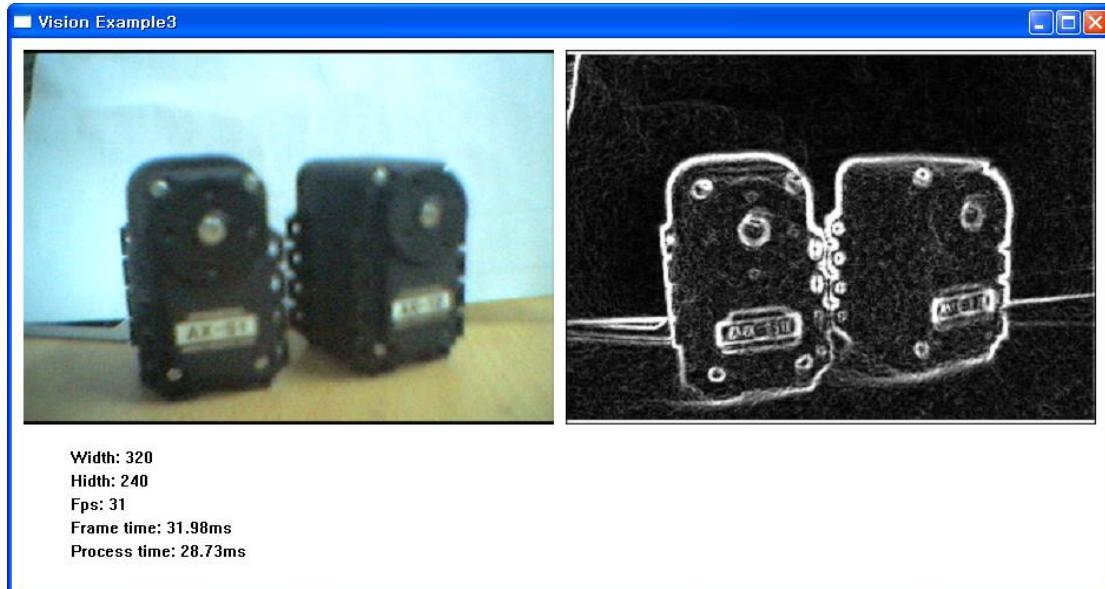
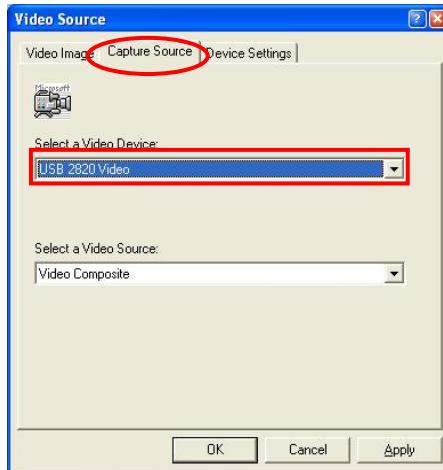
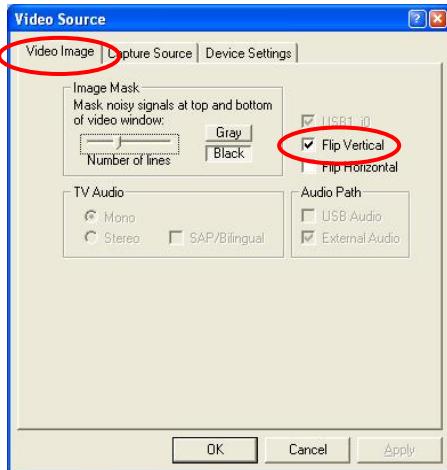
◎ Outputting multiple images to the screen

Construct the test environment as shown below.



If the image does not appear on the monitor, check to see if the image receiver and transmitter are using the same channel.

Execute the Vision Example3.exe from the Bioloid SDK/windows/bin folder. When the following dialog box appears, configure the setting as follows.



First, use Visual C++ 6.0 and open the project file, “Bioloid SDK/windows/app/Vision Example3/Vision Example3.dsw.” For the basics of creating application program using the Vision library, refer to the “Outputting the image to the screen.”

This example uses captured image data, extracts the outline, and outputs the two images, the original and result, on the screen. Let's analyze the ImageProcess function of Main.cpp.

```
void ImageProcess( unsigned char* pImageData, DWORD dwDataSize )
{
    HDC hdc;
    HBITMAP hBitmap1, hBitmap2;
    VISION_STATE VisionState;
    char strText[128];

    VisionState = vision_get_state();

    RGB24toGray( pGrayImage, pImageData, VisionState.width, VisionState.height );
    Sobel_Edge( pEdgeImage, pGrayImage, VisionState.width, VisionState.height );

    hdc = GetDC( g_hWnd );

    // Display image
    // Draw DDB of original image
    hBitmap1 = vision_RGB24toDDB( hdc, pImageData );
    vision_DrawDDB( hdc, 10, 10, 450, 320, hBitmap1 );

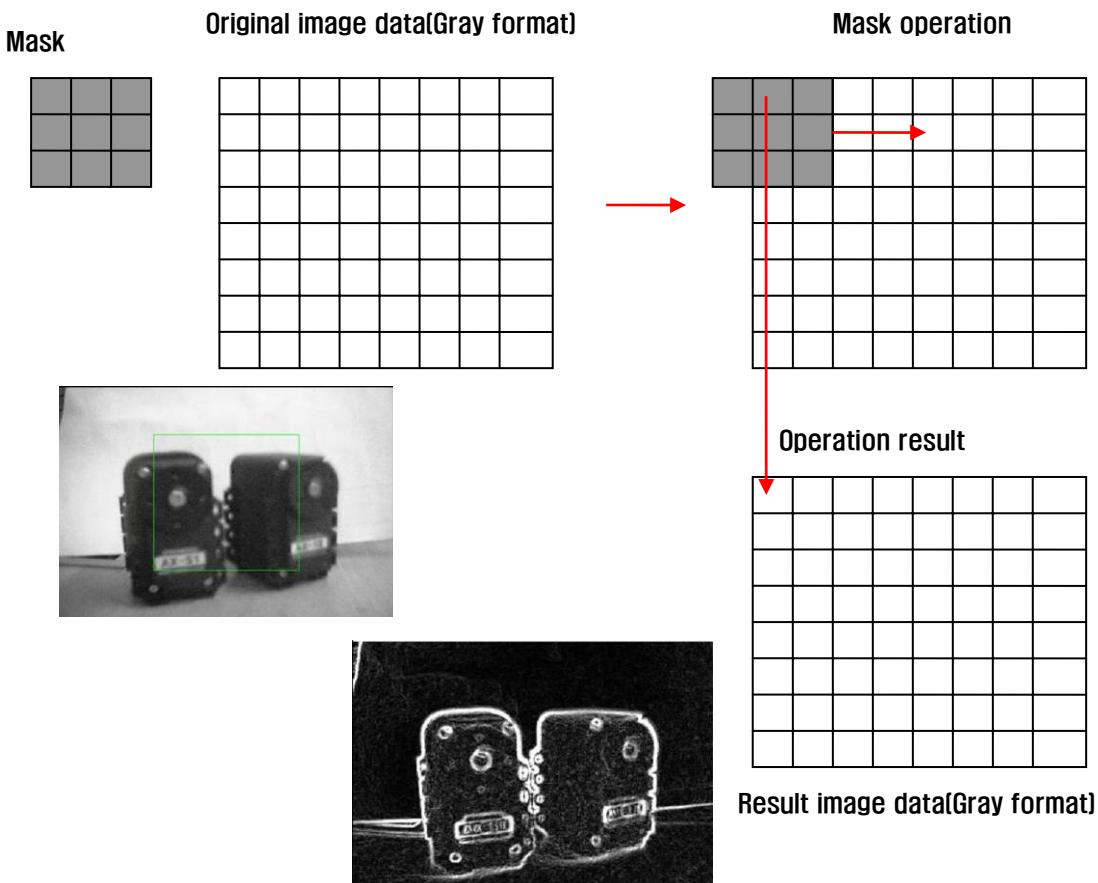
    // Draw DDB of converted image
    hBitmap2 = vision_GraytoDDB( hdc, pEdgeImage );
    vision_DrawDDB( hdc, 470, 10, 450, 320, hBitmap2 );

    // Display state
    sprintf( strText, "Width: %d", VisionState.width );
    TextOut( hdc, 50, 350, strText, strlen(strText) );
    sprintf( strText, "Hidth: %d", VisionState.height );
    TextOut( hdc, 50, 370, strText, strlen(strText) );
    sprintf( strText, "Fps: %d ", VisionState.fps );
    TextOut( hdc, 50, 390, strText, strlen(strText) );
    sprintf( strText, "Frame time: %.2fms ", VisionState.capTime_ms );
    TextOut( hdc, 50, 410, strText, strlen(strText) );
    sprintf( strText, "Process time: %.2fms ", VisionState.procTime_ms );
    TextOut( hdc, 50, 430, strText, strlen(strText) );

    InvalidateRect( g_hWnd, NULL, FALSE ); // Draw result
```

```
// Release  
DeleteObject( hBitmap1 );  
DeleteObject( hBitmap2 );  
ReleaseDC( g_hWnd, hdc );  
}
```

The main algorithm of the ImageProcess function is as follows. After converting pImageData, an RGB24 format to pGrayImage, a Gray format, it uses the Sobel_Edge function to create pEdgelImage, which extracts only the outline. Next, by using vision_RGB24toDDB and vision_GraytoDD, it creates hBitmap1 and hBitmap2, a HBITMAP, and by using the vision_DrawDDB function in HDC, which will output the results in Windows, it outputs two images. In this example, a new function Sobel_Edge is introduced. This function receives Gray format image data, the width and height size parameter, and creates an output of Gray format image data that displays only the outline. There are various outline extraction methods. Some methods include: Sobel, which uses a mask, the Prewitt, Robert, and Laplacian method. In this example, we used the Sobel mask method.



```
void Sobel_Edge( unsigned char* pResult, unsigned char* pOrigin, int width, int
height )
{
    int x, y, i, j, index;
    int sum_x, sum_y, sum, origin;
    int gx_mask[3][3] = {-1, 0, 1, -2, 0, 2, -1, 0, 1}; // Sobel gx mask
    int gy_mask[3][3] = {1, 2, 1, 0, 0, 0, -1, -2, -1}; // Sobel gy mask
    int* pTempImage;
    int min, max, newValue;
    float constVal1, constVal2;

    pTempImage = new int[width*height];

    // Edge detection by Sobel mask
    for( y=0; y<height; y++ )
    {
        for( x=0; x<width; x++ )
        {
            sum = 0;

            if( (x != 0 && x != width-1) && (y != 0 && y != height-1) )
            {
                sum_x = 0;
                sum_y = 0;

                for( j=-1; j<=1; j++ )
                {
                    for( i=-1; i<=1; i++ )
                    {
                        origin = (int)pOrigin[(y+j) * width + (x+i)];
                        sum_x += origin * gx_mask[j+1][i+1];
                        sum_y += origin * gy_mask[j+1][i+1];
                    }
                }
                sum = abs(sum_x) + abs(sum_y);
                sum = max(0, min(sum, 255));
            }

            index = y*width + x;
            pTempImage[index] = sum;
        }
    }
}
```

```
// Find min, max value
min = (int)10e10;
max = (int)-10e10;
for( y=0; y<height; y++ )
{
    for( x=0; x<width; x++ )
    {
        index = y*width + x;
        if( pTempImage[index] < min )
            min = pTempImage[index];
        if( pTempImage[index] > max )
            max = pTempImage[index];
    }
}

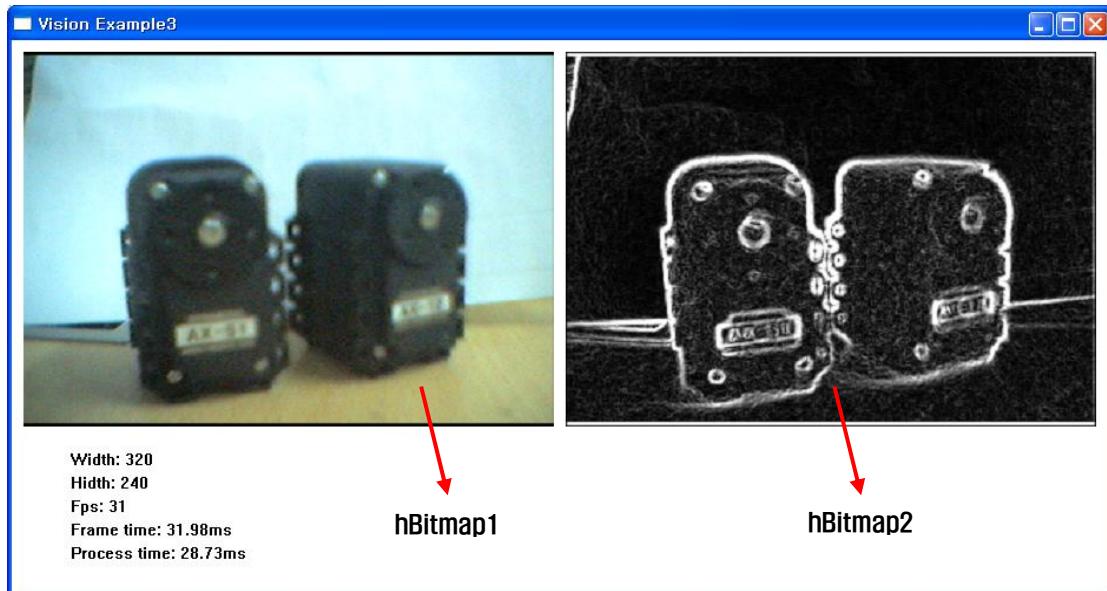
// Normalize level
constVal1 = (float)(255.0 / (max-min));
constVal2 = (float)(255.0*min / (max-min));
for( y=0; y<height; y++ )
{
    for( x=0; x<width; x++ )
    {
        index = y*width + x;
        newValue = (int)(constVal1 * (float)pTempImage[index] + constVal2);
        pResult[index] = (unsigned char)newValue;
    }
}

delete pTempImage;
}
```

It is very simple to output two or more images on the screen. By using Vision_DrawDDB, you only have to assign range coordinates for where the images will be outputted.

```
hBitmap1 = vision_RGB24toDDB( hdc, pImageData );
vision_DrawDDB( hdc, 10, 10, 450, 320, hBitmap1 );

hBitmap2 = vision_GraytoDDB( hdc, pEdgeImage );
vision_DrawDDB( hdc, 470, 10, 450, 320, hBitmap2 );
```

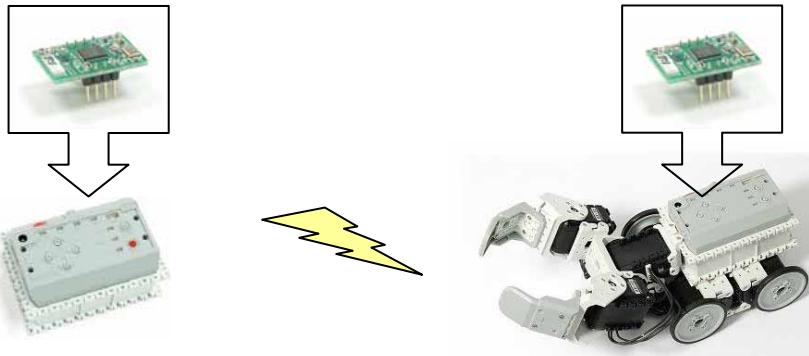


There are many image processing methods apart from the aforementioned methods. The purpose of this example is to use the wireless image receiver to acquire and capture image data taken by the wireless camera, process the image, and output it on the screen. Since this manual is insufficient for gaining thorough knowledge in image processing and recognition, we recommend that you investigate other books that cover this topic in more detail.

3. Applications of the Bioloid Expert Level Educational Kit

3 – 1 . Controlling the Probe Robot Via Remote Control

This chapter uses the probe robot, an intermediate level robot example from the Bioloid comprehensive kit. For this example, the probe robot must have a Zig-100 unit embedded and both Zig-100 units must be set so that the communication mode is 1:1.



◆ Program download

- Remote control

Use either “Bioloid SDK \ CM-5 \ bin \ example_remocon.hex” or “Bioloid SDK \ CM-5 \ bin \ example_remocon.bpg” in the expansion CD.

- Probe robot

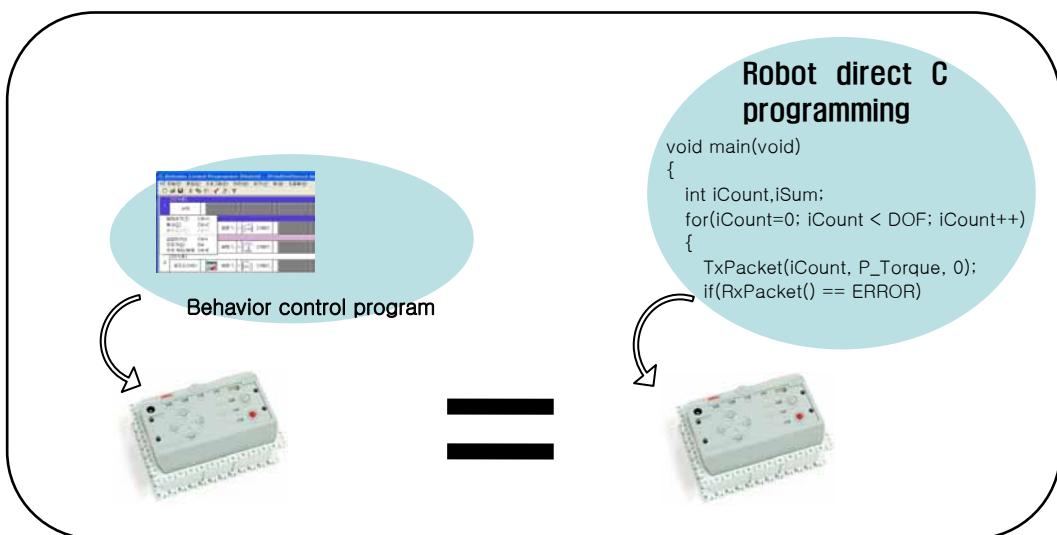
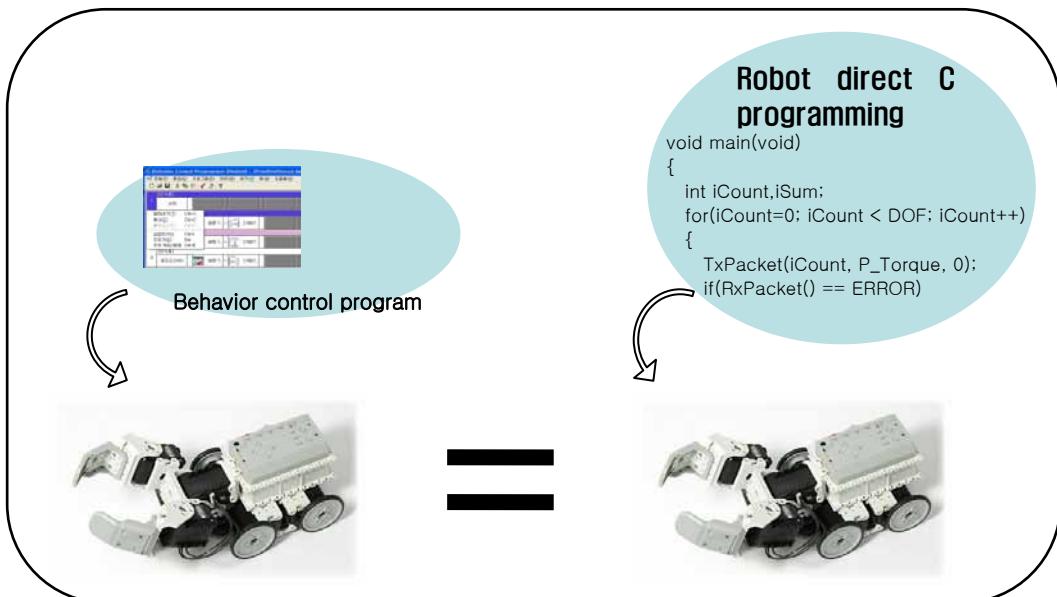
Use either “Bioloid SDK \ CM-5 \ bin \ example_probingrobot.hex” or “Bioloid SDK \ CM-5 \ bin \ example_probingrobot.bpg” in the expansion CD.

◆ Robot Operation

< Remote control operation method >

Button	Function
UP	Forward
DOWN	Backward
LEFT	Left turn
RIGHT	Right turn
START+UP	Raise a claw
START+DOWN	Bring down a claw
START+LEFT	Open a claw
START+RIGHT	Close a claw

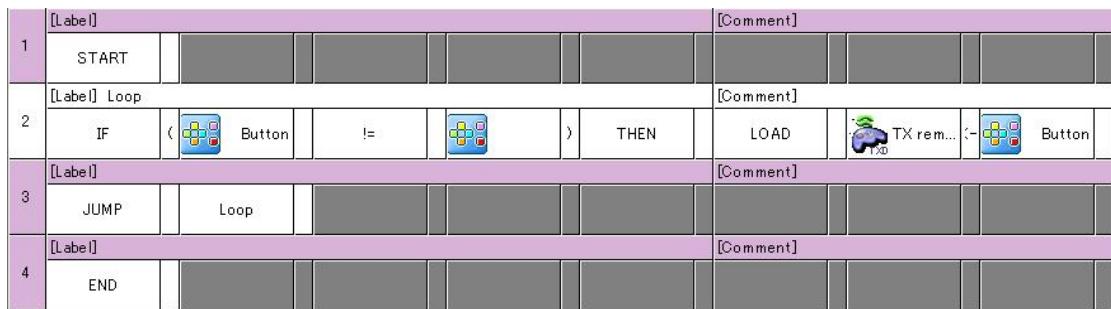
This chapter shows how a C language program that using the CM-5 library can act the same as the behavior control program.



◎ Analyzing the remote control program

This example can be found in the “Bioloid SDK \CM-5 \app \example_remocon \example_remocon.c” of expansion CD. The companion behavior control program can be found in “Bioloid SDK \CM-5 \bin \ example_remocon.bpg,” also in the expansion CD.

The remote behavior control program of the probe robot is very simple. As long as the button state is not 0 (any button is pressed), you can repeat the transmission of the button state value as a wireless data.



The next figure shows how to construct the above in C language using the CM-5 library.

```
int main(void)
{
    PortInitialize();
    TimerInitialize(TIMER0, 128, 1); //Interrupt Enable
    SerialInitialize(SERIAL_PORT0,1,RX_INTERRUPT);//485
    SerialInitialize(SERIAL_PORT1,DEFAULT_BAUD_RATE,RX_INTERRUPT); //232
    ChargeInInterruptInitialize();
    sei();
    TxDString("\r\n CM-5 Example--- Remote Controller\r\n");
    RxD_ZIGBEE;
    byte bKey;

    while(1) {
        bKey = GetButtonState();
        if( bKey != 0 ) {
            ZigbeePacketTransfer( bKey );
        }
    }
    return 0;
}
```

The bolded words highlight the important code, and illustrate how easy the program is to construct with the C library.

◎ Analyzing the probe robot program

This example can be found in the Bioloid SDK \CM-5 \app \example_probingrobot \example_probingrobot.c. The companion behavior control program can be found in Bioloid SDK \CM-5 \bin \example_probingrobot.bpg" also in the expansion CD.

Next shows the motor initialization part that is executed from the starting point of the probe robot' s behavior control program, as controlled by a remote control.

23	[Label] Initialize Motor					[Comment]		
	IF	(1:8)	!=	0)	THEN
24	LOAD		1:8	<-	0			[Comment]
	IF	(2:8)	!=	0)	THEN
25	LOAD		2:8	<-	0			[Comment]
	IF	(3:8)	!=	0)	THEN
26	LOAD		3:8	<-	0			[Comment]
	IF	(4:8)	!=	0)	THEN
27	LOAD		4:8	<-	0			[Comment]
	IF	(5:8)	=	0)	THEN
28	LOAD		5:8	<-	1023			[Comment]
	IF	(6:8)	=	0)	THEN
29	LOAD		6:8	<-	1023			[Comment]
	IF	(7:8)	=	0)	THEN
30	LOAD	SPEED	[5]Dyn...	<-	400			[Comment]
	LOAD	SPEED	[6]Dyn...	<-	400			[Comment]
31	LOAD	SPEED	[7]Dyn...	<-	400			[Comment]
	LOAD	SLOPE	[6]Dyn...	<-	64			[Comment]
32	LOAD	SLOPE	[7]Dyn...	<-	64			[Comment]
	LOAD	SLOPE	[6]Dyn...	<-	64			[Comment]
33	LOAD	SLOPE	[7]Dyn...	<-	64			[Comment]
	LOAD	TORQUE	[6]Dyn...	<-	380			[Comment]
34	LOAD	TORQUE	[7]Dyn...	<-	380			[Comment]
	LOAD	TORQUE	[6]Dyn...	<-	380			[Comment]
35	LOAD	TORQUE	[7]Dyn...	<-	380			[Comment]
	RETURN							[Comment]
36								
37								
38								
39								

For the AX-12 motors that have an ID 1~4, you have to change the value of CCW_ANGLE_LIMIT [Address 0x08] to 0 from the control table so that the motors operate in endless turn mode since

the motors act as driving wheels for the probe robot. Also, the value of CCW_ANGLE_LIMIT must be set to 1023 for motors with IDs 5~7 of the claw arm since they need to operate in a normal mode. Moreover, control is needed for the rotation speed, compliance slope, torque limit, and others so that the motors do not have excessive load from the claw arm.

The next figure shows how to construct the above in C language using the CM-5 library.

```
void MotorInitialize(void)
{
    if( ReadWord(1,P_CCW_ANGLE_LIMIT_L )!=0 ) WriteWord(1,P_CCW_ANGLE_LIMIT_L, 0);
    if( ReadWord(2,P_CCW_ANGLE_LIMIT_L )!=0 ) WriteWord(2,P_CCW_ANGLE_LIMIT_L, 0);
    if( ReadWord(3,P_CCW_ANGLE_LIMIT_L )!=0 ) WriteWord(3,P_CCW_ANGLE_LIMIT_L, 0);
    if( ReadWord(4,P_CCW_ANGLE_LIMIT_L )!=0 ) WriteWord(4,P_CCW_ANGLE_LIMIT_L, 0);
    if( ReadWord(5,P_CCW_ANGLE_LIMIT_L )==0 ) WriteWord(5,P_CCW_ANGLE_LIMIT_L, 1023);
    if( ReadWord(6,P_CCW_ANGLE_LIMIT_L )==0 ) WriteWord(6,P_CCW_ANGLE_LIMIT_L, 1023);
    if( ReadWord(7,P_CCW_ANGLE_LIMIT_L )==0 ) WriteWord(7,P_CCW_ANGLE_LIMIT_L, 1023);
    WriteWord( 5, P_GOAL_SPEED_L, 400 );
    WriteWord( 6, P_GOAL_SPEED_L, 400 );
    WriteWord( 7, P_GOAL_SPEED_L, 400 );
    WriteByte( 6, P_CW_COMPLIANCE_SLOPE, 64 );
    WriteByte( 6, P_CCW_COMPLIANCE_SLOPE, 64 );
    WriteByte( 7, P_CW_COMPLIANCE_SLOPE, 64 );
    WriteByte( 7, P_CCW_COMPLIANCE_SLOPE, 64 );
    WriteWord( 6, P_TORQUE_LIMIT_L, 380 );
    WriteWord( 7, P_TORQUE_LIMIT_L, 380 );
}
```

When compared to the included behavior control program, you can see that the C code is the same line by line.

Next is the C language program that stops a robot.

41	[Label]	Stop					
	LOAD	SPEED [1]Dyn...	<-	0			
42	[Label]						
	LOAD	SPEED [2]Dyn...	<-	0			
43	[Label]						
	LOAD	SPEED [3]Dyn...	<-	0			
44	[Label]						
	LOAD	SPEED [4]Dyn...	<-	0			
45	[Label]						
	RETURN						

```
void Stop(void)
{
    WriteWord( 1, P_GOAL_SPEED_L, 0 );
    WriteWord( 2, P_GOAL_SPEED_L, 0 );
    WriteWord( 3, P_GOAL_SPEED_L, 0 );
    WriteWord( 4, P_GOAL_SPEED_L, 0 );
}
```

To turn off an AX-12 motor in the endless turn mode, include 0 in the Goal Speed parameter.

The behavior control program and C language function that moves the robot forward is as follows.

	[Label] Forward		
47	LOAD	SPEED [1] Dyn... :-	Comple...
	[Label]		
48	LOAD	SPEED [2] Dyn... :-	Speed
	[Label]		
49	LOAD	SPEED [3] Dyn... :-	Comple...
	[Label]		
50	LOAD	SPEED [4] Dyn... :-	Speed
	[Label]		
51	RETURN		

```
void Forward(void)
{
    WriteWord( 1, P_GOAL_SPEED_L, COMPLEMENT_SPEED );
    WriteWord( 2, P_GOAL_SPEED_L, MAX_SPEED );
    WriteWord( 3, P_GOAL_SPEED_L, COMPLEMENT_SPEED );
    WriteWord( 4, P_GOAL_SPEED_L, MAX_SPEED );
}
```

When controlling AX-12 in endless turn mode, only the rotation direction and speed are controllable, which is accomplished by writing a value in the Goal Speed parameter. To move the robot forward the left AX-12 rotates forward and the right in reverse.

The speed maintenance variables that are defined using #define as follows in C language.

	[Label]		
3	LOAD	Speed	:- 1023
	[Label]		
4	COMPUTE	Comple...	= 1024 + Speed

```
#define MAX_SPEED 1023
#define COMPLEMENT_SPEED (1024+MAX_SPEED)
```

The reason for adding 1024 as shown above is because among the values that represent the speed of endless turn mode, the highest bit represents the motor's rotation direction.

By changing the speed values as shown for a wheel that uses endless turn mode, the program can control forward, reverse, left turn, and right turn movements.

The behavior control program and C language function that moves the robot in reverse is as follows.

	[Label] Backward		
53	LOAD	SPEED [1]Dyn... :-	Speed
54	[Label]		
55	LOAD	SPEED [2]Dyn... :-	Comple...
56	[Label]		
57	LOAD	SPEED [3]Dyn... :-	Speed
	[Label]		
	LOAD	SPEED [4]Dyn... :-	Comple...
	[Label]		
	RETURN		

```
void Backward(void)
{
    WriteWord( 1, P_GOAL_SPEED_L, MAX_SPEED );
    WriteWord( 2, P_GOAL_SPEED_L, COMPLEMENT_SPEED );
    WriteWord( 3, P_GOAL_SPEED_L, MAX_SPEED );
    WriteWord( 4, P_GOAL_SPEED_L, COMPLEMENT_SPEED );
}
```

Turn the motor just the opposite direction of the forward movement explained earlier.

The behavior control program and C language function that turns the robot right is as follows.

	[Label] Right		
59	LOAD	SPEED [1]Dyn... :-	Speed
60	[Label]		
61	LOAD	SPEED [2]Dyn... :-	Speed
62	[Label]		
63	LOAD	SPEED [3]Dyn... :-	Speed
	[Label]		
	LOAD	SPEED [4]Dyn... :-	Speed
	[Label]		
	RETURN		

```
void Right(void)
{
    WriteWord( 1, P_GOAL_SPEED_L, MAX_SPEED );
    WriteWord( 2, P_GOAL_SPEED_L, MAX_SPEED );
    WriteWord( 3, P_GOAL_SPEED_L, MAX_SPEED );
    WriteWord( 4, P_GOAL_SPEED_L, MAX_SPEED );
}
```

To turn, the right motors move in reverse and left motor move forward, making the robot turn right.

The behavior control program and C language function that turns the robot left is as follows.

Similar to the forward/backward movement reversal, reversing the motor directions of the right turn motion explained earlier results in a left turn.

65	[Label] Left	LOAD		[1]Dyn... :-	Comple...
66	[Label]	LOAD		[2]Dyn... :-	Comple...
67	[Label]	LOAD		[3]Dyn... :-	Comple...
68	[Label]	LOAD		[4]Dyn... :-	Comple...
69	[Label]	RETURN			

```
void Left(void)
{
    WriteWord( 1, P_GOAL_SPEED_L, COMPLEMENT_SPEED );
    WriteWord( 2, P_GOAL_SPEED_L, COMPLEMENT_SPEED );
    WriteWord( 3, P_GOAL_SPEED_L, COMPLEMENT_SPEED );
    WriteWord( 4, P_GOAL_SPEED_L, COMPLEMENT_SPEED );
}
```

The behavior control program and C language function that closes the claw of robot is as follows.

71	[Label] Closing Gripper	LOAD		[6]Dyn... :-	670
72	[Label]	LOAD		[7]Dyn... :-	352
73	[Label]	LOAD		[100]Dyn... :-	255
74	[Label]	LOAD		[100]Dyn... :-	14
75	[Label]	CALL		Motion...	
76	[Label]	RETURN			

```
void ProbeClose(void)
{
    WriteWord( 6, P_GOAL_POSITION_L, 670 );
    WriteWord( 7, P_GOAL_POSITION_L, 352 );
    WriteByte( 100, P_BUZZER_DATA1, 255 );
    WriteByte( 100, P_BUZZER_DATA0, 14 );
    WaitMotionEnd();
}
```

The reason for writing a value for the buzzer is so that when the claw opens, the buzzer will start playing a specific melody.

The WaitMotionEnd function, which was called in the last part is marked as "Motion Done" in the behavior control program, is shown as follows.

	[Label] Motion Done						[Comment]	
97	IF	(MOVING [5]C?..	!=	0) OR			
	[Label]						[Comment]	
98	CONT IF	(MOVING [6]C?..	!=	0) OR			
	[Label]						[Comment]	
99	CONT IF	(MOVING [7]C?..	!=	0) THEN	JUMP	Motion...	
	[Label]						[Comment]	
100	RETURN							

```
void WaitMotionEnd(void)
{
    while( (.ReadByte(5,P_MOVING) !=0 ) ||
           (.ReadByte(6,P_MOVING) !=0 ) ||
           (.ReadByte(7,P_MOVING) !=0 ) );
}
```

This function makes the program stand by until motors with IDs 5~7 stop, which equates to the claw finishing the closing/opening movement. As you can see, “if” and other condition statements from the behavior control program can be converted to C language.

The behavior control program and C language function that opens the claw of robot is as follows.

	[Label] Opening Gripper						
78	LOAD	POS [6]Dyn...	-	492			
	[Label]						
79	LOAD	POS [7]Dyn...	-	532			
	[Label]						
80	LOAD	[100]D...	-	255			
	[Label]						
81	LOAD	[100]D...	-	13			
	[Label]						
82	CALL	Motion...					
	[Label]						
83	RETURN						

```
void ProbeOpen(void)
{
    WriteWord( 6, P_GOAL_POSITION_L, 492 );
    WriteWord( 7, P_GOAL_POSITION_L, 532 );
    WriteByte( 100, P_BUZZER_DATA1, 255 );
    WriteByte( 100, P_BUZZER_DATA0, 13 );
    WaitMotionEnd();
}
```

By changing the value of the Goal Position to an already known position, the claw opens.

The behavior control program and C language function that lowers the claw of robot is as follows.

	[Label] Put down Gripper			
85	LOAD	 POS [5]Dyn...	[-	465
	[Label]			
86	LOAD	 [100]D...	[-	255
	[Label]			
87	LOAD	 [100]D...	[-	1
	[Label]			
88	CALL	Motion...		
	[Label]			
89	RETURN			

```
void ProbeDown(void)
{
    WriteWord( 5, P_GOAL_POSITION_L, 465 );
    WriteByte( 100, P_BUZZER_DATA1, 255 );
    WriteByte( 100, P_BUZZER_DATA0, 1 );
    WaitMotionEnd();
}
```

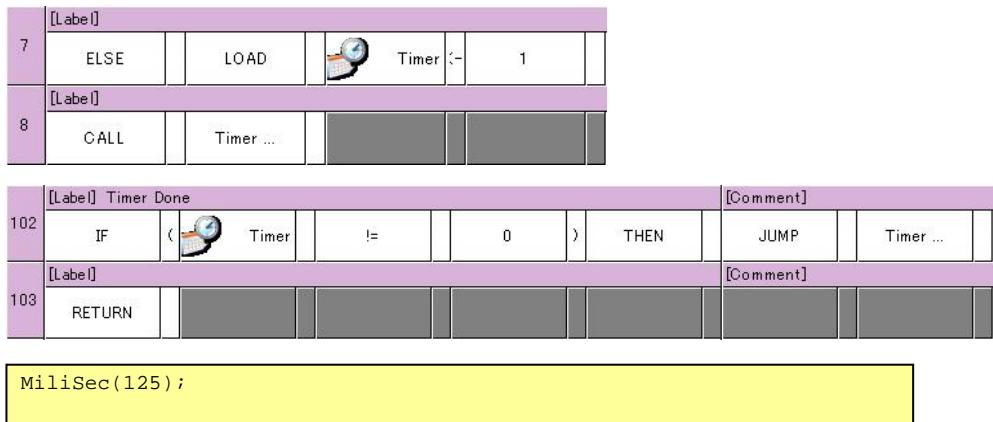
Raising and lowering claw only requires the AX-12 motor with an ID of 5 making the motion even simpler.

The behavior control program and C language function that raises up the claw of robot is as follows.

	[Label] Lifting Gripper			
91	LOAD	 POS [5]Dyn...	[-	825
	[Label]			
92	LOAD	 [100]D...	[-	255
	[Label]			
93	LOAD	 [100]D...	[-	0
	[Label]			
94	CALL	Motion...		
	[Label]			
95	RETURN			

```
void ProbeUp(void)
{
    WriteWord( 5, P_GOAL_POSITION_L, 825 );
    WriteByte( 100, P_BUZZER_DATA1, 255 );
    WriteByte( 100, P_BUZZER_DATA0, 0 );
    WaitMotionEnd();
}
```

Among the used functions, there is a timer function that delays the time.



To delay the time in the behavior control program, you have to use the CM-5 timer. However, since the timer is in increments 0.125sec, you have to write a loop condition to continually check the value, making the delay rather inconvenient.

However, in the C library, only one line for a time delay function needs to be written. This function allows for timer increments of 1/1000sec, making the C language much easier to use than the behavior control program for this case.

Lastly, the main loop routine of the behavior control is as follows.

	[Label] Loop							[Comment]	
6	IF	( RX rem... rx	=	1)	THEN	JUMP	Analyz...
	[Label]							[Comment]	
7	ELSE		LOAD	 Timer	-	1			
	[Label]							[Comment]	
8	CALL		Timer ...						
	[Label]							[Comment]	
9	CALL		Stop						
	[Label]							[Comment]	
10	JUMP		Loop						
	[Label]							[Comment]	
11	Double click!								
	[Label] Analyzing Data							[Comment]	
12	LOAD		Button	 RX rem... rx					
	[Label]							[Comment] R button	
13	IF	(Button	=	1)	THEN	CALL	Right
	[Label]							[Comment] L button	
14	ELSE IF	(Button	=	2)	THEN	CALL	Left
	[Label]							[Comment] D button	
15	ELSE IF	(Button	=	4)	THEN	CALL	Backwa...
	[Label]							[Comment] U button	
16	ELSE IF	(Button	=	8)	THEN	CALL	Forwar...
	[Label]							[Comment] START + R button	
17	ELSE IF	(Button	=	17)	THEN	CALL	Openin...
	[Label]							[Comment] START + L button	
18	ELSE IF	(Button	=	18)	THEN	CALL	Closin...
	[Label]							[Comment] START + D button	
19	ELSE IF	(Button	=	20)	THEN	CALL	Put do...
	[Label]							[Comment] START + U button	
20	ELSE IF	(Button	=	24)	THEN	CALL	Liftin...
	[Label]							[Comment]	
21	JUMP		Loop						

When new data is received, it activates the necessary movement depending on the data. If there is no new data, there a sub-routine stops the movement of the probe robot. The reason for using a 0.125sec delay time when there is no received data is because if there is no time delay, the program will falsely determine that there is no new data before the actual data is completely received. As such, when new data arrives within the 0.125 time delay, the delay has no significance, but when no new data is received, the routine will stop the program.

The received data represents the button value of remote control. To understand the meaning of each number, refer to the following table.

Value	Meaning	Number
BIT_SW_RT	Right direction button	1 (0x01)
BIT_SW_LF	Left direction button	2 (0x02)
BIT_SW_DN	Downward direction button	4 (0x04)
BIT_SW_UP	Upward direction button	8 (0x08)
BIT_SW_START	Start button	16 (0x10)

Looking at the above table, you can tell what 1, 2, 4, 8 represent from the “If” statement from the behavior control program. Let’s see how you can decode 17, 18, 20, and 24.

When the Start button and Right button are pressed at the same time, the 0x11 key value is read. This is because BIT_SW_START and BIT_SW_RT went through an “OR” operation.

So, when combining the Start button and R button, the key value is 17 or 0x11. Likewise, 18 (0x12) combines the Start button and L button; 20 (0x14) combines the Start button and D button; and finally, 24 (0x18) combines the Start button and U button.

The C code using the CM-5 library is as follows

```
int main(void)
{
    PortInitialize();
    TimerInitialize(TIMER0, 128, 1); //Interrupt Enable
    SerialInitialize(SERIAL_PORT0,1,RX_INTERRUPT); //485
    SerialInitialize(SERIAL_PORT1,DEFAULT_BAUD_RATE,RX_INTERRUPT); //232
    ChargeInInterruptInitialize();
    sei();

    TxDString("\r\n CM-5 Example--- Remote Controlled Probing Robot\r\n");

    RXD_ZIGBEE;

    MotorInitialize();

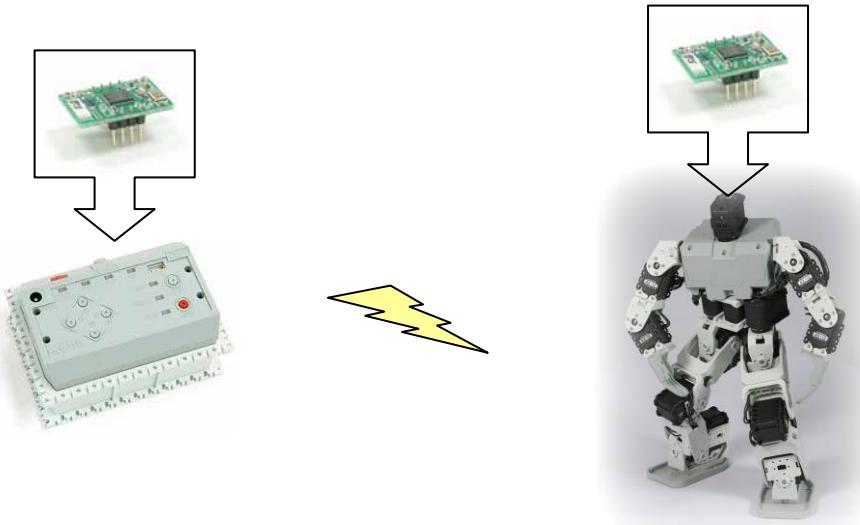
    while(1)
    {
        ZigbeePacketCheck();
    }
}
```

```
if( gbNewPacket )
{
    switch( gwZigbeeRxData )
    {
        case BIT_SW_RT           : Right();      break;
        case BIT_SW_LF           : Left ();      break;
        case BIT_SW_DN           : Backward();   break;
        case BIT_SW_UP           : Forward();    break;
        case BIT_SW_START|BIT_SW_RT : ProbeOpen(); break;
        case BIT_SW_START|BIT_SW_LF: ProbeClose(); break;
        case BIT_SW_START|BIT_SW_DN: ProbeDown(); break;
        case BIT_SW_START|BIT_SW_UP: ProbeUp();   break;
    }
}
else
{
    Stop();
}
MiliSec(50);
gbNewPacket = 0;
}

return 0;
}
```

3 – 2 . Controlling Humanoid Robot Via Remote Control

This chapter focuses on the Humanoid robot, an advanced level robot example from the Bioloid comprehensive kit. The Humanoid robot must have both of the Zig-100 units embedded and both Zig-100s must be set so that the communication mode is 1:1.



◆ Program download

- Remote control

Use “Bioloid SDK \ CM-5 \ bin \ example_remocon.hex”

- Humanoid

Use the “Applied Robots\ Advanced\ Humanoid\ DemoExample(Humanoid).mtn” of basic CD. and “Bioloid SDK \ CM-5 \ bin \ example_humanoid.bpg of expansion CD.

◆ Robot operation

< Remote control operation method >

Button	Function
UP	Forward
DOWN	Backward
LEFT	Left turn
RIGHT	Right turn
START	Get up

◎ Analyzing the remote control program

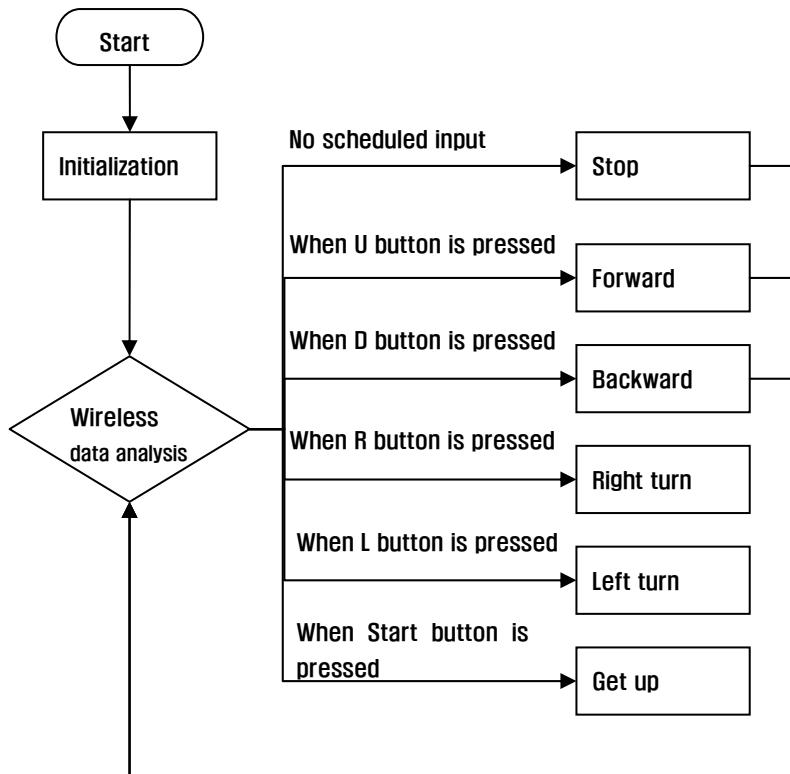
The C source code of this example can be found in “Bioloid SDK \CM-5 \app \example_remocon \example_remocon.c” on the expansion CD. The remote control program is the same as example 3-1, so refer to it if necessary.

```
int main(void)
{
    PortInitialize();
    TimerInitialize(TIMER0, 128, 1); //Interrupt Enable
    SerialInitialize(SERIAL_PORT0,1,RX_INTERRUPT); //485
    SerialInitialize(SERIAL_PORT1,DEFAULT_BAUD_RATE,RX_INTERRUPT); //232
    ChargeInInterruptInitialize();
    sei();
    TxDString("\r\n CM-5 Example--- Remote Controller\r\n");
    RXD_ZIGBEE;
    byte bKey;

    while(1) {
        bKey = GetButtonState();
        if( bKey != 0 ) {
            ZigbeePacketTransfer( bKey );
        }
    }
    return 0;
}
```

◎ Analyzing the Humanoid program

This example can be found in “Bioloid SDK \ CM-5 \ bin \ example_humanoid.bpg.” Let’s first look at the overall algorithm.



<Initialization>

For initialization part, the program executes as follows.

> Change each Dynamixel to “joint mode.”

	[Label] Initialize Motor						[Comment]			
17	IF	(1:8	=	0)	THEN	LOAD	1:8	{-} 1023
	[Label]						[Comment]			
18	IF	(2:8	=	0)	THEN	LOAD	2:8	{-} 1023
	[Label]						[Comment]			
19	IF	(3:8	=	0)	THEN	LOAD	3:8	{-} 1023
	[Label]						[Comment]			
20	IF	(4:8	=	0)	THEN	LOAD	4:8	{-} 1023
	[Label]						[Comment]			
21	IF	(5:8	=	0)	THEN	LOAD	5:8	{-} 1023
	[Label]						[Comment]			
22	IF	(6:8	=	0)	THEN	LOAD	6:8	{-} 1023
	[Label]						[Comment]			
23	IF	(7:8	=	0)	THEN	LOAD	7:8	{-} 1023
	[Label]						[Comment]			
24	IF	(8:8	=	0)	THEN	LOAD	8:8	{-} 1023
	[Label]						[Comment]			
25	IF	(9:8	=	0)	THEN	LOAD	9:8	{-} 1023
	[Label]						[Comment]			
26	IF	(10:8	=	0)	THEN	LOAD	10:8	{-} 1023
	[Label]						[Comment]			
27	IF	(11:8	=	0)	THEN	LOAD	11:8	{-} 1023
	[Label]						[Comment]			
28	IF	(12:8	=	0)	THEN	LOAD	12:8	{-} 1023
	[Label]						[Comment]			
29	IF	(13:8	=	0)	THEN	LOAD	13:8	{-} 1023

Refer to the Bioloid User’s Guide for the basic CD for changing “wheel mode” and “joint mode.”

> Initialization variable

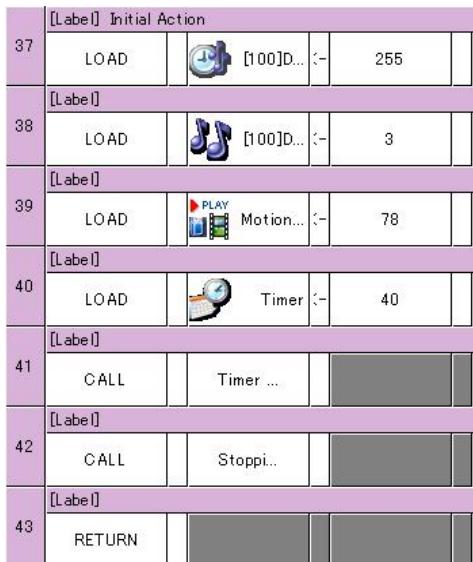
	[Label]					
3	LOAD	Walkin...	{-}	0		
	[Label]					
4	LOAD	Waitin...	{-}	4		

A variable used in this program is for walking direction and input wait time. For walking direction, the following shows what motion is currently executed.

Value	Meaning
0	Stand upright position
1	Walking forward
2	Walking backward
3	Walking toward right
4	Walking toward left

Input wait time is the waiting time for wireless data to arrive. When there is no input within this time, the robot automatically changes to the stand upright position. In this example, the wait time is set to 0.5 seconds.

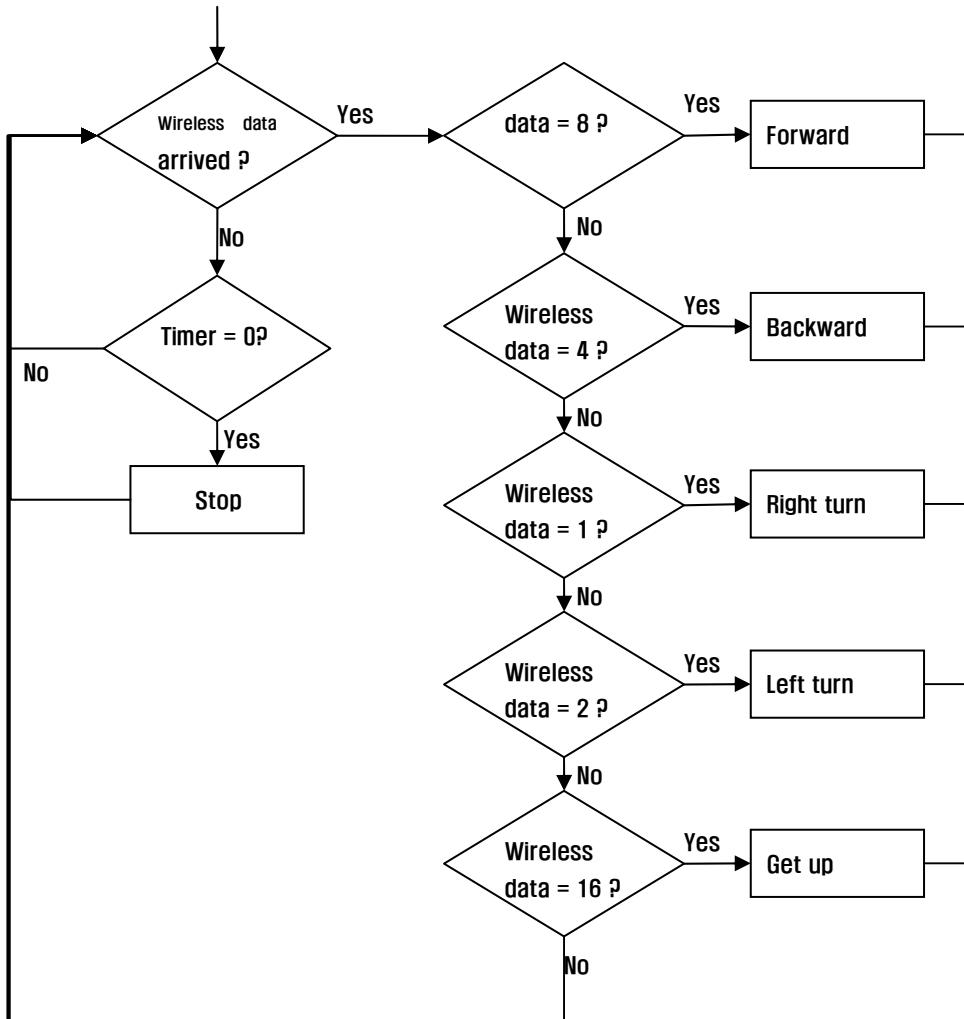
> Execute the robot's initial behavior



An initial behavior is a behavior the robot will take when the behavior control program starts. This example is created so that the #3 special melody is played. Then, after executing the #78 motion for approximately 5 seconds, the robot stands in an upright position. Since the #78 motion commands the robot to walk round and round, when this example is executed, the Humanoid robot walks around for 5 seconds making a melodic noise and then stops to wait for wireless data.

<Analyzing the wireless data>

The wireless data analysis routine allows appropriate action to be taken after receiving the wireless control data. The algorithm is as follows.



When the value of the wireless data is checked through a remote control program, it can be seen what buttons' state are on the CM-5. Received data represents the button value of the remote control. Accordingly, to understand the meaning of each number, you must know what value each button has. Refer to the following table.

Value	Meaning	Number
BIT_SW_RT	Right direction button	1 (0x01)
BIT_SW_LF	Left direction button	2 (0x02)
BIT_SW_DN	Downward direction button	4 (0x04)
BIT_SW_UP	Upward direction button	8 (0x08)
BIT_SW_START	Start button	16 (0x10)

Looking at the above table and the behavior control program, you can tell what 1, 2, 4, 8, and 16 represent from the “If” statement. However, the table does not address the combination of buttons?

When the Start button and Right button are pressed at the same time, the 0x11 key value is read. This is because BIT_SW_START and BIT_SW_RT went through an “OR” operation.

That is, 17 is 0x11, which is created by combining the Start button and R button. Likewise, 18 (0x12) combines the Start button and L button; 20 (0x14) combines the Start button and D button; and 24 (0x18) combines the Start button and U button.

The reason for checking the timeout value is so that Humanoid robot stops itself when it is not controlled via remote control.

The actual behavior control program source is as follows.

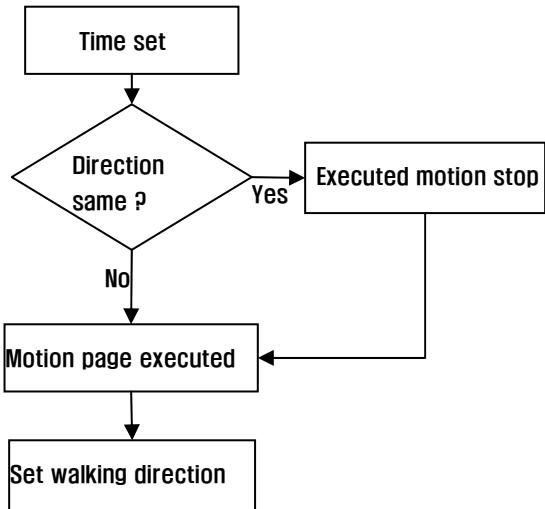
7	[Label] Loop	IF	( RX rem...)	=	1)	THEN	JUMP	Analyz...	[Comment]
8	[Label]	IF	( Timer)	=	0)	THEN	CALL	Stop	[Comment]
9	[Label]	JUMP	Loop						[Comment]
10	[Label] Analyzing Data	IF	( RX rem...)	=	8)	THEN	CALL	Forwar...	[Comment]
11	[Label]	ELSE IF	( RX rem...)	=	4)	THEN	CALL	Backwa...	[Comment]
12	[Label]	ELSE IF	( RX rem...)	=	1)	THEN	CALL	Right ...	[Comment]
13	[Label]	ELSE IF	( RX rem...)	=	2)	THEN	CALL	Left T...	[Comment]
14	[Label]	ELSE IF	( RX rem...)	=	16)	THEN	CALL	Get up	[Comment]
15	[Label]	JUMP	Loop						[Comment]

<Forward, backward, right turn, left turn>

Here, let's analyze the source code of the Humanoid robot by each situation. Forward and backward movement's only difference is the motion page number, but the underlying principle of the algorithm is the same.

Timer set is a routine to find out if there is no scheduled time input. When it is controlled by a remote control, due to this routine, the timer is not set to 0, and Stop is not executed.

The reason for checking for walking direction is because when a change in walking direction executes other motion pages, the current executed motion has to be stopped. If this routine is not applied, the robot will repeat walking and stopping, making the whole movement process unnatural. This routine allows the robot to move forward without the need to stop.



For executing the motion page, use applicable page numbers corresponding to each situation. Last, set the current walking direction

Name	Meaning
Init	Stop (Stand upright)
Forward walk	Forward
Backward walk	Backward
Turn right	Right turn
Turn left	Left turn
GetupBackView	Get up from lying face down position
GetupFrontView	Get up from lying face up position

(Forward)

	[Label] Forward						[Comment]		
49	LOAD		Timer	{-	Waitin...				
50	IF	(Walkin...		!=		1)	THEN
51	LOAD		Motion...	{-	60				
52	LOAD		Walkin...	{-	1				
53	RETURN								

(Backward)

	[Label] Backward						[Comment]		
55	LOAD		Timer	{-	Waitin...				
56	IF	(Walkin...		!=		2)	THEN
57	LOAD		Motion...	{-	69				
58	LOAD		Walkin...	{-	2				
59	RETURN								

(Right turn)

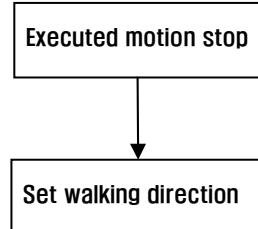
	[Label] Right Turn						[Comment]		
61	LOAD		Timer	{-	Waitin...				
62	IF	(Walkin...		!=		3)	THEN
63	LOAD		Motion...	{-	87				
64	LOAD		Walkin...	{-	3				
65	RETURN								

(Left turn)

	[Label] Left Turn							[Comment]	
67	LOAD		Timer	<-	Waitin...				
68	IF	(Walkin...		!=		4)	THEN
69	LOAD		Motion...	<-	96				
70	LOAD		Walkin...	<-	4				
71	RETURN								

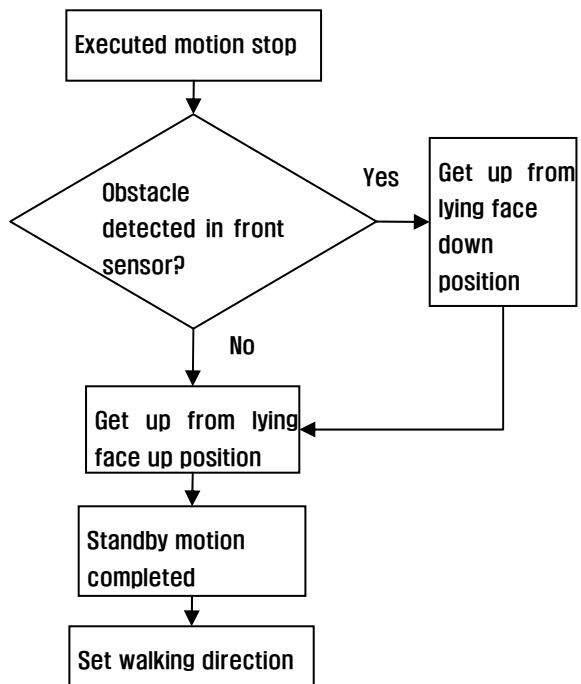
<Stop, get up>

For a stop motion, neither time set nor direction is checked. The reason is that since the timer is called and set to 0, it is not necessary to set the timer again. Additionally, the current motion has to be stopped regardless of the walking direction.



Unlike other behavior routines that execute a motion page to stand the robot upright, this routine has been provided with Humanoid motion data that controls forward, backward, left and right turn, and “the last page” that allocates stand upright position. When you execute the stop motion, the robot will automatically stand upright.

In the case of a “get up” motion, the process is far different from other motions. First, any currently executed motion must be stopped. The Humanoid robot has two types of “get up” motions. First is when they get up from lying in the face down position and other is from the face up position. With a single command to select one of the “get up” methods intelligently, the robot uses the front IR sensor of the AX-S1 attached to the head of the Humanoid robot. When obstacle is detected



by the IR sensor, the robot will get up from face down position; otherwise it means that the robot should get up from the face up position. To ensure that the Humanoid robot completely stands up, the program will execute the motion complete standby process. After the robot is fully up, it will be ready for a walking direction and will have the same value as the stopped state.

(Stop)

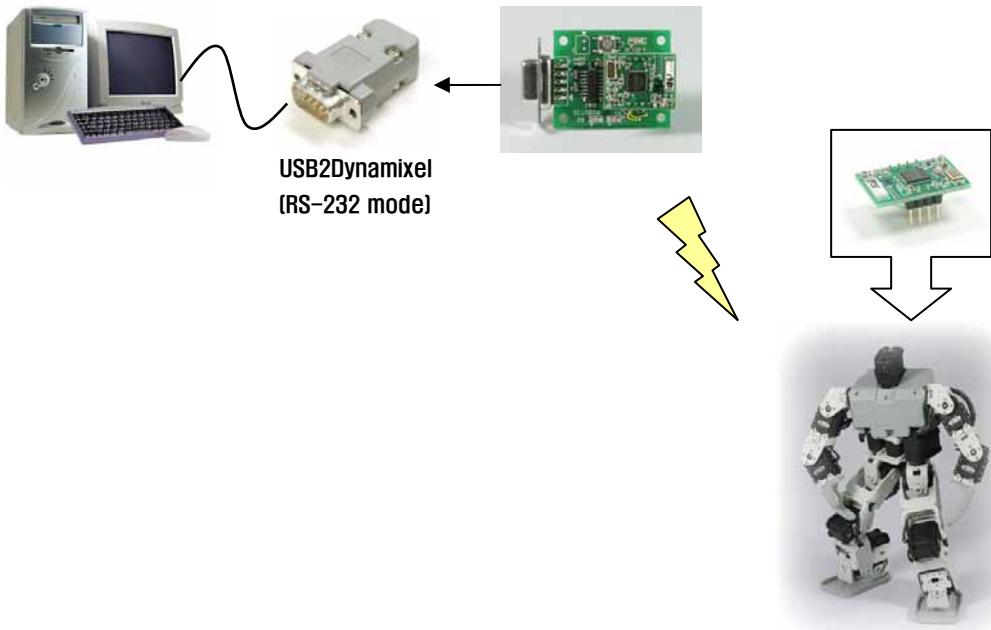
	[Label] Stop					
45	CALL		Stoppi...			
	[Label]					
46	LOAD		Walkin...	<-	0	
	[Label]					
47	RETURN					

(Get up)

	[Label] Get up						[Comment]
73	CALL		Stoppi...				
	[Label]						[Comment]
74	IF	([100]?..	>=	150)	THEN
	[Label]						[Comment]
75	ELSE		LOAD	Motion...	<-	13	
	[Label]						[Comment]
76	CALL		Motion...				
	[Label]						[Comment]
77	LOAD		Walkin...	<-	0		
	[Label]						[Comment]
78	LOAD		Clear ...	<-	RX rem...		
	[Label]						[Comment]
79	RETURN						

3 – 3. Controlling the Humanoid Robot Via PC

This chapter uses the expert level Humanoid robot from the Bioloid comprehensive kit as an example. For this example, the Humanoid robot must have a Zig-100 unit embedded and both Zig-100 units must be set to a 1:1 communication mode.



◆ Program download

- Remote control

Use “Bioloid SDK \ Windows \ bin \ PC controller.exe”

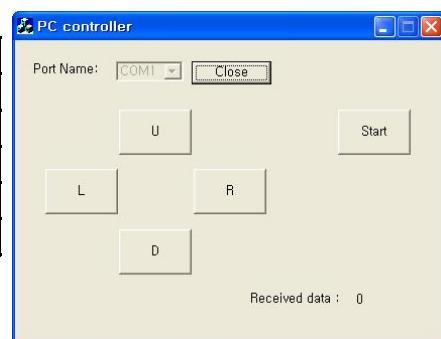
- Humanoid

Use the “Applied Robots\ Advanced\ Humanoid\ DemoExample(Humanoid).mtn” found in the basic CD and “Bioloid SDK \ CM-5 \ bin \ example_humanoid.bpg” in the expansion CD.

◆ Robot operation

< Remote control operation method >

Button	Function
U (or keyboard ↑)	Forward
D (or keyboard ↓)	Backward
L (or keyboard ←)	Left turn
R (or keyboard →)	Right turn
START (or keyboard Space)	Get up



◎ Analyzing the remote control program

First, using Visual C++ 6.0, open “Bioloid SDK \Windows \app \PC controller\PC controller.dsw” found in the expansion CD provided.

Background knowledge in MFC Programming is required for analyzing this example. Refer to reference books on MFC programming for more details.

Within the source of the PC controllerDlg.h file, let's analyze the CPCcontrollerDlg class. Created sources are indicated by the bolded words.

```
////////////////////////////////////////////////////////////////  
// CPCcontrollerDlg dialog  
  
class CPCcontrollerDlg : public CDialog  
{  
// Construction  
public:  
    CPCcontrollerDlg(CWnd* pParent = NULL); // standard constructor  
  
    BOOL m_bConnected; // check zigbee library open  
    BYTE m_ButtonState;  
  
// Dialog Data  
    //{{AFX_DATA(CPCcontrollerDlg)  
    enum { IDD = IDD_PCCONTROLLER_DIALOG } ;  
    CStatic m_Static_RcvData;  
    CComboBox m_Combo_Port;  
    CButton m_Button_U;  
    CButton m_Button_S;  
    CButton m_Button_R;  
    CButton m_Button_Port;  
    CButton m_Button_L;  
    CButton m_Button_D;  
    //}}AFX_DATA
```

Here, a global variable is declared. **m_bConnected** is a variable that checks to see if Zig2Serial is currently open. Additionally, **m_ButtonState** is a variable, just like in the CM-5, that saves the current button state.

This time, let's analyze the PC controllerDlg.cpp file. Most of the program's functions are included in this file. The initialization routine is as follows. Bolded words are created sources.

```
BOOL CPCcontrollerDlg::OnInitDialog()  
{
```

```
CDialog::OnInitDialog();

// Set the icon for this dialog. The framework does this automatically
// when the application's main window is not a dialog
SetIcon(m_hIcon, TRUE);           // Set big icon
SetIcon(m_hIcon, FALSE);          // Set small icon

// Initialization program
m_Combo_Port.SetCurSel( 0 );
m_Button_U.EnableWindow( FALSE );
m_Button_R.EnableWindow( FALSE );
m_Button_L.EnableWindow( FALSE );
m_Button_D.EnableWindow( FALSE );
m_Button_S.EnableWindow( FALSE );

return TRUE; // return TRUE unless you set the focus to a control
}
```

In the initialization section, the program first creates a list for selecting the COM port and makes the buttons inactive.

When the open button is pressed on the program screen, Zig2Serial is initialized, and when the close button is pressed, Zig2Serial will be closed. This process is constructed as follows.

```
void CPCcontrollerDlg::OnButtonPort()
{
    CString strPortName;
    if( m_bConnected == TRUE ) // If already open zigbee libarary
    {
        KillTimer( 0 );
        zigbee_close();
        m_bConnected = FALSE;
        m_Button_Port.SetWindowText( "Open" );
        m_Combo_Port.EnableWindow( TRUE );
        m_Button_U.EnableWindow( FALSE );
        m_Button_R.EnableWindow( FALSE );
        m_Button_L.EnableWindow( FALSE );
        m_Button_D.EnableWindow( FALSE );
        m_Button_S.EnableWindow( FALSE );
    }
    else // If close zigbee libarary
    {
        m_Combo_Port.GetLBText( m_Combo_Port.GetCurSel(), strPortName ); // Get port
        name
        if( zigbee_open( strPortName.GetBuffer(strPortName.GetLength()) ,
GetSafeHwnd(), WM_ZIGBEE_RECEIVE ) == true )
        {
            m_bConnected = TRUE;
            m_ButtonState = 0;
            SetTimer( 0, 10, NULL ); // Send zigbee data per 10ms
            m_Button_Port.SetWindowText( "Close" );
            m_Combo_Port.EnableWindow( FALSE );
            m_Button_U.EnableWindow( TRUE );
            m_Button_R.EnableWindow( TRUE );
            m_Button_L.EnableWindow( TRUE );
            m_Button_D.EnableWindow( TRUE );
            m_Button_S.EnableWindow( TRUE );
        }
        else
            AfxMessageBox( "Fail to open zigbee!" );
    }
}
```

OnButtonPort() function's main purpose is to initialize the Zig2Serial and prepare for communication, or to terminate the Zig2Serial when it has already been initialized.

When the Zig2Serial is initialized, the timer is activated and the program will check for the button state and then calculate and transfer the pressed state values.

```
void CPCcontrollerDlg::OnTimer(UINT nIDEvent)
{
    static count = 0;

    if( m_bConnected == TRUE )
    {
        if( m_Button_U.GetState() & 0x0004 )
            m_ButtonState = m_ButtonState | BIT_SW_UP;
        else
            m_ButtonState = m_ButtonState & (~BIT_SW_UP);

        if( m_Button_D.GetState() & 0x0004 )
            m_ButtonState = m_ButtonState | BIT_SW_DN;
        else
            m_ButtonState = m_ButtonState & (~BIT_SW_DN);

        if( m_Button_L.GetState() & 0x0004 )
            m_ButtonState = m_ButtonState | BIT_SW_LF;
        else
            m_ButtonState = m_ButtonState & (~BIT_SW_LF);

        if( m_Button_R.GetState() & 0x0004 )
            m_ButtonState = m_ButtonState | BIT_SW_RT;
        else
            m_ButtonState = m_ButtonState & (~BIT_SW_RT);

        if( m_Button_S.GetState() & 0x0004 )
            m_ButtonState = m_ButtonState | BIT_SW_START;
        else
            m_ButtonState = m_ButtonState & (~BIT_SW_START);

        if( m_ButtonState != 0 )
        {
            zigbee_send( (WORD)m_ButtonState );
            count++;
            TRACE( "%d\n", count );
        }
    }

    CDialog::OnTimer(nIDEvent);
}
```

Transferred button state value, m_ButtonState, is created the same way as the CM-5 button's state value. In the following, you can see that the declared CM-5 library is the same. For more details on button combinations, refer to the Chapter 3-1.

```
// CM-5 button state
#define BIT_SW_RT      0x01
#define BIT_SW_LF      0x02
#define BIT_SW_DN      0x04
#define BIT_SW_UP      0x08
#define BIT_SW_START    0x10
```

In addition to the above, this program also outputs received data on the screen. The following is the source code for that process. For more details on how to receive data from the Zigbee library, refer to Chapter 2-2.

```
// User message for zigbee communication
#define WM_ZIGBEE_RECEIVE      WM_USER+1

// Window message procedure
LRESULT CPCcontrollerDlg::WindowProc(UINT message, WPARAM wParam, LPARAM lParam)
{
    CString strText;

    if( message == WM_ZIGBEE_RECEIVE ) // If received zigbee data
    {
        strText.Format( "%d", (WORD)wParam );
        m_Static_RcvData.SetWindowText( strText );
    }

    return CDialog::WindowProc(message, wParam, lParam);
}
```

This program can uses a mouse or keyboard (arrows and spacebar) to press buttons and control the program. The following source shows the keyboard message processing.

```
BOOL CPCcontrollerDlg::PreTranslateMessage(MSG* pMsg)
{
    // TODO: Add your specialized code here and/or call the base class
    if( m_bConnected == TRUE )
    {
        if( pMsg->message == WM_KEYDOWN )
        {
            switch((int)pMsg->wParam)
            {
                case VK_UP:
```

```
    m_Button_U.SetState( TRUE );
    return TRUE;

    case VK_DOWN:
        m_Button_D.SetState( TRUE );
        return TRUE;

    case VK_RIGHT:
        m_Button_R.SetState( TRUE );
        return TRUE;

    case VK_LEFT:
        m_Button_L.SetState( TRUE );
        return TRUE;

    case VK_SPACE:
        m_Button_S.SetState( TRUE );
        return TRUE;
    }

}

else if( pMsg->message == WM_KEYUP )
{
    switch((int)pMsg->wParam)
    {
        case VK_UP:
            m_Button_U.SetState( FALSE );
            return TRUE;

        case VK_DOWN:
            m_Button_D.SetState( FALSE );
            return TRUE;

        case VK_RIGHT:
            m_Button_R.SetState( FALSE );
            return TRUE;

        case VK_LEFT:
            m_Button_L.SetState( FALSE );
            return TRUE;

        case VK_SPACE:
            m_Button_S.SetState( FALSE );
            return TRUE;
    }
}

return CDIALOG::PreTranslateMessage(pMsg);
}
```

Next is the closing of the program.

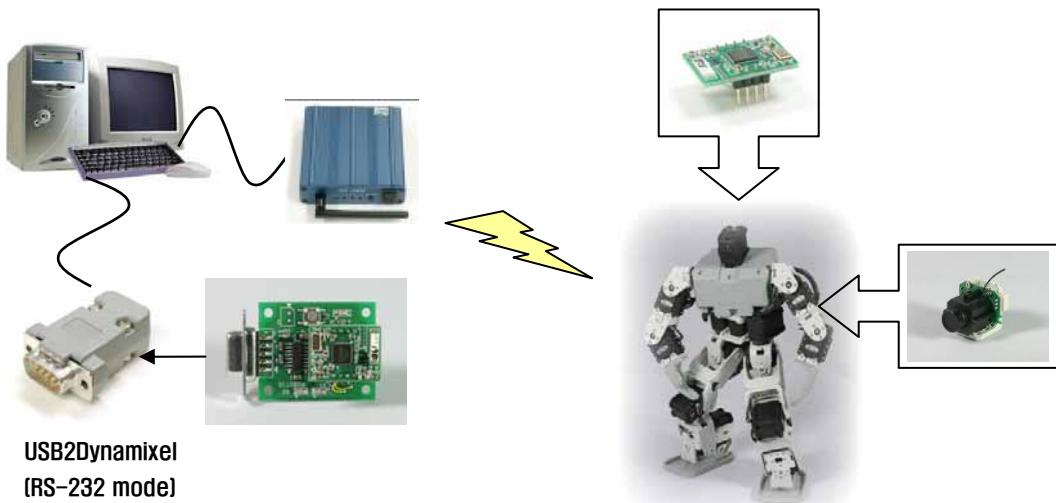
```
void CPCcontrollerDlg::OnClose()
{
    // TODO: Add your message handler code here and/or call default
    zigbee_close();
    CDialog::OnClose();
}
```

○ Analyzing the Humanoid program

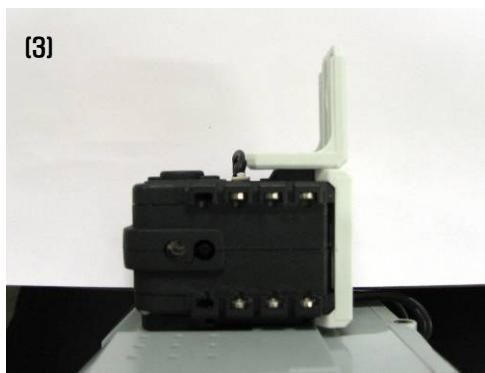
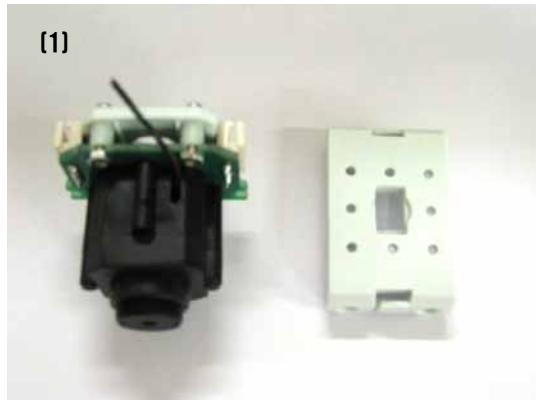
This program is the same as the previous in Chapter 3-2. Therefore, for detailed information, refer back to Chapter 3-2.

3 – 4 . Image Communication and Wireless Control of the Humanoid Robot

This chapter uses the Humanoid robot, an expert level robot example from the Bioloid comprehensive kit. For this example, the Humanoid robot must have a Zig-100 unit embedded and a wireless image transmitter. In addition, both Zig-100 units must be set to the 1:1 communication mode.



◆ How to install a camera



◆ Program download

- Remote control

Use “Bioloid SDK \ Windows \ bin \ PC controller2.exe”

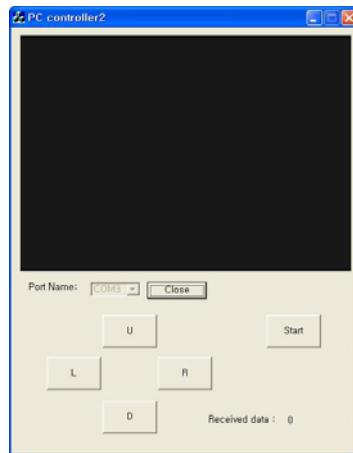
- Humanoid

Use the “Applied Robots\ Advanced\ Humanoid\ DemoExample(Humanoid).mtn” of basic CD.
and “Bioloid SDK \ CM-5 \ bin \ example_humanoid.bpg of expansion CD.

◆ Robot operation

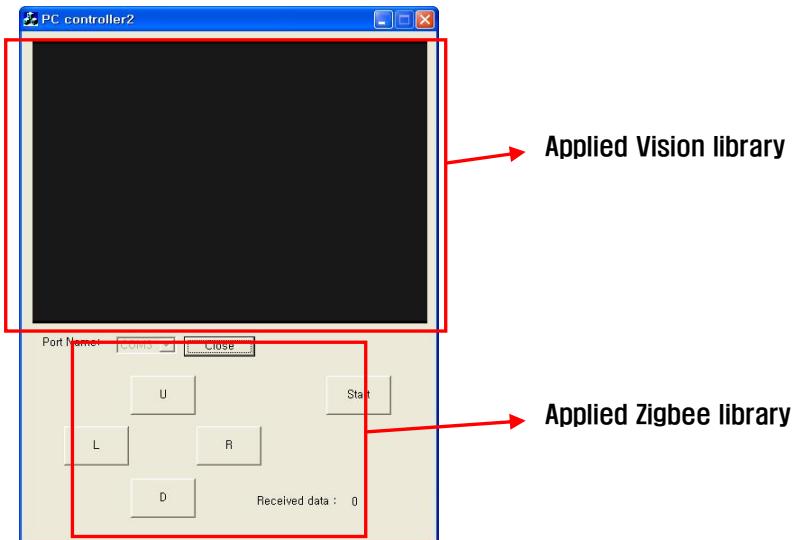
< Remote control operation method >

Button	Function
U (or keyboard ↑)	Forward
D (or keyboard ↓)	Backward
L (or keyboard ←)	Left turn
R (or keyboard →)	Right turn
START (or keyboard Space)	Get up



◎ Analyzing the remote control program

The program here combines the robot wireless control function that uses the Zigbee library and the image processing function that uses the vision library. However, since the robot wireless control function is the same as the remote control program source previously explained in Chapter 3-3, this chapter will only discuss the image processing function.



First, using Visual C++ 6.0, open the “Bioloid SDK \Windows \app \PC controller2\PC controller2.dsw” file from the expansion CD provided.

Background knowledge in MFC Programming is required for analyzing this example. Refer to reference books on MFC programming for more details.

From the PC controller2Dlg.cpp source, let's look at the program initiation routine.

```
BOOL CPCcontroller2Dlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);          // Set small icon

    // Initialization program
    VISION_PARAM vision_param;

    vision_param.width = 320;
```

```
vision_param.height = 240;
vision_param.pProcessFunc = ImageProcess;

if( vision_open( &vision_param ) == true )
    vision_start();
else
    AfxMessageBox( "Fail to open Camera!" );

m_Combo_Port.SetCurSel( 0 );
m_Button_U.EnableWindow( FALSE );
m_Button_R.EnableWindow( FALSE );
m_Button_L.EnableWindow( FALSE );
m_Button_D.EnableWindow( FALSE );
m_Button_S.EnableWindow( FALSE );

return TRUE; // return TRUE unless you set the focus to a control
}
```

Looking at the vision library initialization section, it shows that the image captured had a 320 X 240 resolution and registered imageProcess function as an image processing function.

The actual image processing is handled by the ImageProcess function whose source is shown below.

```
void ImageProcess( unsigned char* pImage, DWORD dwSize )
{
    CDC *pDC;
    CBitmap *pBitmap;
    CRect rtArea;

    // Initialize
    pDC = pThis->m_Static_Video.GetDC();

    // Converting
    pBitmap = CBitmap::FromHandle( vision_RGB24toDDB( pDC->m_hDC, pImage ) );

    // Drawing

    // Update screen
    pThis->m_Static_Video.GetClientRect( rtArea );
    vision_DrawDDB( pDC->m_hDC, 0, 0, rtArea.Width(), rtArea.Height(),
(HBITMAP)*pBitmap );
    pThis->InvalidateRect( NULL );
}
```

```
// Release  
pBitmap->DeleteObject();  
pDC->DeleteDC();  
}
```

The **ImageProcess** function draws the appropriate sized image on the screen after receiving captured images from the vision library.

Upon closing the program, the program closes both the Zigbee library and Vision library.

```
void CPCcontroller2Dlg::OnClose()  
{  
    // TODO: Add your message handler code here and/or call default  
    zigbee_close();  
    vision_close();  
    CDialog::OnClose();  
}
```

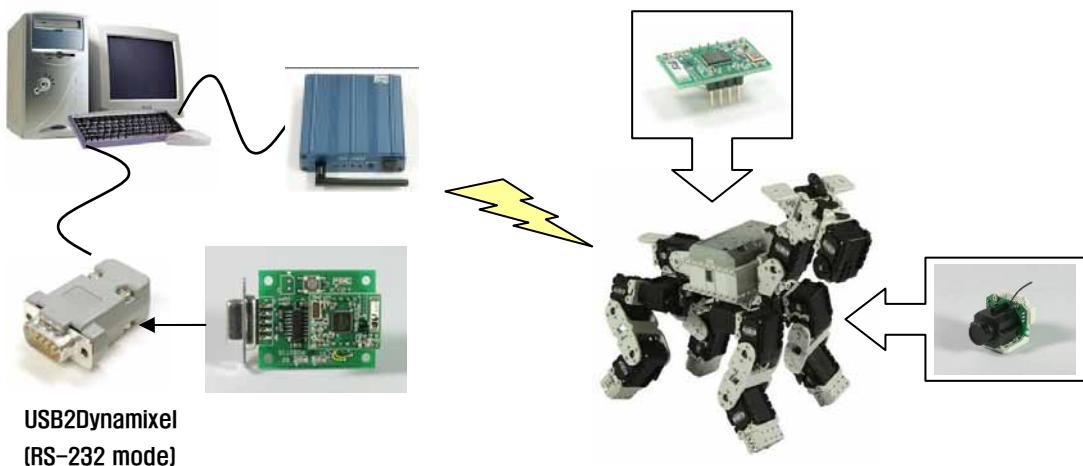
The Communication routine analysis using the Zigbee library is the same as Chapter 3-3. Refer to the chapter for more information.

○ Analyzing Humanoid program

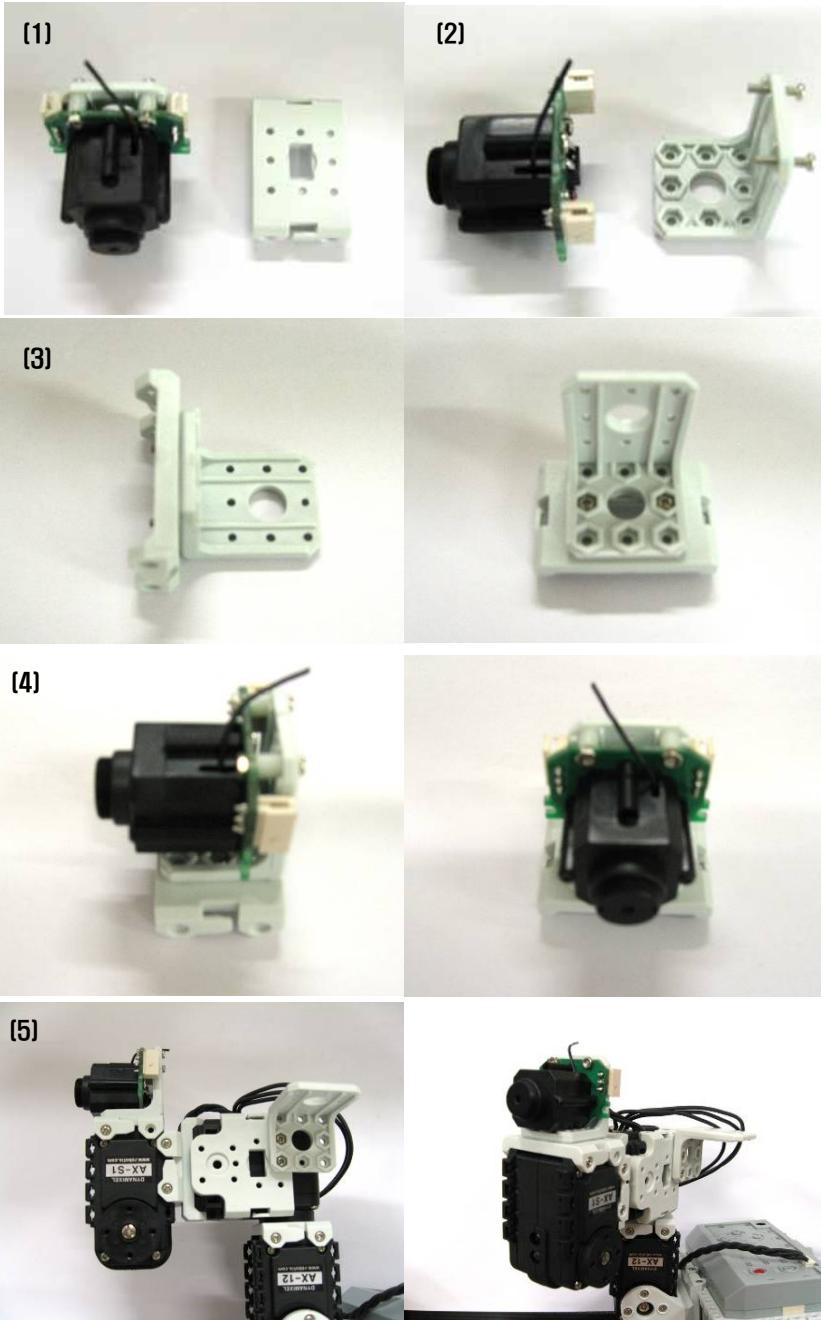
This program is the same as the program in Chapter 3-2. Therefore, for detailed information, refer back to Chapter 3-2.

3 – 5. Image Recognition and Movement Detection for the Puppy Robot

This chapter uses the puppy robot, an expert level robot example from the Bioloid comprehensive kit. For this example, the puppy robot must have a Zig-100 unit embedded and a wireless image transmitter. In addition, both Zig-100 units must be set to the 1:1 communication mode.



◆ How to install a camera



◆ Program download

- Remote control

Use “Bioloid SDK \ Windows \ bin \ Motion Detector.exe”

- Puppy robot

Use “Applied Robots \ Advanced \ Puppy \ DemoExample(Puppy).mtn” of basic CD and

“Bioloid SDK \ Windows \ bin \ example_puppy.bpg” of expansion CD.

◆ Robot operation

- Execute a behavior control program befitting the CM-5 of the puppy robot to play mode.

- Execute Motion Detector.exe.

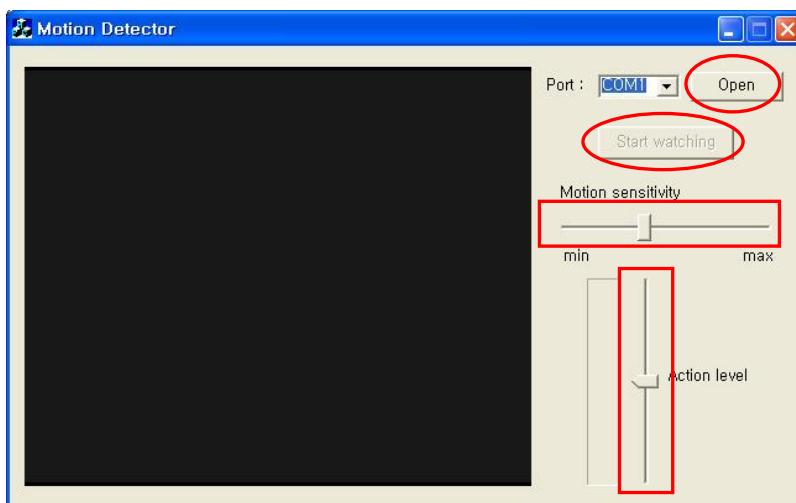
- Configure communication befitting where Zig2Serial B/D is connected to the COM port of PC.

- Start the remote control by pressing the “Start watching” button.

- Create an optimal environment by controlling the motion sensitivity and the action level.

- When the puppy robot’s program starts and the puppy is in a sitting position, when the movement is detected by a camera, it will get up and start barking.

If no movement is detected for specific time, it will sit down.



◎ Analyzing the Motion Detector program

This program uses the vision library to check the images and when the movement is detected, the program sends a control command to a robot by using the Zigbee library. Before proceeding with the analysis, you are recommended to refer books on image processing and MFC programming.

First, using Visual C++ 6.0, open the “Bioloid SDK \Windows \app \Motion Detector \Motion Detector.dsw” file from the expansion CD provided. Most of the source code is written in the MotionDetectorDlg.h and MotionDetectorDlg.cpp file.

The program initiation routine is as follows.

```
BOOL CMotionDetectorDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);         // Set small icon

    // Initialize vision
    VISION_PARAM vision_param;

    vision_param.width = 320;
    vision_param.height = 240;
    vision_param.pProcessFunc = ImageProcess;

    if( vision_open( &vision_param ) == true )
        vision_start();
    else
        AfxMessageBox( "Fail to open Camera!" );

    // Initialize controls
    m_Combo_Port.SetCurSel( 0 );
    m_Button_Watch.EnableWindow( FALSE );
    m_Progress_Motion.SetRange( 0, 40 );
    m_Progress_Motion.SetPos( 0 );
    m_Slider_Sensitivity.SetRange( 1, 50 );
    m_Slider_Sensitivity.SetPos( 20 );
    m_Slider_Action.SetRange( 0, 40 );
    m_Slider_Action.SetPos( 20 );

    // Create Image
```

```
m_pGrayImage = new BYTE[vision_param.width * vision_param.height];
m_pBeforeImage = new BYTE[vision_param.width * vision_param.height];
m_pDiffImage = new BYTE[vision_param.width * vision_param.height];
m_pBinaryImage = new BYTE[vision_param.width * vision_param.height];

return TRUE; // return TRUE unless you set the focus to a control
}
```

Here, the Vision Library captures images with a resolution of 320 X 240 and registers the ImageProcess function as an image processing function. Also, memory space will be allocated for image data utilized in control initialization and image processing.

The main algorithm for detecting movement and controlling the robot is embedded in the ImageProcess function.

```
void ImageProcess( unsigned char* pImage, DWORD dwSize )
{
    CDC *pDC;
    CBitmap *pBitmap;
    CRect rtArea;
    VISION_STATE vision_state;
    long SumPixel;
    float Percent;
    int MotionSensitivity;
    int ThresholdAction;

    // Initialize
    pDC = pHs->m_Static_Video.GetDC();
    vision_state = vision_get_state();

    // Processing
    if( pHs->m_bWatching == TRUE )
    {
        RGB24toGray( pHs->m_pGrayImage, pImage, vision_state.width,
vision_state.height );
        DifferenceImage( pHs->m_pDiffImage, pHs->m_pBeforeImage, pHs-
>m_pGrayImage, vision_state.width, vision_state.height );
        CopyImage( pHs->m_pBeforeImage, pHs->m_pGrayImage, vision_state.width,
vision_state.height );
        MotionSensitivity = 50 - pHs->m_Slider_Sensitivity.GetPos();
        BinaryImage( pHs->m_pBinaryImage, pHs->m_pDiffImage,
(BYTE)MotionSensitivity, vision_state.width, vision_state.height );
    }

    // Converting
```

```
if( pThis->m_bWatching == TRUE )
    pBitmap = CBitmap::FromHandle( vision_GraytoDDB( pDC->m_hDC, pThis-
>m_pBinaryImage ) );
else
    pBitmap = CBitmap::FromHandle( vision_RGB24toDDB( pDC->m_hDC, pImage ) );

// Drawing

// Command Robot
if( pThis->m_bWatching == TRUE )
{
    SumPixel = GetSumPixel( pThis->m_pBinaryImage, 255, vision_state.width,
vision_state.height );
    Percent = (float)((float)SumPixel / (float)(vision_state.width *
vision_state.height)) * 100.0f;
    pThis->m_Progress_Motion.SetPos( (int)Percent );

    ThresholdAction = 40 - pThis->m_Slider_Action.GetPos();
    if( (int)Percent >= ThresholdAction )
        zigbee_send( 1 );
}

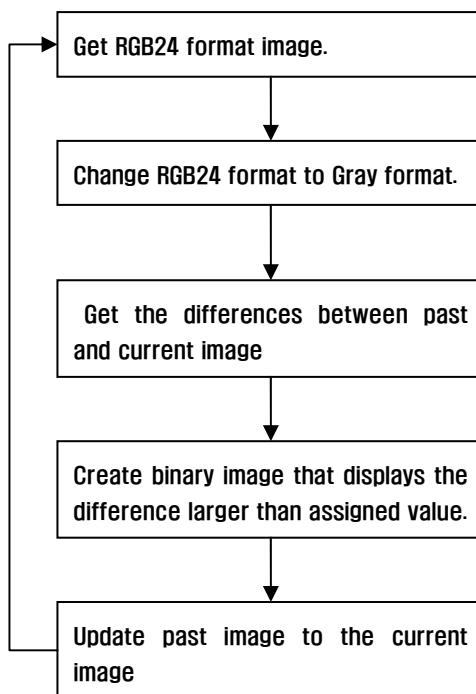
// Update screen
pThis->m_Static_Video.GetClientRect( rtArea );
vision_DrawDDB( pDC->m_hDC, 0, 0, rtArea.Width(), rtArea.Height(),
(HBITMAP)*pBitmap );
pThis->InvalidateRect( NULL );

// Release
pBitmap->DeleteObject();
pDC->DeleteDC();
}
```

The image processing functions used for the movement detection and its capacities are shown below.

Function	Capacity
RGB24toGray	Change RGB24 format image to Gray format.
CopyImage	Copy the Gray format image.
DifferenceImage	Create image from the differences in two Gray format image.
BinaryImage	Change the image created in Gray level (256 color) to binary level (2 color).
GetSumPixel	Get large pixel number than the assigned Gray format value.

The image processing algorithm for movement detection is as follows.



< Get RGB24 format image >

The vision library calls the image processing function, which was registered on initialization, on scheduled time interval. At this time, it transfers the image captured by the camera. The format of image is RGB24.

< Change RGB24 format to Gray format >

RGB24 format represents a color image, whereas Gray format represents a black and white image. Currently most of the image processing algorithms use Gray format image data. This

function is embedded in the **RGB24toGray** function. In addition, this function receives inputs of image and the resolution of the RGB24 format, and then converts the image to Gray format.

```
void RGB24toGray( unsigned char* pGray, unsigned char* pRGB24, int width, int height )
{
    int x, y, index;
    unsigned char r, g, b, gray;

    // Convert Image (RGB -> Gray)
    for( y=0; y<height; y++ )
    {
        for( x=0; x<width; x++ )
        {
            index = y*width + x;

            b = pRGB24[3*index];
            g = pRGB24[3*index+1];
            r = pRGB24[3*index+2];

            gray = (unsigned char)((r + g + b) / 3);
            pGray[index] = gray;
        }
    }
}
```

< Find the differences between the past and the current image >

The most basic way to find movement from images is by checking the differences in pixel values. Simply put, when there is no change in the pixel value of particular location, it means that there is no movement, and conversely, if there is a change in the pixel value, it indicates that there is a change in the environment.

Image data1



Image data 2



Image data result



```
void DifferenceImage( unsigned char* pDiff, unsigned char* pImage1, unsigned char*
pImage2, int width, int height )
{
    int x, y, index;
    int diff;

    // Make difference Image
    for( y=0; y<height; y++ )
    {
        for( x=0; x<width; x++ )
        {
            index = y*width + x;

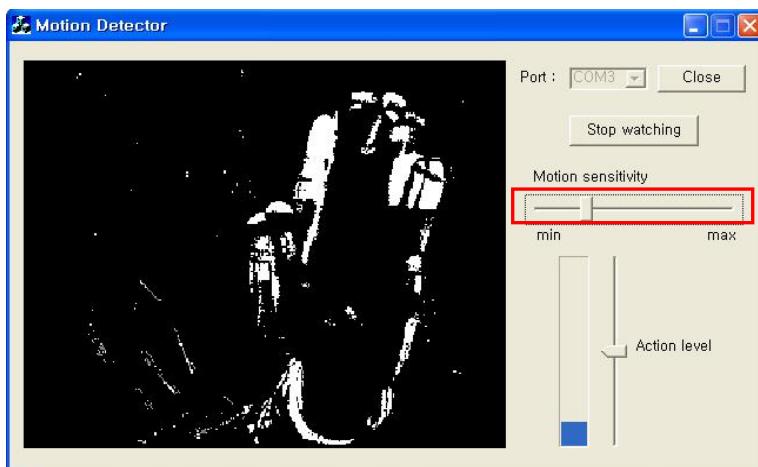
            diff = (int)(pImage1[index] - pImage2[index]);

            pDiff[index] = (unsigned char)abs(diff);
        }
    }
}
```

In this function, a black pixel represents a non-movement pixel, and any other color pixel represents a changed pixel.

< Create a binary image that displays pixels whose values change is larger than the assigned value >

The image result, as a result of the `DifferenceImage` function, detects even the smallest pixel movement. Although change in pixel values is a result of actual movement, some changes are caused by camera noise. Therefore, the program must be configured so that insignificant changes are ignored. Additionally, data 0 and 1 data are needed so that it can represent actual movement and non-movement. This kind of function is included in `BinaryImage` function. Insert the Gray value and set the standard value. If the standard value is small, make it 0, and if large, make it 1. In actuality, it is not structured as 0 and 1, but rather, it transfers the image result of Gray format, which is composed of 0 and 255. The standard value used here is the program's motion sensitivity.

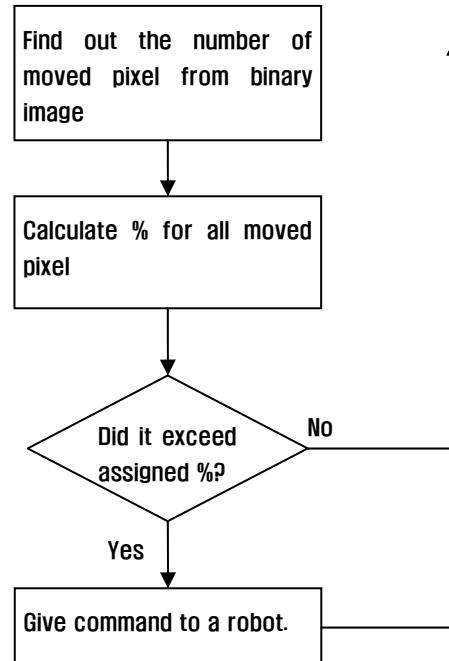


```
void BinaryImage( unsigned char* pBinary, unsigned char* pGray, int Threshold, int width, int height )
{
    int x, y, index;
    // Convert Image (Gray -> Binary)
    for( y=0; y<height; y++ )
    {
        for( x=0; x<width; x++ )
        {
            index = y*width + x;
            if( pGray[index] > (unsigned char)Threshold )
                pBinary[index] = 255;      // 1 (Binary)
            else
                pBinary[index] = 0;      // 0 (Binary)
        }
    }
}
```

< Update the past image to the current image>

After the image result of movement detection is received, for the next calculation, you have to change past image data to current image data. This is because the current image will be the past image in the next process.

The Image copy function is embedded in the CopyImage function.



```

void CopyImage( unsigned char* pDest, unsigned char* pSrc, int width, int height )
{
    int x, y, index;

    // Copy Image
    for( y=0; y<height; y++ )
    {
        for( x=0; x<width; x++ )
        {
            index = y*width + x;

            pDest[index] = pSrc[index];
        }
    }
}
  
```

Although the image marking movement is received, it cannot directly give command to the robot. As such, by using the results acquired from the image processing, the program can give commands. The following is its algorithm.

< Find the number of moved pixels from a binary image >

This function is embedded in the GetPixelSum function. This function returns the total number of pixels larger than the standard value when it receives the original image data and its size and the value that will be compared.

```
long GetSumPixel( unsigned char* pImage, int Threshold, int width, int height )  
{  
    int x, y, index;  
    long sum = 0;  
  
    // Count pixel of over threshold  
    for( y=0; y<height; ; y++ )  
    {  
        for( x=0; x<width; x++ )  
        {  
            index = y*width + x;  
  
            if( pImage[index] >= (unsigned char)Threshold )  
                sum++;  
        }  
    }  
  
    return sum;  
}
```

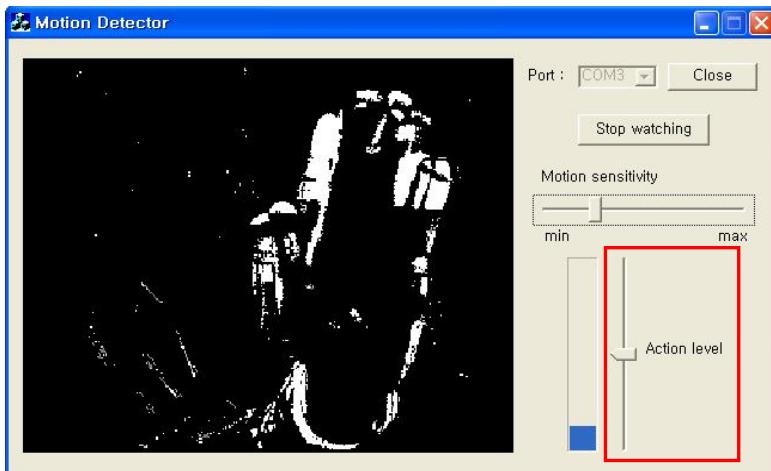
< Calculate % for all moved pixels >

The total number of pixels is the width X height of image data. For example, if the resolution width is 320 and height is 240, then it is 76800 (320 x 240 = 76800).



< When assigned % exceeds, give a command to a robot >

Assigned values here indicate the Action level value. If the number of moved pixels is larger than the value of the action level, the program will, through Zigbee library, send a control command to the robot. When the Action level is high, there must be a lot of movement in order for the robot to move, and conversely, when it is low, there need only be a small amount of movement in order for the robot to move.



```
// Command Robot
if( pThis->m_bWatching == TRUE )
{
    SumPixel = GetSumPixel( pThis->m_pBinaryImage, 255, vision_state.width,
vision_state.height );
    Percent = (float)((float)SumPixel / (float)(vision_state.width *
vision_state.height)) * 100.0f;
    pThis->m_Progress_Motion.SetPos( (int)Percent );

    ThresholdAction = 40 - pThis->m_Slider_Action.GetPos();
    if( (int)Percent >= ThresholdAction )
        zigbee_send( 1 );
}
```

The value for the robot controlling command is 1. That is, when the robot receives the value of 1 through Zigbee wireless communication, it will move.

The finalizing routine is as follows.

```
void CMotionDetectorDlg::OnClose()
{
    // TODO: Add your message handler code here and/or call default
    zigbee_close();
    vision_close();

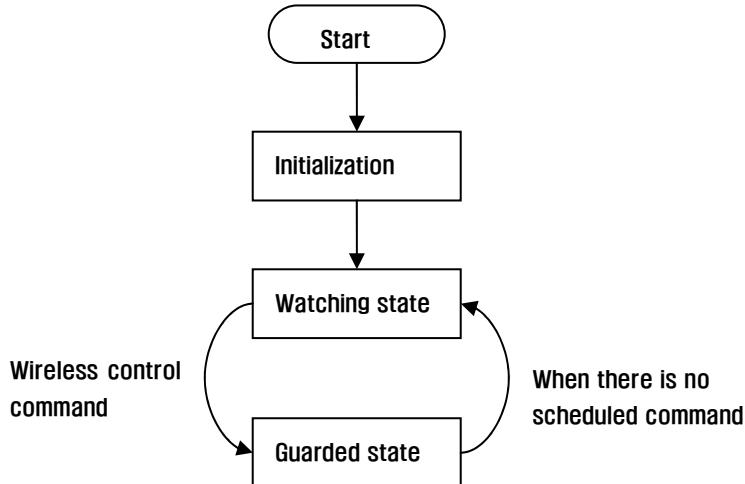
    // Destroy Image
    delete m_pGrayImage;
    delete m_pBeforeImage;
    delete m_pDiffImage;
    delete m_pBinaryImage;

    CDialog::OnClose();
}
```

Here, the program allocates memory for terminating the used library and image processing and again, it removes used pointers.

◎ Analyzing the puppy program

This example can be found in the “Bioloid SDK \ Windows \ bin \ example_puppy.bpg” . Before proceeding with program analysis, let’s look at the overall algorithm.



In “Applied Robots \ Advanced \ Puppy \ Motion list[DemoExample[Puppy]].txt” of the basic CD, you can check the motion page number used in the behavior control program.

Page No.	Name	Meaning
1	Ready	Stand position
6	Sit down	Sit down
7	Stand up	Stand up
16	Bark	Bark

< Initialization>

In the initialization section, perform the following work.

> Change each Dynamixel motor to joint mode

	[Label:] Initialize Motor					[Comment]
29	IF	(1:8	=	0) THEN
30	LOAD		1:8	<-	1023	
31	IF	(4:8	=	0) THEN
32	LOAD		4:8	<-	1023	
33	IF	(5:8	=	0) THEN
34	LOAD		5:8	<-	1023	
35	IF	(6:8	=	0) THEN
36	LOAD		6:8	<-	1023	
37	IF	(7:8	=	0) THEN
38	LOAD		7:8	<-	1023	
39	IF	(8:8	=	0) THEN
40	LOAD		8:8	<-	1023	
41	IF	(9:8	=	0) THEN
	LOAD		9:8	<-	1023	
	IF	(10:8	=	0) THEN
	LOAD		10:8	<-	1023	
	IF	(11:8	=	0) THEN
	LOAD		11:8	<-	1023	
	IF	(12:8	=	0) THEN
	LOAD		12:8	<-	1023	
	IF	(13:8	=	0) THEN
	LOAD		13:8	<-	1023	
	IF	(14:8	=	0) THEN
	LOAD		14:8	<-	1023	
	IF	(15:8	=	0) THEN
	LOAD		15:8	<-	1023	

Refer to the Bioloid user's guide from the basic CD for switching between "wheel mode" and "joint mode."

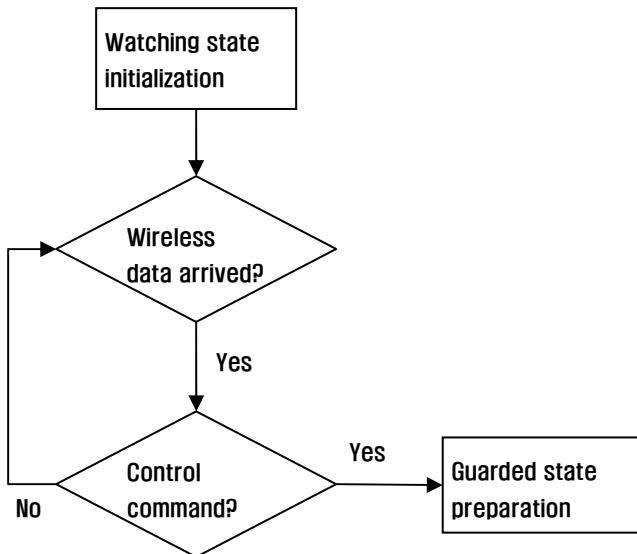
> Initial behavior

The initial behavior represents the behavior the robot takes when the program first starts.

	[Label:] Initial Action				
46	LOAD		[100]D...	<-	255
47	LOAD		[100]D...	<-	0
48	LOAD		PLAY Motion...	<-	1
49	CALL	Motion...			
50	RETURN				

< Watching state >

The algorithm is as follows.



< Watching state initialization >

Here, the initialization content is included for entering the watching state phase. When the robot is in the watching state, it will be in a sitting mode. Thus, it should be in sitting mode and furthermore, give little bit of a delay time so that the robot does not respond to its own movement. The initialization also clears the wireless data once.

[Label] Watching State				
5	CALL	Sit do...		
[Label]				
6	LOAD	Timer	-	8
[Label]				
7	CALL	Timer ...		
[Label]				
8	LOAD	Clear ...	RX rem...	

< After wireless data arrives, check the control command >

The value of the robot control command is 1, sent by the Motion Detector program from the PC. When the received wireless data is 1, then robot will respond.

	[Label] Watching State Loop							[Comment]	
9	IF	( RX rem... RXD	=		0)	THEN	JUMP
	[Label]							[Comment]	
10	IF	( RX rem... RXD	=		1)	THEN	JUMP
	[Label]							[Comment]	
11	JUMP		Watchi...						

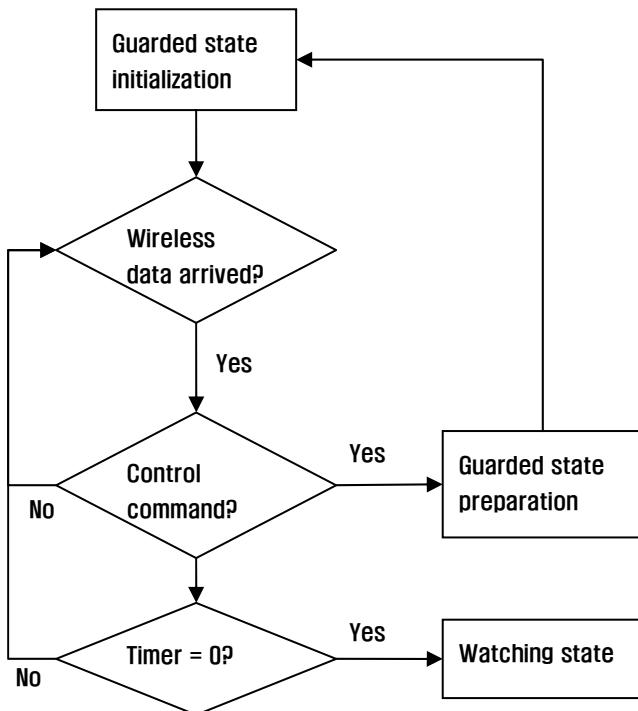
< Guarded state preparation >

In the guarded state preparation, which makes the robot continually move and bark upon detection, the robot must be in the stand position and in the guarded state.

	[Label] Ready to guard						
12	CALL		Stand ...				
	[Label]						
13	LOAD		 Timer	: -	8		
	[Label]						
14	CALL		Timer ...				
	[Label]						
15	JUMP		Guardi...				

< Guarded state >

The algorithm is as follows.



< Guarded state initialization >

Here, the initialization content is included for entering the guarded state phase. First, the program clears the wireless data and initializes the timer that will maintain the guarded state without a command.

[Label] Guarding State	
17	LOAD Clear ... (- RX rem...
[Label]	
18	LOAD Timer (- 40

< Check for wireless data arrival and timer completion >

When the robot receives a control command from the Motion Detector program, it takes the guarded state position, but if the timer completes without receiving command, it returns back to the watching state.

	[Label] Guarding State Loop							[Comment]		
19	IF	( RX rem...)	=		1)	THEN	JUMP	Analyz...	
	[Label]							[Comment]		
20	IF	( Timer)	=		0)	THEN	JUMP	Watchi...	
	[Label]							[Comment]		
21	JUMP		Guardi...							
	[Label] Analyzing Data							[Comment]		
22	IF	( RX rem...)	=		1)	THEN	JUMP	Guardi...	
	[Label]							[Comment]		
23	JUMP		Guardi...							

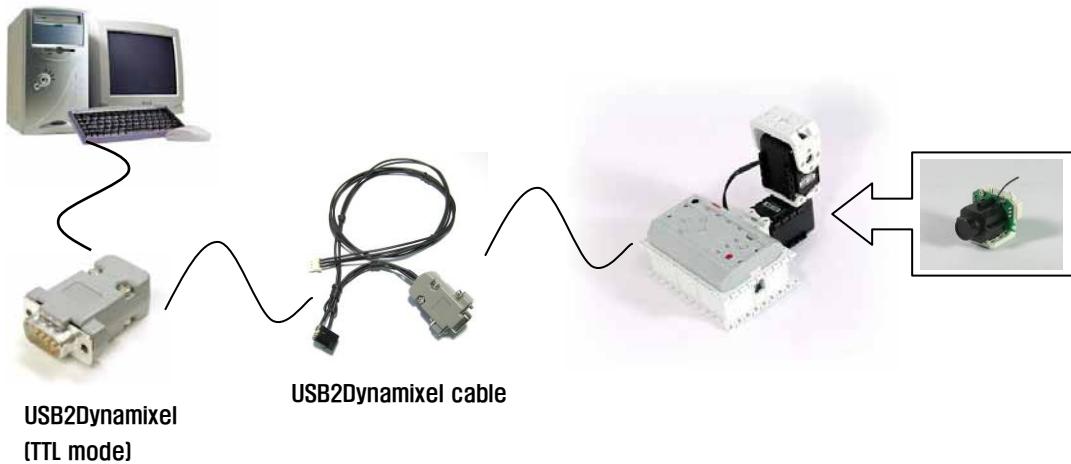
< Guarded state behavior >

The puppy robot barks in the guarded state. Again, so that the Motion Detector does not respond to the robot's own movement and send the control command, the robot will standby for fixed delay time.

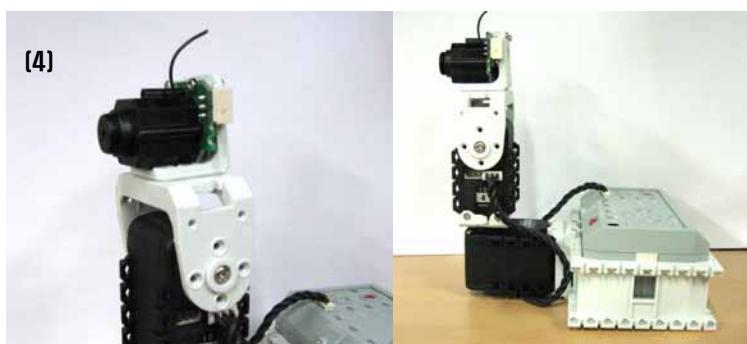
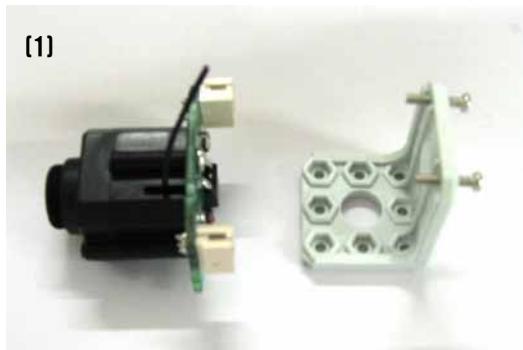
	[Label] Guarding Action									
24	CALL		Bark							
	[Label]									
25	LOAD	( Timer)	-	8						
	[Label]									
26	CALL		Timer ...							
	[Label]									
27	JUMP		Guardi...							

3 – 6 . Image Recognition and Object Tracking with a Pan/Tilt Camera Robot

In this chapter, the pan/tilt unit, a beginner example shown in the Bioloid Manual (Comprehensive Kit) will be used. The pant/tilt used here is equipped with a Zig-100 unit and a wireless image transmitter. In addition, the Zig-100 unit shall be set for 1:1 communication mode.



◆ How to install a camera



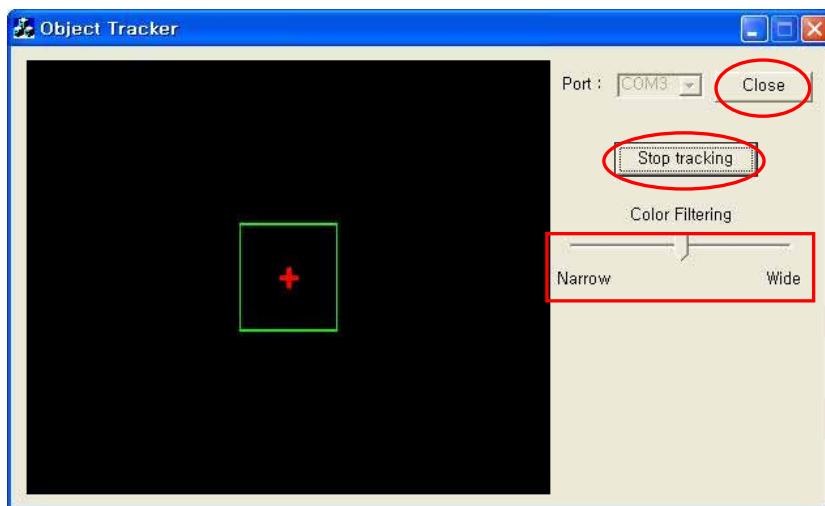
◆ Program download

- Control program

Use “Bioloid SDK \ Windows \ bin \ Object Tracker.exe”

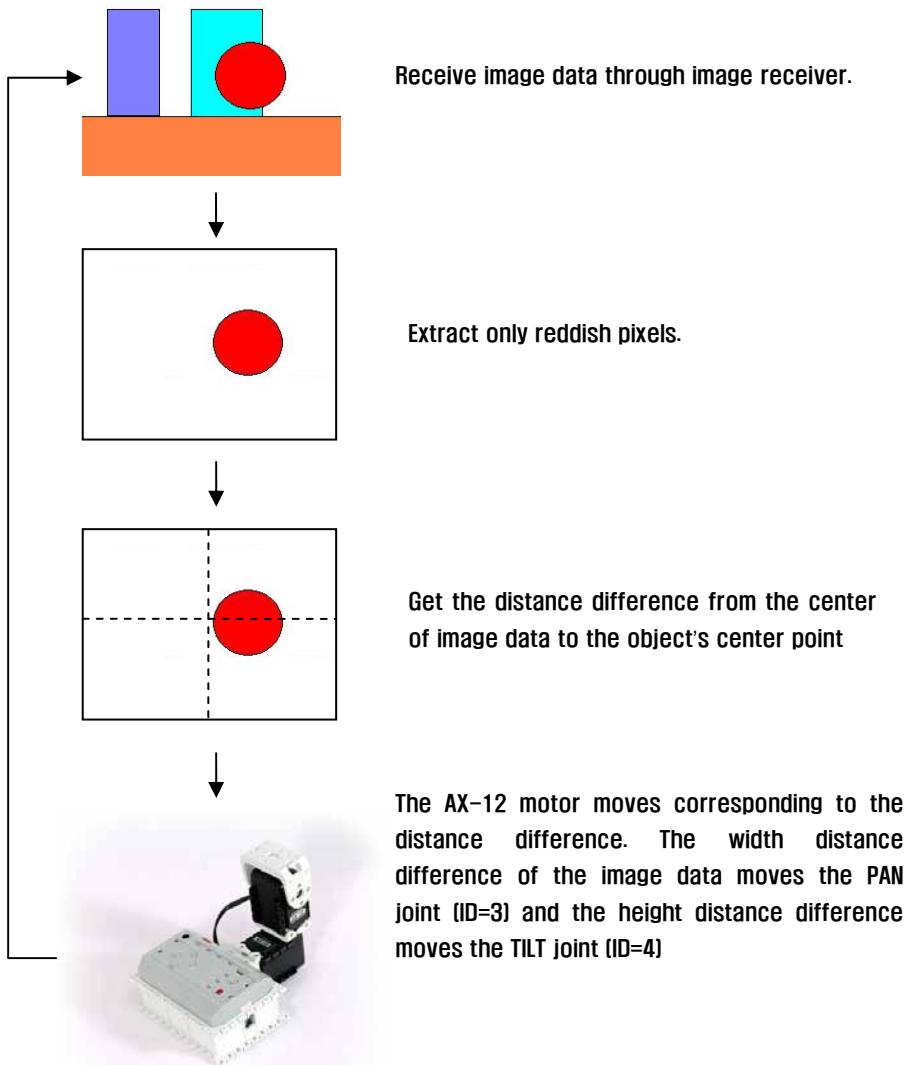
◆ Robot Operation

- Turn on the power for the CM-5.
- Execute Object Tracker.exe.
- Configure communication befitting where USB2Dynamixel is connected to the COM port of PC.
- Start the remote control by pressing the “Start tracking” button.
- Create an optimal environment by controlling the color filtering.
- When a reddish object appears, the pan/tilt device will continually move and follow the object.



◎ Control program movement process

Before proceeding with the program source analysis, let's review the overall program process.



◎ Analyzing the control program

The function of this program is to find the position of reddish objects using images acquired from the Vision library and then to track down objects by moving the AX-12 motors of the pan/tilt device by utilizing the Dynamixel Library. Before proceeding with the analysis, we recommended that you refer to books on image processing and MFC programming.

First, by using Visual C++ 6.0, open the “Bioloid SDK \Windows \app \Object Tracker \Object Tracker.dsw” file in the expansion CD provided. Most of the source code is written in the Object TrackerDlg.h and Object TrackerDlg.cpp file.

The program initiation routine is as follows.

```
BOOL CObjectTrackerDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);          // Set small icon

    // Initialize vision
    VISION_PARAM vision_param;

    vision_param.width = 320;
    vision_param.height = 240;
    vision_param.pProcessFunc = ImageProcess;

    if( vision_open( &vision_param ) == true )
        vision_start();
    else
        AfxMessageBox( "Fail to open Camera!" );

    // Initialize controls
    m_Combo_Port.SetCurSel( 0 );
    m_Button_Tracking.EnableWindow( FALSE );
    m_Slider_Range.SetRange(0, 80);
    m_Slider_Range.SetPos( 40 );

    // Initialize variables
    m_pHSI = new unsigned char[vision_param.width * vision_param.height * 3];
    m_pBinary = new unsigned char[vision_param.width * vision_param.height];

    return TRUE; // return TRUE unless you set the focus to a control
}
```

Here, the Vision Library captures images at a resolution of 320 X 240 and registers the ImageProcess function as an image processing function. Also, memory space will be allocated for image data utilized in control initialization and image processing.

The main algorithm for locating reddish objects and moving the robot is embedded in the ImageProcess function.

```
void ImageProcess( unsigned char* pImage, DWORD dwSize )
{
    CDC *pDC;
    CBitmap *pBitmap;
    CRect rtArea;
    VISION_STATE vision_state;
    int center_x, center_y;
    int object_x, object_y;

    // Initialize
    pDC = pThis->m_Static_Video.GetDC();
    vision_state = vision_get_state();
    center_x = vision_state.width/2;
    center_y = vision_state.height/2;

    // Processing
    if( pThis->m_bTracking == TRUE )
    {
        RGB24toHSI( pThis->m_pHSI, pImage, vision_state.width, vision_state.height );
        FindRedFromHSI( pThis->m_pBinary, pThis->m_Slider_Range.GetPos()
                        ,pThis->m_pHSI, vision_state.width, vision_state.height );
        GetCenterPos( &object_x, &object_y, pThis->m_pBinary, vision_state.width,
                      vision_state.height );
    }

    // Control Robot
    if( pThis->m_bTracking == TRUE )
    {
        pThis->Tracking_PanTilt( center_x - object_x, center_y - object_y );
    }

    // Converting bitmap
    if( pThis->m_bTracking == TRUE )
        pBitmap = CBitmap::FromHandle( vision_GraytoDDB( pDC->m_hDC, pThis-
>m_pBinary ) );
    else
```

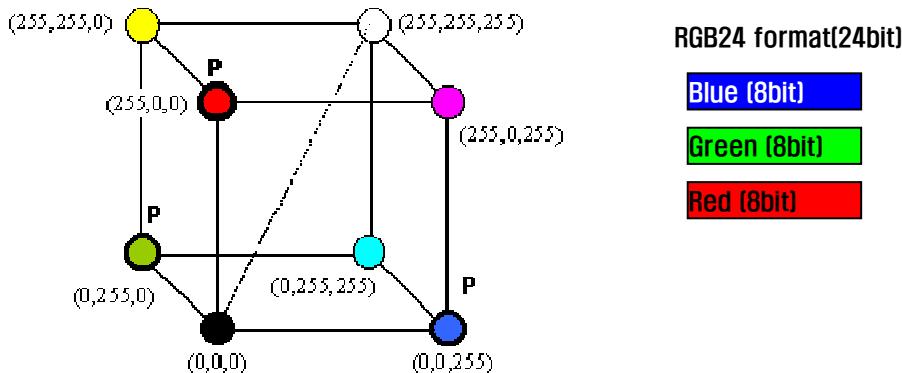
```
pBitmap = CBitmap::FromHandle( vision_RGB24toDDB( pDC->m_hDC, pImage ) );  
  
// Drawing bitmap  
if( pHsi->m_bTracking == TRUE )  
{  
    CDC MemDC;  
    CPen Pen;  
    CBrush Brush;  
    CRect rtArea;  
    CBitmap *pOldBitmap;  
  
    // Ready  
    MemDC.CreateCompatibleDC( pDC );  
    pOldBitmap = MemDC.SelectObject( pBitmap );  
  
    // Drawing center cross  
    Brush.CreateSolidBrush( RGB(0,255,0) ); // Green color  
    rtArea.SetRect( center_x-(CENTER_WIDTH/2), center_y-(CENTER_HEIGHT/2)  
                    ,center_x+(CENTER_WIDTH/2), center_y+(CENTER_HEIGHT/2) );  
    MemDC.FrameRect( rtArea, &Brush );  
  
    // Drawing object cross  
    Pen.CreatePen( PS_SOLID, 3, RGB(255,0,0) ); // Red color  
    MemDC.SelectObject( Pen );  
    MemDC.MoveTo( object_x, vision_state.height - object_y-5 );  
    MemDC.LineTo( object_x, vision_state.height - object_y+5 );  
    MemDC.MoveTo( object_x-5, vision_state.height - object_y );  
    MemDC.LineTo( object_x+5, vision_state.height - object_y );  
  
    MemDC.SelectObject( pOldBitmap );  
    Pen.DeleteObject();  
    Brush.DeleteObject();  
    MemDC.DeleteDC();  
}  
  
// Update screen  
pHsi->m_Static_Video.GetClientRect( rtArea );  
vision_DrawDDB( pDC->m_hDC, 0, 0, rtArea.Width(), rtArea.Height(),  
(HBITMAP)*pBitmap );  
pHsi->InvalidateRect( NULL );
```

```
// Release
pBitmap->DeleteObject();
pDC->DeleteDC();
}
```

< Converting RGB color model into HSI color model >

The first problem faced in image recognition is brightness processing. A camera recognizes the brightness of light and displays the pixel value on the screen. People recognize color by using the pixel value. However, it is difficult to acquire uniform images due to performance of the camera and changes in the external environment and light. This means that the program processes different images despite the fact that the image acquisition is conducted in the same environment, leading to a very low acquisition rate. Therefore, the image recognition program has to process the various images to reduce the effects caused by light.

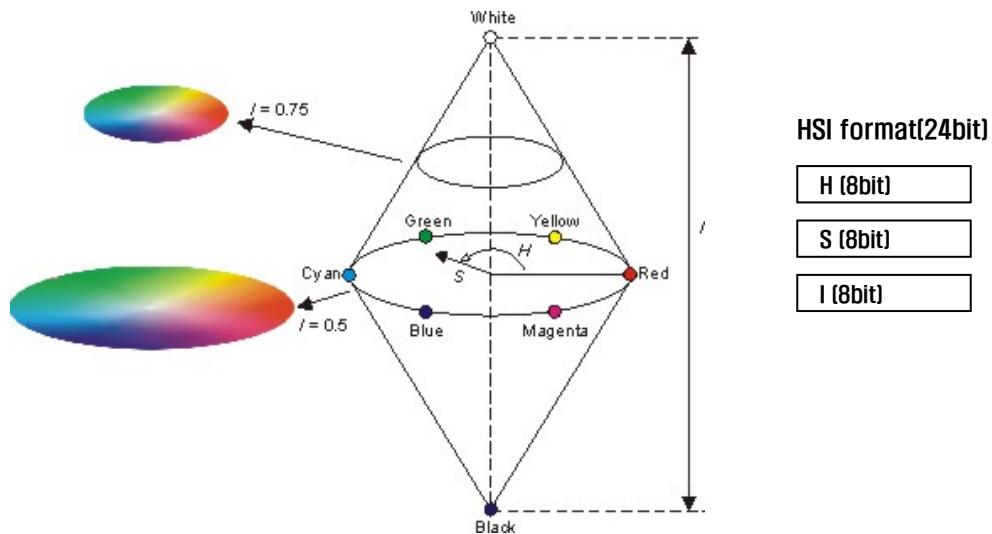
The most generally used color model is the RGB color model. A color model refers to a dimensional coordinate system used to describe the colors numerically. The RGB Color Model describes all colors using 3 colors; namely Red, Green, Blue. The pixel format of image data transmitted by the wireless image transmitter is RGB24, which describes color based on the RGB color model.



As shown in the above illustration, the RGB Color Model expresses various colors by allocating 1byte for R, G and B. This method is easy to understand, but is very difficult to use when recognizing colors and does not include information about lighting or brightness. R=100, G=10, B=10 refers to a red color and R=200, G=20, B=20 also refers to a red color. Therefore, it is impossible to differentiate this information in an image processing routine.

A solution to these problems with regards to brightness in the RGB color model is to utilize the HSI color model. The HSI color model describes color through Hue, Saturation and Intensity. That is, brightness information is included in the Intensity and the actual color of an object is described by Hue and Saturation. The following illustration shows color expressed in a circle from 0° to 360° to set the H value and Saturation, the degree of inclusion of black or white color is denoted

as S. The height of the cone expresses the brightness information, and by ignoring this value, an image recognition program can be unaffected by lighting.



A function to convert an RGB color model to the HSI color model is **RGB24toHSI**, which is described in below.

```
void RGB24toHSI( unsigned char* pHsi, unsigned char* pRGB, int width, int height )
{
    float R,G,B, H,S,I;
    float Min, Angle;
    int iH;
    int x, y, index;

    for( y=0; y<height; y++ )
    {
        for( x=0; x<width; x++ )
        {
            index = width*y + x;

            B = (float)pRGB[3*index]/255.0f;
            G = (float)pRGB[3*index+1]/255.0f;
            R = (float)pRGB[3*index+2]/255.0f;

            Min = min(min(R,B),G);

            I = (R + G + B)/3.0f;
```

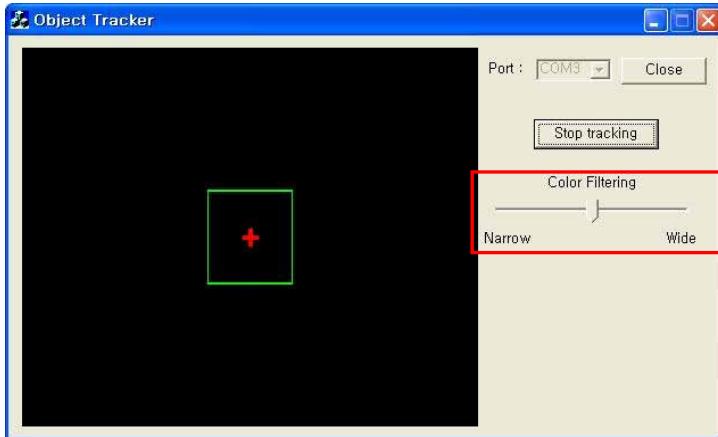
```
    if( (R == G) && (G == B) )
    {
        S = 0.0f;
        H = 0.0f;
    }
    else
    {
        S = 1.0f - (3.0f / (R + G + B)) * Min;

        Angle = (float)((R - G*0.5f - B*0.5f) / sqrt((R-G) * (R-G) + (R-B) *
(G-B)));
        H = (float)acos( Angle ) * 57.29577951f;
        if( B > G )
            H = 360.0f - H;
    }

    iH = (int)(H * 255.0f / 360.0f);
    pHSI[3*index] = (unsigned char)iH;
    pHSI[3*index+1] = (unsigned char)(S * 255.0f);
    pHSI[3*index+2] = (unsigned char)(I * 255.0f);
}
}
}
```

< Extracting a reddish pixel from an HSI Image >

In the HSI color model, a reddish color can be extracted by utilizing color information and excluding brightness information. A reddish color has an H Value range of $0 \leq H < 30$, $225 < H \leq 255$. In the case of S, white or black color is extracted from a 0 to 255 range. Therefore, an S value within the range of $80 < S < 175$ is used to extract a pure reddish color. A range of reddish color may need to be adjusted depending on the environment. A color filtering scroll bar from the program interface processes this job. When the color filtering becomes narrow, clear red colors can be extracted and when it becomes wide, all ranges of reddish color can be extracted.



FindRedFromHSI function is used to extract reddish color from the actual image data.

```
void FindRedFromHSI( unsigned char* pBinary, int range, unsigned char* pHsi, int width, int height )
{
    int x, y, index;
    int H, S;
    int radius;

    radius = range / 2;
    for( y=0; y<height; y++ )
    {
        for( x=0; x<width; x++ )
        {
            index = y*width + x;

            H = (int)pHsi[3*index];
            S = (int)pHsi[3*index+1];

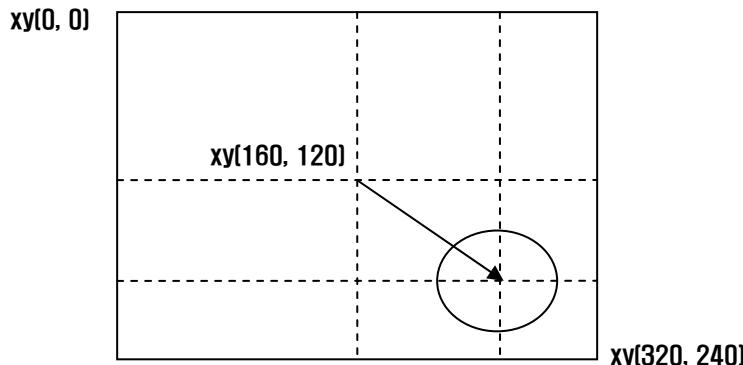
            // Red: (0 < H < 30), (80 < S < 175), (I = don't care)
            // Green: (60 < H < 120), (80 < S < 175), (I = don't care)
            // Blue: (140 < H < 200), (80 < S < 175), (I = don't care)
            if( S > 80 && S < 175 )
            {
                if( H <= radius || H >= (255-radius) )
                    pBinary[index] = 255;
                else
                    pBinary[index] = 0;
            }
            else

```

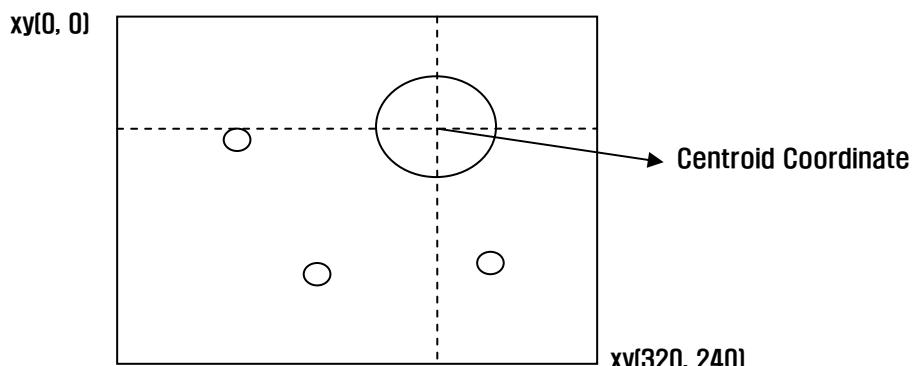
```
    pBinary[index] = 0;  
}  
}  
}
```

< Finding the central coordinate of the reddish object >

After finding the reddish object from the Image Data, the distance of the object from the center of the camera is found by calculating the difference between the central coordinate of the object and the center of the coordinate system.



An image acquired through the FindRedFromHSI function is a binary image. A reddish pixel is marked with 1 while others are marked with 0. The central coordinate of the object is the center of the object marked with 1. There are various methods for finding the centroid of an object and the Centroid method has been adopted by this program. The advantage of the Centroid method is in finding the center of an object despite various scattered pixels



The Centroid method is embedded in the GetCenterPos function.

```
void GetCenterPos( int* pX, int* pY, unsigned char* pBinary, int width, int height )  
{  
    int x, y, index;  
    int count=0;  
    long sum_x=0, sum_y=0;  
  
    // Using the centroid method
```

```
for( y=0; y<height; y++ )
{
    for( x=0; x<width; x++ )
    {
        index = y*width + x;

        if( pBinary[index] == 255 )
        {
            sum_x += x;
            sum_y += y;
            count++;
        }
    }
}

if( count > 0 )
{
    *pX = (int)(sum_x / count);
    *pY = (int)(sum_y / count);
}
else
{
    *pX = (int)(width / 2);
    *pY = (int)(height / 2);
}
```

When there is no object within the coordinate system, the function returns center coordinate of the entire image.

< Controlling the pan/tilt >

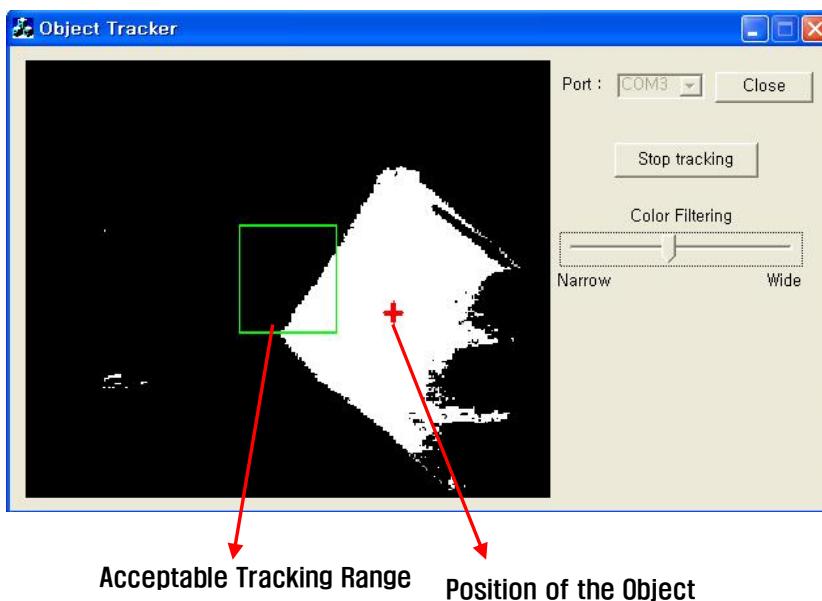
After calculating the difference between the centroid and the Center of the coordinate system, the result can be utilized to control the Pan/tilt. This function is embedded in the Tracking_PanTilt Function.

```
void CObjectTrackerDlg::Tracking_PanTilt( int diff_pan, int diff_tilt )
{
    // Tracking pan
    if( diff_pan > -(CENTER_WIDTH/2) && diff_pan < (CENTER_WIDTH/2) )
        Stop_Motor( PAN_ID );
    else
    {
        if( diff_pan < 0 )
            Move_Motor( PAN_ID, 0, TRACKING_VELOCITY );
    }
}
```

```
else
    Move_Motor( PAN_ID, 1023, TRACKING_VELOCITY );
}

// Tracking tilt
if( diff_tilt > -(CENTER_HEIGHT/2) && diff_tilt < (CENTER_HEIGHT/2) )
    Stop_Motor( TILT_ID );
else
{
    if( diff_tilt > 0 )
        Move_Motor( TILT_ID, 0, TRACKING_VELOCITY );
    else
        Move_Motor( TILT_ID, 1023, TRACKING_VELOCITY );
}
```

In the source code of this function, variable `diff_pan` and `diff_tilt` is used for storing distance differences from the central position. The control is separated into pan control (left and right control) and tilt control (up and down control). When the difference result falls below a center range of values, the program stops the motor and marks the acceptable tracking range. When the object is not detected within the acceptable tracking range, the goal position of the AX-12 motor is adjusted from 0 to 1023 following the +, - adjustments, to control the pan/tilt device and track down the object.



The actual functions utilized when moving the AX-12 motor are the Stop_Motor and Motor_Move functions. Refer to “2-2-2 Dynamixel Control” for control applications that utilize the Dynamixel library.

```
void CObjectTrackerDlg::Move_Motor( int id, int pos, int speed )
{
    // Packet structure
    dxl_instruction InstPacket;
    dxl_status StatusPacket;

    // Make packet
    InstPacket.id          = id;
    InstPacket.instruction = INST_WRITE;
    InstPacket.address     = P_GOAL_POSITION_L;
    InstPacket.parameter[0] = LOW_BYT(pos);
    InstPacket.parameter[1] = HIGH_BYT(pos);
    InstPacket.parameter[2] = LOW_BYT(speed);
    InstPacket.parameter[3] = HIGH_BYT(speed);
    InstPacket.length       = 7;

    // Communication
    dxl_control( &InstPacket, &StatusPacket, true );
}
```

```
void CObjectTrackerDlg::Stop_Motor( int id )
{
    // Packet structure
    dxl_instruction InstPacket;
    dxl_status StatusPacket;
    int ErrorCode;
    WORD pos;

    // Read present position
    InstPacket.id          = id;
    InstPacket.instruction = INST_READ;
    InstPacket.address     = P_PRESENT_POSITION_L;
    InstPacket.parameter[0] = 2;
    InstPacket.length       = 4;

    ErrorCode = dxl_control( &InstPacket, &StatusPacket, true );
    if( ErrorCode == ERR_CODE_SUCCESS ) // Communication success
        pos = MAKE_WORD(StatusPacket.parameter[0], StatusPacket.parameter[1]);
}
```

```
    else
        pos = 512;

    Move_Motor( id, pos, TRACKING_VELOCITY );
}
```

After the control, the result is outputted on a screen. A Win32GDI related API can be utilized for outputting necessary information on the screen. Refer to Win32 programming books for more details. Take note that displayed information on the screen are the coordinates stored in the Bitmap memory, which is a mirror image of information to be display on the screen. Therefore, the y coordinate has to be reversed prior to the display.

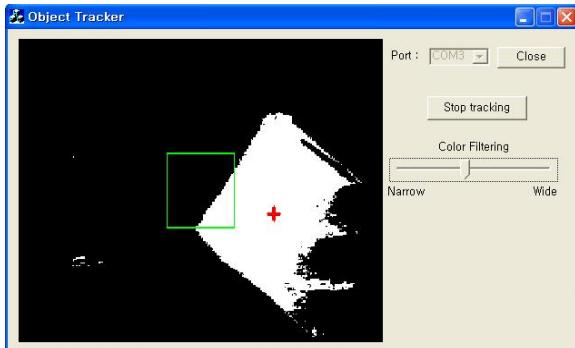
```
// Drawing bitmap
if( pThis->m_bTracking == TRUE )
{
    CDC MemDC;
    CPen Pen;
    CBrush Brush;
    CRect rtArea;
    CBitmap *pOldBitmap;

    // Ready
    MemDC.CreateCompatibleDC( pDC );
    pOldBitmap = MemDC.SelectObject( pBitmap );

    // Drawing center area
    Brush.CreateSolidBrush( RGB(0,255,0) ); // Green color
    rtArea.SetRect( center_x-(CENTER_WIDTH/2), center_y-(CENTER_HEIGHT/2)
                    ,center_x+(CENTER_WIDTH/2), center_y+(CENTER_HEIGHT/2) );
    MemDC.FrameRect( rtArea, &Brush );

    // Drawing object cross
    Pen.CreatePen( PS_SOLID, 3, RGB(255,0,0) ); // Red color
    MemDC.SelectObject( Pen );
    MemDC.MoveTo( object_x, vision_state.height - object_y-5 );
    MemDC.LineTo( object_x, vision_state.height - object_y+5 );
    MemDC.MoveTo( object_x-5, vision_state.height - object_y );
    MemDC.LineTo( object_x+5, vision_state.height - object_y );

    MemDC.SelectObject( pOldBitmap );
    Pen.DeleteObject();
    Brush.DeleteObject();
    MemDC.DeleteDC();
}
```

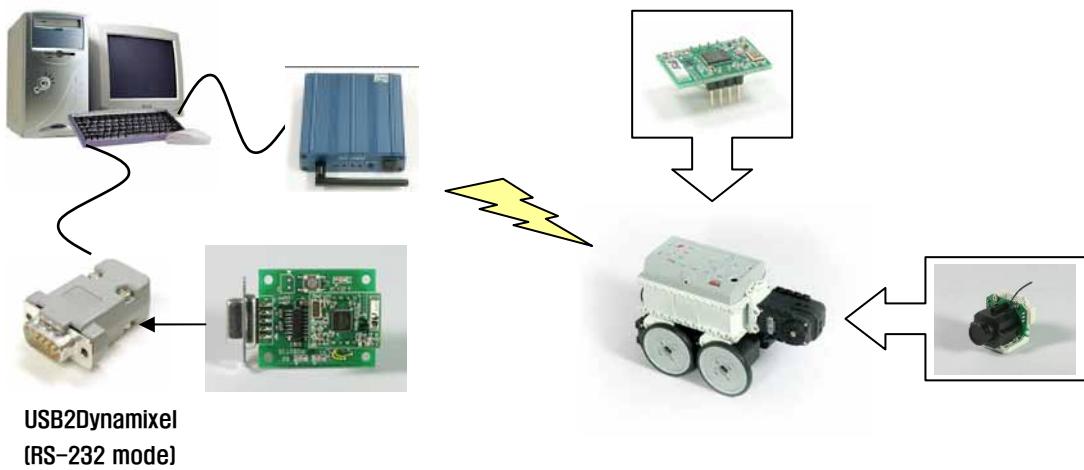


For convenience, there is a function that automatically adjusts the pan/tilt device to the center position when the USB2Dynamixel is connected. This function is referred as “home location transfer” and is embedded in the Home_PanTilt function.

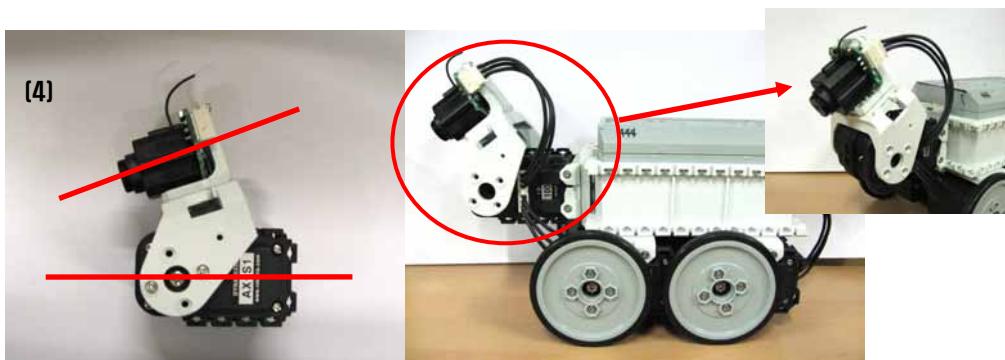
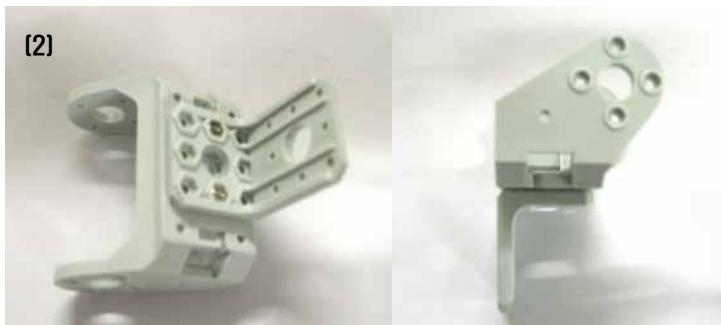
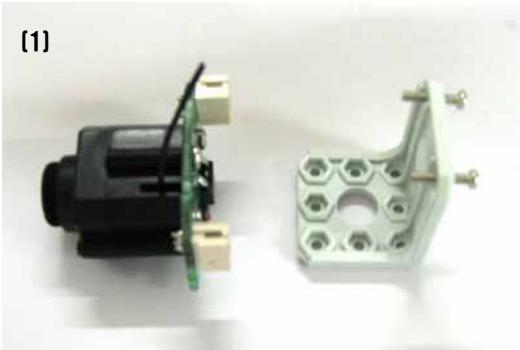
```
void CObjectTrackerDlg::Home_PanTilt()
{
    Move_Motor( PAN_ID, 512, TRACKING_VELOCITY );
    Move_Motor( TILT_ID, 512, TRACKING_VELOCITY );
}
```

3 - 7. Image Recognition and a Self-Controlled Wheeled Robot

In this chapter, the cliff detection car, a beginner example shown in the Bioloid Manual (Comprehensive Kit) will be used. The obstacle detection car used here is equipped with a Zig-100 unit and wireless image transmitter. In addition, the Zig-100 unit is set for 1:1 communication mode.



◆ How to install a camera



◆ Program download

- Remote control

Use “Bioloid SDK \ Windows \ bin \ Path Finder.exe”

- Cliff detection car

Use “Bioloid SDK \ CM-5 \ bin \ example_cliffdetectioncar.hex”

◆ Robot Operation

- Execute cliff detection car program.

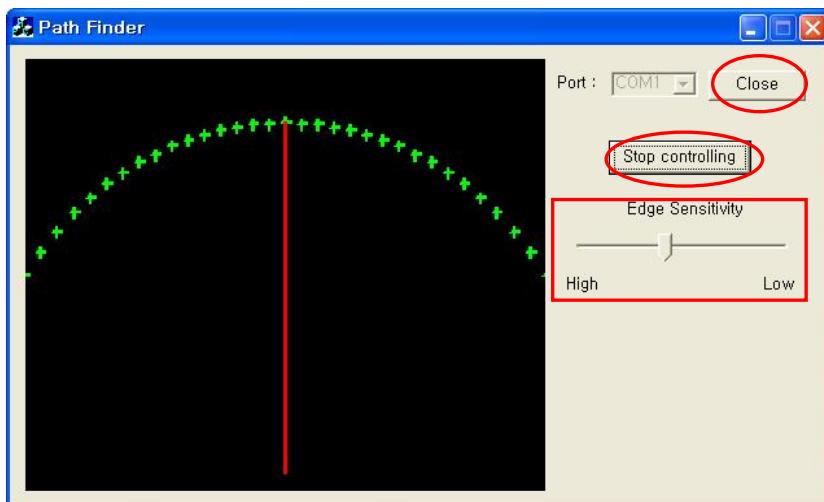
- Execute Path Finder.exe.

- Configure communication befitting when Zig2Serial B/D is connected to the COM port of PC.

- Start the remote control pressing the “Start controlling” button.

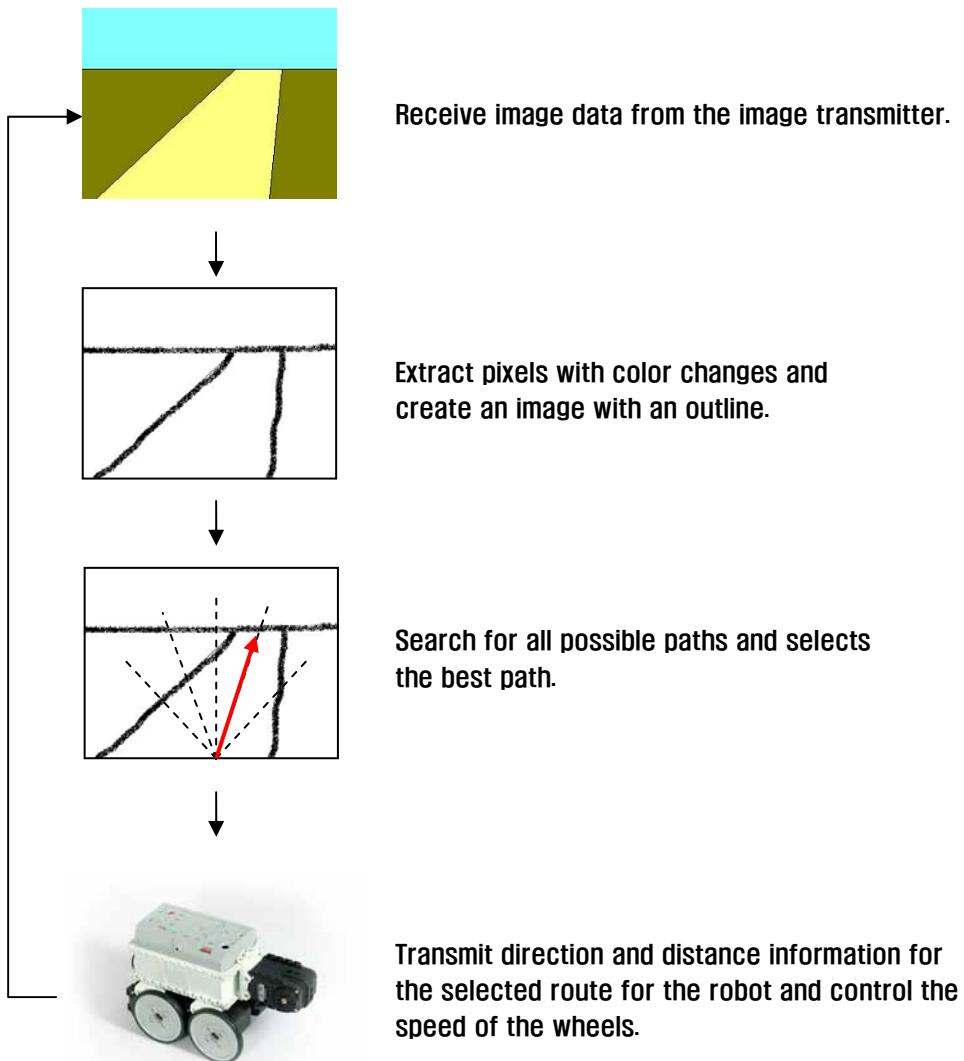
- Create an optimal environment by controlling the edge sensitivity.

- When the cliff detection car program starts, it will move toward the direction of a reddish color.



◎ Control algorithm

Let's look at the overall control algorithm prior to analyzing the program source code.



◎ Analyzing the path finder program

This program searches for all possible paths for the robot by utilizing images acquired through the vision library and utilizes the Zigbee library to remotely control the robot. Before proceeding with the analysis, you are recommended to refer books on image processing and MFC programming.

First, using Visual C++ 6.0, open the “Bioloid SDK \Windows \app \Object Tracker \Object Tracker.dsw” file from the expansion CD provided. Most of the source code is written in the Object TrackerDlg.h and Object TrackerDlg.cpp file.

The program initiation routine is as follows.

```
BOOL CPathFinderDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);           // Set big icon
    SetIcon(m_hIcon, FALSE);         // Set small icon

    // Initialize vision
    VISION_PARAM vision_param;

    vision_param.width = 320;
    vision_param.height = 240;
    vision_param.pProcessFunc = ImageProcess;

    if( vision_open( &vision_param ) == true )
        vision_start();
    else
        AfxMessageBox( "Fail to open Camera!" );

    // Initialize controls
    m_Combo_Port.SetCurSel( 0 );
    m_Button_Control.EnableWindow( FALSE );
    m_Slider_Sensitivity.SetRange( 80, 250 );
    m_Slider_Sensitivity.SetPos( 150 );

    // Initialize variables
    m_pHSI          = new unsigned char[vision_param.width * vision_param.height *
3];
    m_pEdge_H       = new unsigned char[vision_param.width * vision_param.height];
    m_pEdge_S       = new unsigned char[vision_param.width * vision_param.height];
```

```
m_pOR_HS      = new unsigned char[vision_param.width * vision_param.height];
m_pResult      = new unsigned char[vision_param.width * vision_param.height];
m_pTempImage = new unsigned char[vision_param.width * vision_param.height];

// Initialize path data
InitPathData( 10, 2*ROBOT_DIRECTION_LEVEL+1, vision_param.width,
vision_param.height );

return TRUE; // return TRUE unless you set the focus to a control
}
```

Here, the vision library captures images at a resolution of 320 X 240 and registers the ImageProcess function as an image processing function. Also, memory space will be allocated for image data utilized in control initialization and image processing.

The main algorithm for path finding and controlling the robot is embedded in the ImageProcess function.

```
void ImageProcess( unsigned char* pImage, DWORD dwSize )
{
    CDC *pDC;
    CBitmap *pBitmap;
    CRect rtArea;
    VISION_STATE vision_state;

    // Initialize
    pDC = pThis->m_Static_Video.GetDC();
    vision_state = vision_get_state();

    // Processing
    if( pThis->m_bControl == TRUE )
    {
        // Converting RGB -> HSI color model
        RGB24toHSI( pThis->m_pHSI, pImage, vision_state.width, vision_state.height );

        // Processing H channel
        GetSngleChannel( pThis->m_pTempImage, pThis->m_pHSI, 0, 3,
vision_state.width, vision_state.height );
        Sobel_Edge( pThis->m_pEdge_H, pThis->m_pTempImage, vision_state.width,
vision_state.height );
        SetThreshold( pThis->m_Slider_Sensitivity.GetPos(), 240, pThis->m_pEdge_H,
vision_state.width, vision_state.height );
    }
}
```

```
    Erosion( pThis->m_pTempImage, pThis->m_pEdge_H, vision_state.width,
vision_state.height );
    Dilation( pThis->m_pEdge_H, pThis->m_pTempImage, vision_state.width,
vision_state.height );

    // Processing S channel
    GetSngleChannel( pThis->m_pTempImage, pThis->m_pHSI, 1, 3,
vision_state.width, vision_state.height );
    Sobel_Edge( pThis->m_pEdge_S, pThis->m_pTempImage, vision_state.width,
vision_state.height );
    SetThreshold( pThis->m_Slider_Sensitivity.GetPos(), 255, pThis->m_pEdge_S,
vision_state.width, vision_state.height );
    Erosion( pThis->m_pTempImage, pThis->m_pEdge_S, vision_state.width,
vision_state.height );
    Dilation( pThis->m_pEdge_S, pThis->m_pTempImage, vision_state.width,
vision_state.height );

    // Result = H OR S
    Image_OR( pThis->m_pResult, pThis->m_pEdge_H, pThis->m_pEdge_S,
vision_state.width, vision_state.height );

    // Find best path
    pThis->GetEdgePoint( pThis->m_pResult, vision_state.width,
vision_state.height );
    pThis->FindBestPoint();

    // Control Robot
    pThis->ControlRobot();
}

// Converting bitmap
if( pThis->m_bControl == TRUE )
    pBitmap = CBitmap::FromHandle( vision_GraytoDDB( pDC->m_hDC, pThis-
>m_pResult ) );
else
    pBitmap = CBitmap::FromHandle( vision_RGB24toDDB( pDC->m_hDC, pImage ) );

// Drawing bitmap
if( pThis->m_bControl == TRUE )
{
    CDC MemDC;
    CPen Pen1, Pen2;
```

```
CBitmap *pOldBitmap;

MemDC.CreateCompatibleDC( pDC );
Pen1.CreatePen( PS_SOLID, 2, RGB(0,255,0) ); // Green color
Pen2.CreatePen( PS_SOLID, 2, RGB(255,0,0) ); // Red color

pOldBitmap = MemDC.SelectObject( pBitmap );

// Drawing path
MemDC.SelectObject( Pen1 );
for( int i=0; i<pThis->m_NumPath; i++ )
{
    MemDC.MoveTo( pThis->m_pEdgePos[i].x, vision_state.height - pThis->m_pEdgePos[i].y-3 );
    MemDC.LineTo( pThis->m_pEdgePos[i].x, vision_state.height - pThis->m_pEdgePos[i].y+3 );
    MemDC.MoveTo( pThis->m_pEdgePos[i].x-3, vision_state.height - pThis->m_pEdgePos[i].y );
    MemDC.LineTo( pThis->m_pEdgePos[i].x+3, vision_state.height - pThis->m_pEdgePos[i].y );
}

MemDC.SelectObject( Pen2 );
MemDC.MoveTo( pThis->m_SrcPos.x, vision_state.height - pThis->m_SrcPos.y );
MemDC.LineTo( pThis->m_BestPos.x, vision_state.height - pThis->m_BestPos.y );

MemDC.SelectObject( pOldBitmap );
Pen1.DeleteObject();
Pen2.DeleteObject();
MemDC.DeleteDC();
}

// Update screen
pThis->m_Static_Video.GetClientRect( rtArea );
vision_DrawDDB( pDC->m_hDC, 0, 0, rtArea.Width(), rtArea.Height(),
(HBITMAP)*pBitmap );
pThis->InvalidateRect( NULL );

// Release
pBitmap->DeleteObject();
pDC->DeleteDC();
}
```

< Converting an RGB color model to an HSI color model >

The RGB and HSI color models were explained in “3-6, Image Recognition Object Tracking for the Pan/Tilt Robot.” Refer to the Chapter 3-6 for more details. The biggest reason for using the HSI color model is to minimize the effect of brightness.

```
void RGB24toHSI( unsigned char* pHsi, unsigned char* pRGB, int width, int height )
{
    float R,G,B, H,S,I;
    float Min, Angle;
    int iH;
    int x, y, index;

    for( y=0; y<height; y++ )
    {
        for( x=0; x<width; x++ )
        {
            index = width*y + x;

            B = (float)pRGB[3*index]/255.0f;
            G = (float)pRGB[3*index+1]/255.0f;
            R = (float)pRGB[3*index+2]/255.0f;

            Min = min(min(R,B),G);

            I = (R + G + B)/3.0f;

            if( (R == G) && (G == B) )
            {
                S = 0.0f;
                H = 0.0f;
            }
            else
            {
                S = 1.0f - (3.0f / (R + G + B)) * Min;

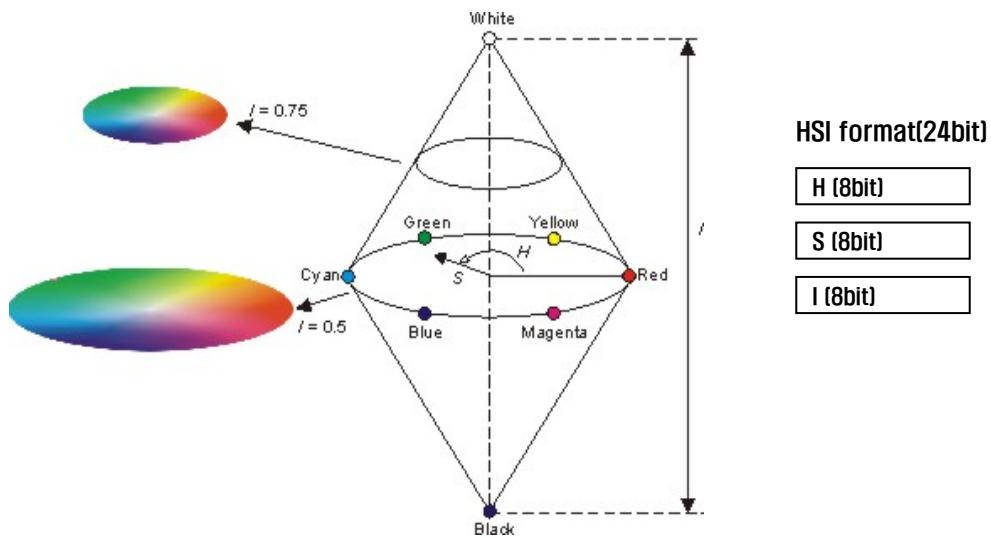
                Angle = (float)((R - G*0.5f - B*0.5f) / sqrt((R-G) * (R-G) + (R-B) *
(G-B)));
                H = (float)acos( Angle ) * 57.29577951f;
                if( B > G )
                    H = 360.0f - H;
            }
        }
    }
}
```

```
iH = (int)(H * 255.0f / 360.0f);
pHSI[3*index] = (unsigned char)iH;
pHSI[3*index+1] = (unsigned char)(S * 255.0f);
pHSI[3*index+2] = (unsigned char)(I * 255.0f);
}
}
```

< Extracting outline by changing the colors of HSI Images >

In most of cases, floors have uniform color. That is, the absence of color changes can be thought of as a path, and color changes can be thought of as obstacles.

The “I” channel from HSI images will be ignored since it is the brightness information. Information on color changes exists in the H and S channels. Therefore, color changes can be detected by using these two channels. The reason for using both channels is that the H channel may be adequate for color changes alone, but the S channel is included with information on color differentiation of a uniform color and the difference between white and black color. Therefore, the final outline images must be combined with results acquired from the H and S channels. In the following example, the OR operation on pixels was conducted to combine the results.



First, it is necessary to extract images on each channel from an HSI image. This function is embedded in the GetSingleChannel function.

```
void GetSngleChannel( unsigned char* pSingle, unsigned char* pMulti, int nChannel,
int Total, int width, int height )
{
    int x, y, index;
```

```
for( y=0; y<height; y++ )  
{  
    for( x=0; x<width; x++ )  
    {  
        index = width*y + x;  
  
        pSingle[index] = pMulti[Total*index+nChannel];  
    }  
}  
}
```

There are various methods for extracting the outline of an image. The above example has adopted the Sobel mask outline extraction method.



Original Image



Sobel



Prewitt



Robert



Laplacian



Canny

Conduct the outline extraction procedure on each channel using the Sobel mask operation method. The outline extraction method utilizing Sobel mask is embedded in the sobel_Edge function.

```
void Sobel_Edge( unsigned char* pResult, unsigned char* pOrigin, int width, int height )
{
    int x, y, i, j, index;
    int sum_x, sum_y, sum, origin;
    int gx_mask[3][3] = {-1, 0, 1, -2, 0, 2, -1, 0, 1}; // Sobel gx mask
    int gy_mask[3][3] = {1, 2, 1, 0, 0, 0, -1, -2, -1}; // Sobel gy mask
    int* pTempImage;
    int min, max, newValue;
    float constVal1, constVal2;

    pTempImage = new int[width*height];

    // Edge detection by Sobel mask
    for( y=0; y<height; y++ )
    {
        for( x=0; x<width; x++ )
        {
            sum = 0;

            if( (x != 0 && x != width-1) && (y != 0 && y != height-1) )
            {
                sum_x = 0;
                sum_y = 0;

                for( j=-1; j<=1; j++ )
                {
                    for( i=-1; i<=1; i++ )
                    {
                        origin = (int)pOrigin[(y+j) * width + (x+i)];
                        sum_x += origin * gx_mask[j+1][i+1];
                        sum_y += origin * gy_mask[j+1][i+1];
                    }
                }
                sum = abs(sum_x) + abs(sum_y);
                sum = max(0, min(sum, 255));
            }
            index = y*width + x;
            pResult[index] = sum;
        }
    }
}
```

```
    pTempImage[index] = sum;
}

}

// Find min, max value
min = (int)10e10;
max = (int)-10e10;
for( y=0; y<height; y++ )
{
    for( x=0; x<width; x++ )
    {
        index = y*width + x;
        if( pTempImage[index] < min )
            min = pTempImage[index];
        if( pTempImage[index] > max )
            max = pTempImage[index];
    }
}

// Normalize level
constVal1 = (float)(255.0 / (max-min));
constVal2 = (float)(255.0*min / (max-min));
for( y=0; y<height; y++ )
{
    for( x=0; x<width; x++ )
    {
        index = y*width + x;
        newValue = (int)(constVal1 * (float)pTempImage[index] + constVal2);
        pResult[index] = (unsigned char)newValue;
    }
}

delete pTempImage;
}
```

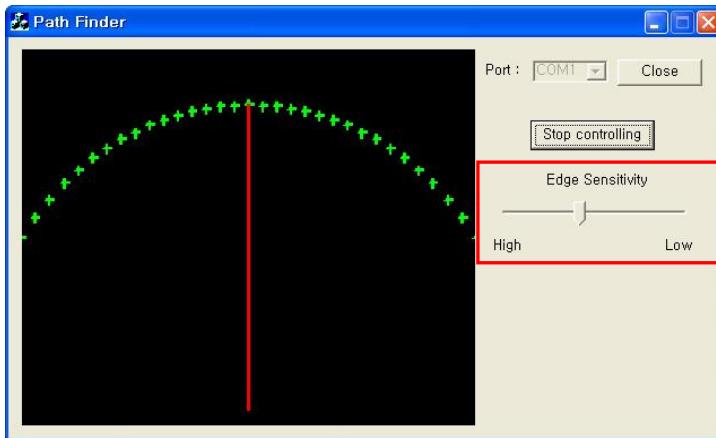
After extracting the outline, pixels have to be classified as either an outline or non-outline pixel. That is, a binary image consisting of 0s and 1s is created. The SetThreshold function is used to convert pixels falling under a given range to 1s and converts other pixels into 0s.

```
void SetThreshold( int Threshold_L, int Threshold_H, unsigned char* pImage, int
width, int height )
{
    int x, y, index;
```

```
// Convert Image (Gray -> Binary)
for( y=0; y<height; y++ )
{
    for( x=0; x<width; x++ )
    {
        index = y*width + x;

        if( pImage[index] >= (unsigned char)Threshold_L
            && pImage[index] <= (unsigned char)Threshold_H )
            pImage[index] = 255; // 1 (Binary)
        else
            pImage[index] = 0;           // 0 (Binary)
    }
}
}
```

Among the parameters of this function, Threshold_L can be adjusted by using the Edge Sensitivity scroll bar shown on the Program Interface



After acquiring the outline, there is noise induced by the camera and other environmental factors. There are many methods for removing noise in the image and below, the example has adopted the Erosion and Dilation operation methods for removing noise.



Original image

Image after Erosion operation

Image after Dilation operation

The Erosion operation on outline images acquired from each channel is embedded in the Erosion function

```
void Erosion( unsigned char* pResult, unsigned char* pSource, int width, int height )
{
    int x, y, index;
    int ma_x, ma_y, ma_index;
    int sum = 0;

    // Set filter
    int szMask_x = 2, szMask_y = 2; // size of mask
    int threshold = 1;

    for( y=0; y<height; y++ )
    {
        for( x=0; x<width; x++ )
        {
            index = width*y + x;

            if( y > (height-szMask_y) || x > (width-szMask_x) )
                pResult[index] = 0;
            else
            {
                // Mask process
                sum = 0;
                for( ma_y=0; ma_y<szMask_y; ma_y++ )
                {
                    for( ma_x=0; ma_x<szMask_x; ma_x++ )
                    {
                        ma_index = width*(y+ma_y) + (x+ma_x);
                        if( pSource[ma_index] == 0 )
                            sum++;
                    }
                }
                if( sum < threshold )
                    pResult[index] = 0;
            }
        }
    }
}
```

```
        sum++;
    }

}

// Determine value
if( sum >= threshold )
    pResult[index] = 0;
else
    pResult[index] = 255;
}
}
}
}
```

After conducting the Erosion operation, the Dilation operation has to be conducted for restoration. This function is embedded in the Dilation function.

```
void Dilation( unsigned char* pResult, unsigned char* pSource, int width, int
height )
{
    int x, y, index;
    int ma_x, ma_y, ma_index;
    int sum = 0;

    // Set filter
    int szMask_x = 2, szMask_y = 2; // size of mask
    int threshold = 1;

    for( y=0; y<height; y++ )
    {
        for( x=0; x<width; x++ )
        {
            index = width*y + x;

            if( y > (height-szMask_y) || x > (width-szMask_x) )
                pResult[index] = 0;
            else
            {
                // Mask process
                sum = 0;
                for( ma_y=0; ma_y<szMask_y; ma_y++ )
                {
                    for( ma_x=0; ma_x<szMask_x; ma_x++ )
```

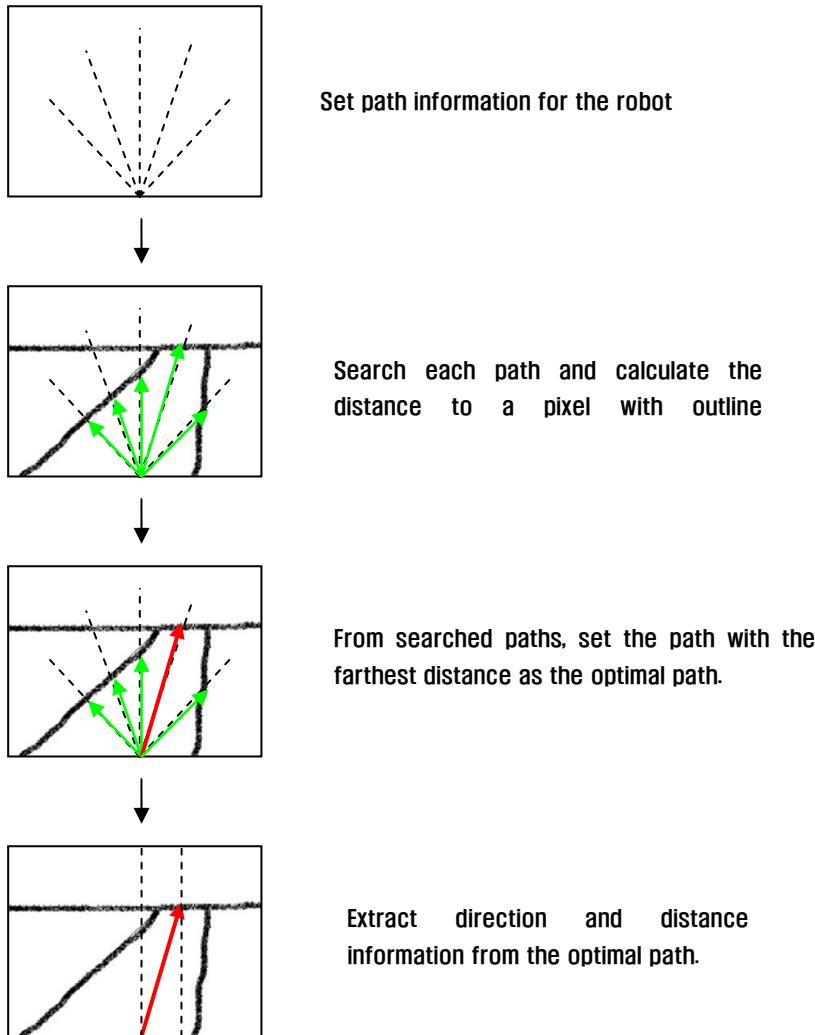
```
{  
    ma_index = width*(y+ma_y) + (x+ma_x);  
    if( pSource[ma_index] == 255 )  
        sum++;  
}  
}  
  
// Determine value  
if( sum >= threshold )  
    pResult[index] = 255;  
else  
    pResult[index] = 0;  
}  
}  
}  
}
```

By conducting the above procedures on channels H and S, outline images of how channel H changed with color changes and how channel S changed with saturation changes can be acquired. Using these two pieces of image information, the OR operation will be conducted on a pixel by pixel basis. The OR operation for a pixel by pixel basis is embedded in the Image_OR function.

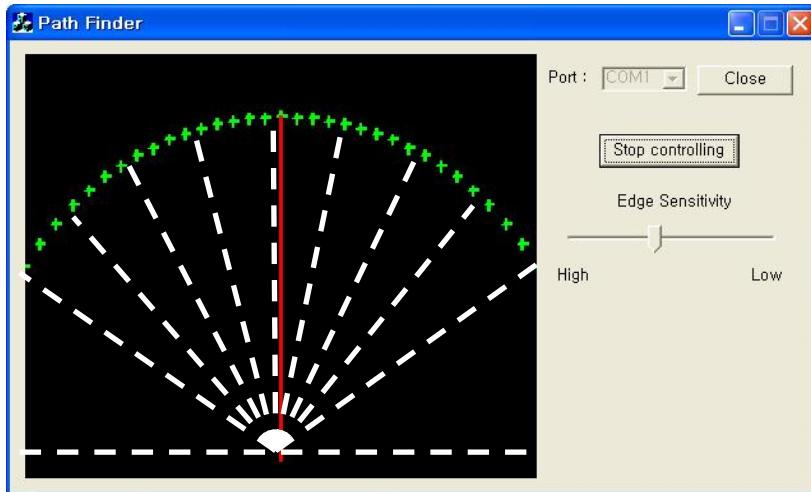
```
void Image_OR( unsigned char* pResult, unsigned char* pImage1, unsigned char*  
pImage2, int width, int height )  
{  
    int x, y, index;  
  
    for( y=0; y<height; y++ )  
    {  
        for( x=0; x<width; x++ )  
        {  
            index = width*y + x;  
  
            if( pImage1[index] > 0 || pImage2[index] > 0 )  
                pResult[index] = 255;  
            else  
                pResult[index] = 0;  
        }  
    }  
}
```

< Path finding from an outline image >

The algorithm for path finding is as follows.



The InitPathData function is used for initializing path information. The path information has a vector form and allocates a uniform length to the different paths. An offset is used along the Y Axis to eliminate effects from noises.



```
void CPathFinderDlg::InitPathData( int offset, int NumPath, int width, int height )
{
    int i;
    double x, y, radius, width_diff;

    m_NumPath = NumPath;

    m_SrcPos.x = width / 2;
    m_SrcPos.y = offset;

    m_pEdgePos = new CPoint[NumPath];
    m_pDestPos = new CPoint[NumPath];

    width_diff = width / (NumPath-1);
    radius = sqrt( m_SrcPos.x*m_SrcPos.x + (height/2 - offset)*(height/2 - offset) );

    // Path right part
    for( i=0; i<((NumPath-1)/2); i++ )
    {
        x = width_diff * i;
        y = sqrt( radius*radius - (x - m_SrcPos.x)*(x - m_SrcPos.x) ) + m_SrcPos.y;
```

```
m_pDestPos[i].x = (long)x;
m_pDestPos[i].y = (long)y;
}

// Path center part
m_pDestPos[(NumPath-1)/2].x = (long)(width / 2);
m_pDestPos[(NumPath-1)/2].y = (long)(offset + radius);

// Path left part
for( i=0; i<(NumPath-1)/2; i++ )
{
    x = width - width_diff * i;
    y = sqrt( radius*radius - (x - m_SrcPos.x)*(x - m_SrcPos.x) ) + m_SrcPos.y;

    m_pDestPos[(NumPath-1)-i].x = (long)x;
    m_pDestPos[(NumPath-1)-i].y = (long)y;
}
}
```

The NumPath parameter from the above function refers to number of paths. The number of paths is calculated as [the number of possible paths in a unidirection x 2 + 1]. For example, to set 8 paths in a unidirection, the number for all of the possible paths becomes $17 = 8 \times 2 + 1$.

The GetEdgePoint function is a function utilized to search all possible paths from an outline image acquired through image processing.

```
void CPathFinderDlg::GetEdgePoint( unsigned char* pImage, int width, int height )
{
    int i, x, y, diff_x, diff_y;
    double a;

    for( i=0; i<m_NumPath; i++ )
    {
        x = y = 0;
        diff_x = m_pDestPos[i].x - m_SrcPos.x;
        diff_y = m_pDestPos[i].y - m_SrcPos.y;

        if( diff_x == 0 )
            a = 0;
        else if( diff_x == diff_y )
            a = 1;
        else
            a = (double)diff_y / (double)diff_x;
```

```
if( abs(diff_x) <= abs(diff_y) )
{
    if( diff_y > 0 )
    {
        for( y=m_SrcPos.y; y<=m_pDestPos[i].y; y++ )
        {
            if( a == 0 )
                x = m_SrcPos.x;
            else
            {
                x = (int)((double)(y - m_SrcPos.y) / a + m_SrcPos.x);
                x = max(0, min(x, width));
            }

            if( pImage[y*width+x] == 255 )
                break;
        }
    }
    else
    {
        for( y=m_SrcPos.y; y>=m_pDestPos[i].y; y-- )
        {
            if( a == 0 )
                x = m_SrcPos.x;
            else
            {
                x = (int)((double)(y - m_SrcPos.y) / a + m_SrcPos.x);
                x = max(0, min(x, width));
            }

            if( pImage[y*width+x] == 255 )
                break;
        }
    }
}
else
{
    if( diff_x > 0 )
    {
        for( x=m_SrcPos.x; x<=m_pDestPos[i].x; x++ )
        {
            y = (int)(a * (double)(x - m_SrcPos.x) + m_SrcPos.y);
            if( pImage[y*width+x] == 255 )
                break;
        }
    }
}
```

```

        y = max(0, min(y, height));

        if( pImage[y*width+x] == 255 )
            break;
        }
    }
else
{
    for( x=m_SrcPos.x; x>=m_pDestPos[i].x; x-- )
    {
        y = (int)(a * (double)(x - m_SrcPos.x) + m_SrcPos.y);
        y = max(0, min(y, height));

        if( pImage[y*width+x] == 255 )
            break;
        }
    }

    m_pEdgePos[i].x = x;
    m_pEdgePos[i].y = y;
}
}
}

```

After finding all possible paths, use the FindBestPoint function to find the optimal path.

```

void CPathFinderDlg::FindBestPoint()
{
    int i, R_index, L_index;
    int length, R_max, L_max;
    double weight, diff_x, diff_y;

    // Search left part of best point
    L_index = 0;
    L_max = 0;
    for( i=m_BestIndex; i>=0; i-- )
    {
        weight = 1 - (0.02 * abs((m_NumPath-1)/2 - i));
        diff_x = m_pEdgePos[i].x - m_SrcPos.x;
        diff_y = m_pEdgePos[i].y - m_SrcPos.y;
        length = (int)(sqrt( diff_x * diff_x + diff_y * diff_y ) * weight);

        if( length > L_max )

```

```
{  
    L_max = length;  
    L_index = i;  
}  
}  
  
// Search right part of best point  
R_index = 0;  
R_max = 0;  
for( i=m_BestIndex; i<m_NumPath; i++ )  
{  
    weight = 1 - (0.02 * abs((m_NumPath-1)/2 - i));  
    diff_x = m_pEdgePos[i].x - m_SrcPos.x;  
    diff_y = m_pEdgePos[i].y - m_SrcPos.y;  
    length = (int)(sqrt( diff_x * diff_x + diff_y * diff_y ) * weight);  
  
    if( length > R_max )  
    {  
        R_max = length;  
        R_index = i;  
    }  
}  
  
// Compare both part  
if( R_max == L_max )  
{  
    if( abs(m_BestIndex - L_index) > abs(m_BestIndex - R_index) )  
    {  
        m_BestPos = m_pEdgePos[R_index];  
        m_BestIndex = R_index;  
    }  
    else  
    {  
        m_BestPos = m_pEdgePos[L_index];  
        m_BestIndex = L_index;  
    }  
}  
else if( R_max > L_max )  
{  
    m_BestPos = m_pEdgePos[R_index];  
    m_BestIndex = R_index;  
}
```

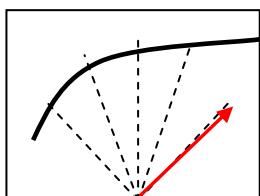
```

else
{
    m_BestPos = m_pEdgePos[L_index];
    m_BestIndex = L_index;
}

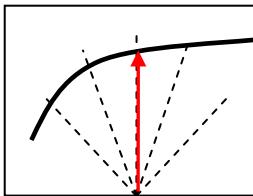
int a;
if( m_BestIndex > 16 )
    a = 0;
}

```

The FindBestPoint function does not simply search for longest path among the various possible paths. Various methods are used for intelligent path finding, and the weight comparison method is one of them. The weight comparison method does not compare all paths with the same conditions, but compares values by assigning more importance to certain values. In the FindBestPoint function, more weight has been given to center path than the left and right edge paths. Using this method, the direction does not have to be changed despite of moderate number of obstacles.

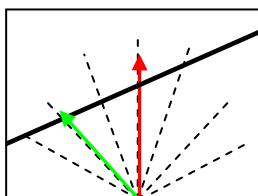


Prior to application of weighting

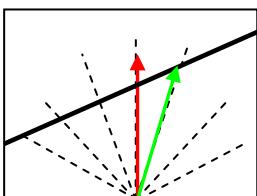


After application of weighting

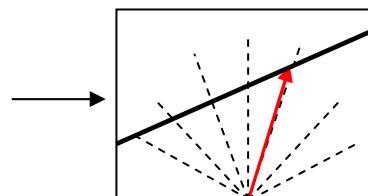
The next method used is the left and right separation comparison method. In the left and right separation comparison method, two initial optimal paths are found from dividing the choices into left and right paths. The optimal path is calculated for each side respectively and the best path is the better of the two calculated optimal paths. By using this method, the problem of a unidirectional tendency in a unidirectional search can be eliminated. When the optimal path from left and right are the same, the nearest optimal path from previous optimal path will be selected as the optimal path.



Left Optimal Path



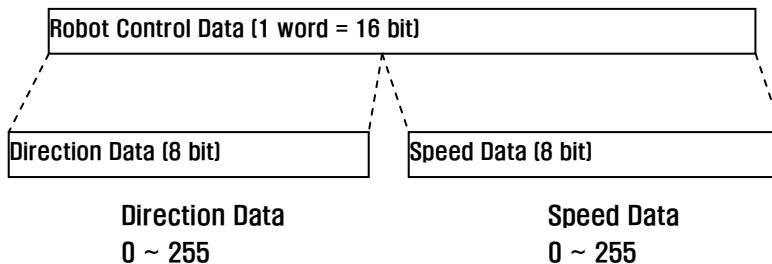
Right Optimal Path



Final Optimal Path

< Controlling robot to follow the optimal path >

After finding the optimal path, the control command is transmitted to the robot. The Zigbee library allows for a single data transmission. In robot control, two kinds of information are needed direction and speed. Protocol denotes a handshake in data transmission between two different systems. From here, a protocol will be selected for the path finder program and obstacle vehicle. This protocol may be used in controlling other robots using the path finder program.



This protocol is defined in the source code as follows. Direction is expressed in 33 increments (0~15 refers to left, 16 refers to center and 17~32 refers to right) and speed is converted in to 255 increments to be transmitted to the robot.

```
#define ROBOT_DIRECTION_LEVEL    33
#define ROBOT_VELOCITY_LEVEL     255
```

The actual source code is embedded in the ControlRobot function.

```
void CPathFinderDlg::ControlRobot()
{
    int iTemp;
    double dTemp, diff_x, diff_y;
    double length1, length2;
    WORD wTemp, wCommand;

    /////////////// Command data format ///////////////////////////////
    // 16 ~ 9bit : direction( 0 ~ 255 )
    // 8 ~ 1bit : velocity( 0 ~ 255 )
    /////////////////////////////// ///////////////////////////////

    // Make direction
    wTemp = (WORD)m_BestIndex;
    wTemp = min(255, wTemp);
    wCommand = (WORD)m_BestIndex << 8;

    // Make velocity
    diff_x = m_pEdgePos[m_BestIndex].x - m_SrcPos.x;
    diff_y = m_pEdgePos[m_BestIndex].y - m_SrcPos.y;
```

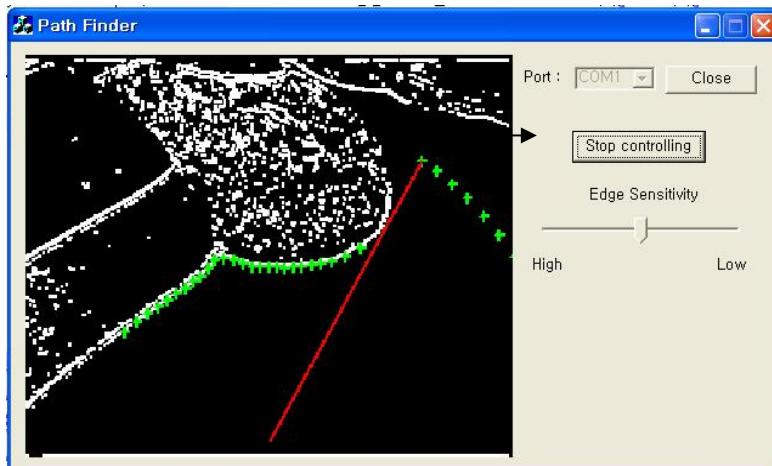
```
length1 = sqrt( diff_x * diff_x + diff_y * diff_y );
length2 = abs(m_pDestPos[(m_NumPath-1)/2].y - m_SrcPos.y);

dTemp = ROBOT_VELOCITY_LEVEL * (length1 / length2);
iTemp = (int)dTemp;
wTemp = (WORD)iTemp;
wTemp = min(255, wTemp);
wCommand = wCommand + wTemp;

// Send command
zigbee_send( wCommand );
}
```

< Displaying Information on the screen >

After conducting all the procedures, the path finder program will output the result on the screen.

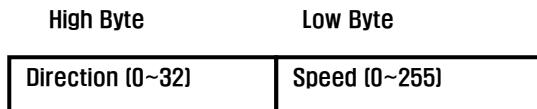


◎ The cliff detector vehicle program analysis

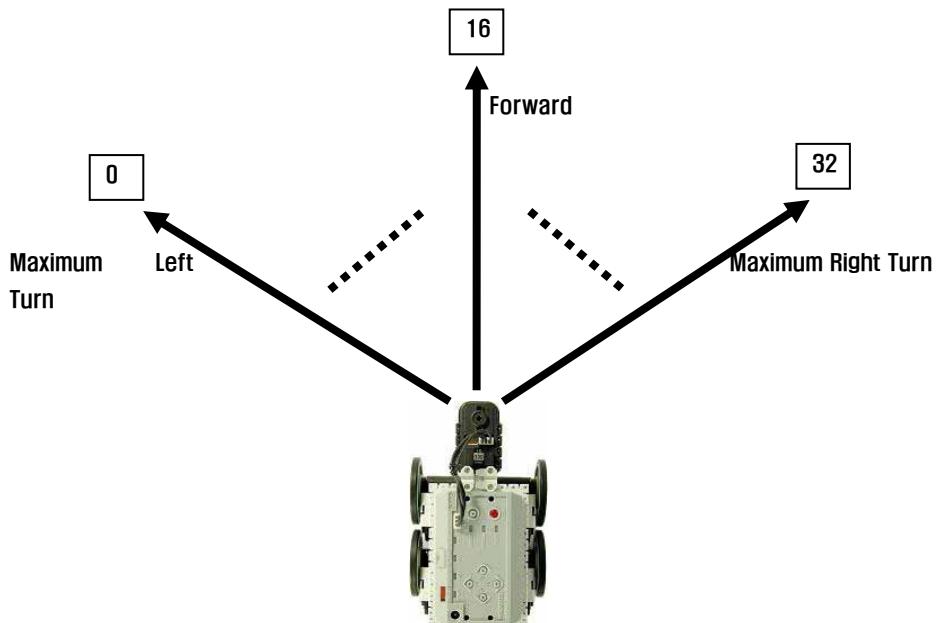
The program for the Cliff Detector Vehicle utilizes C Libraries for a more advantageous control and multiple direction changes rather than a simple forward, left and right direction change. [The Movement Control Program may be utilized. However, the complexity in program design and difficulties in debugging may be an issue when using the control movement program.]

Thus, the C language has been used to generate the Cliff Detector Vehicle program.

After conducting image processing through the PC and the PC determines the direction and speed of the Cliff Detector Vehicle, this information is transmitted to the vehicle wirelessly by using the ZIG-100 module.



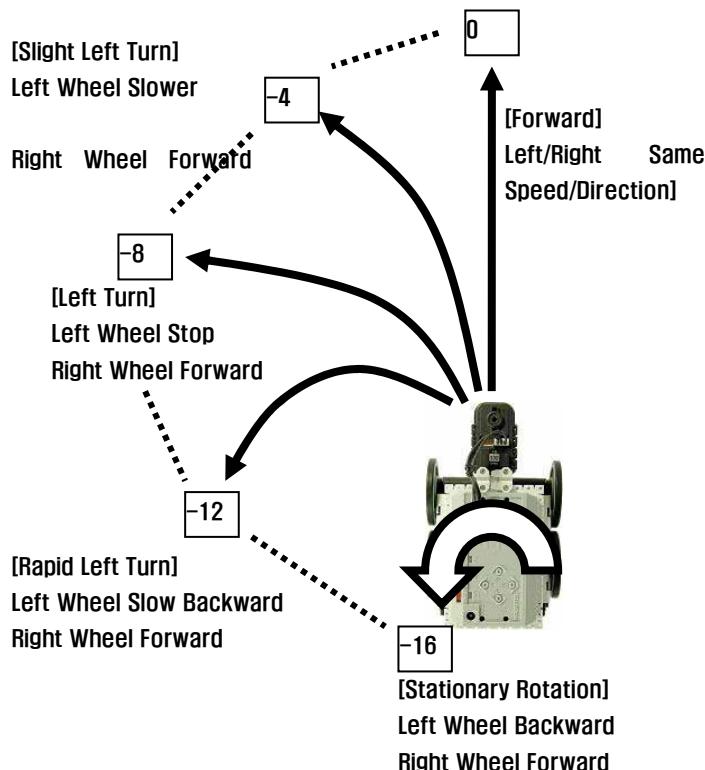
0 refers to Stop and 255 refers to the Maximum Speed in the Speed information and direction information is explained in the below illustration.



After receiving direction data, the unsigned variable is converted into signed variables and 16 is subtracted from the variable. Here, a 0 value would mean forward, while a negative value means a left turn and a positive value refers to a right turn.

To change the direction of the robot (turning), the difference in the speed of left/right wheel has to be controlled. To turn left, the right wheel has to turn fast and left wheel has to turn slow.

Using this method, the robot can perform a zero radius turn or anything in between. These functions are considered in the following example. The value shown in the square box that the arrow points refers to previously explained direction data that had been subtracted with 16.



That is, -8 becomes the center value in determining the rotation of the left wheel. The program has added 8 to the left and subtracted 8 from the right to create this new center value.

Specifically, after determining if the direction data subtracted with 16 is negative or positive [Deciding Left/Right direction], 8 is added to negative value [Left] and a forward rotation results for positive values and a backward rotation for negative values, and 8 is subtracted from positive value [Right] and a negative value results in forward rotation and positive values a backward turn.

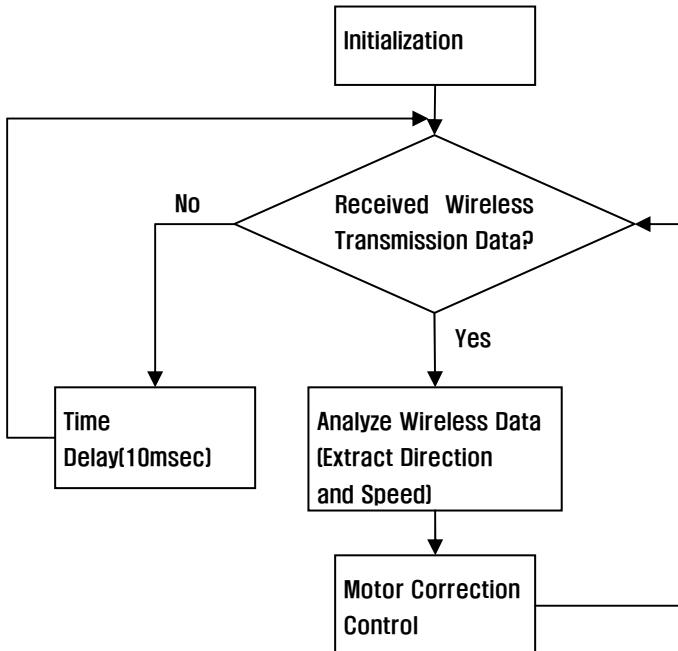
Note that the terms “backward and forward” were used in relation to the above figure. In reality, the left and right motors are connected symmetrically. Therefore, when making a left turn,

the left motors turn forwards, and also the right motors turn forwards. [Take special precaution on this matter].

A right turn can be explained to have similar structure.

<Main routine>

The algorithm is as follows.



As you can see, the main routine has very simple structure.

To eliminate the hassle in downloading the bioloid.hex program prior to initiating charging, the initialization function will automatically initiate charging. By connecting the SMPS adapter, the POWER LED will blink automatically and charging will begin.

The source code corresponding to flow chart utilizing the CM-5 C library is as follows.

The location of the source code is as follows.

“Bioloid SDK \ CM-5 \ app \ example_cliffdetectioncar \ example_cliffdetectioncar.c”

```
int main(void)
{
    PortInitialize();
    TimerInitialize(TIMER0, 128, 1); //Interrupt Enable
    SerialInitialize(SERIAL_PORT0, 1, RX_INTERRUPT); //485
    SerialInitialize(SERIAL_PORT1, DEFAULT_BAUD_RATE, RX_INTERRUPT); //232
    ChargeInInterruptInitialize();
    sei();
}
```

```
TxDString("\r\n CM-5 Example--Remote Controlled Cliff Detection Car with Vision
System\r\n");

gbBatteryChargeMode = BATTERY_CHARGE_MODE_ING; // Charge Enable(AutoCharge)
RXD_ZIGBEE;

word wDirection, wSpeed;

MotorInitialize();

while(1)
{
    ZigbeePacketCheck();

    if( gbNewPacket )      // new Packet found
    {
        wSpeed = (gwZigbeeRxData & 0x00FF);           // Packet analysis
        wDirection = (word)gwZigbeeRxData >> 8;

        MoveCar( wDirection, wSpeed );
        gbNewPacket = 0; // new packet flag clear
    }
    else
    {
        MiliSec(10);
    }
}

return 0;
}
```

ZIG-100 wireless data is transmitted as a word (16bit) and has to be divided into two parameters. The high byte can be calculated by using a shift operation eight times and the low byte can be calculated by conducting an AND operation with 0x00FF.

Motor initiation refers to changing the Counter-Clockwise Angle Limit of AX-12 motors with ID 1~4 to 0 in order to change the motors in to Endless Turn mode.

```
void MotorInitialize(void)
{
    if( ReadWord(1,P_CCW_ANGLE_LIMIT_L) != 0 ) WriteWord(1, P_CCW_ANGLE_LIMIT_L, 0 );
```

```
// Making AX-12 to Endless turn mode  
if( ReadWord(2,P_CCW_ANGLE_LIMIT_L) != 0 ) WriteWord(2, P_CCW_ANGLE_LIMIT_L, 0 );  
if( ReadWord(3,P_CCW_ANGLE_LIMIT_L) != 0 ) WriteWord(3, P_CCW_ANGLE_LIMIT_L, 0 );  
if( ReadWord(4,P_CCW_ANGLE_LIMIT_L) != 0 ) WriteWord(4, P_CCW_ANGLE_LIMIT_L, 0 );  
}
```

To control the motor speed in Endless Turn mode, the Goal Speed has to be modified. To stop the motor, simply use 0 as the Goal Speed Value. The following shows the stop routine.

```
void Stop(void)  
{  
    WriteWord( 1, P_GOAL_SPEED_L, 0 );      // All AX-12s stop  
    WriteWord( 2, P_GOAL_SPEED_L, 0 );  
    WriteWord( 3, P_GOAL_SPEED_L, 0 );  
    WriteWord( 4, P_GOAL_SPEED_L, 0 );  
}
```

The MoveCar function controls the wheel speed by receiving wireless data that includes direction and speed information. The direction value (0~32) and speed value (0~255) are inputted as parameters and the MoveCar function uses the previously explained algorithm to control the motors.

```
#define      COMPLEMENT_SPEED(x)  (1024+(x))  
  
void MoveCar(word Direction, word Speed)  
{  
    word wLeftSpeed=0, wRightSpeed=0, wUnitSpeed;  
    int nDirection, nDetailedDirection;  
  
    wUnitSpeed = (Speed*3)/8;  
  
    if( Speed == 0 ) Stop();  
    else  
    {  
        nDirection = (int)Direction - 16;  
  
        if( nDirection < 0 ) // Left  
        {  
            wRightSpeed = COMPLEMENT_SPEED(256 + Speed*3);      //inverse-round  
            nDetailedDirection = nDirection + 8;  
        }  
    }  
}
```

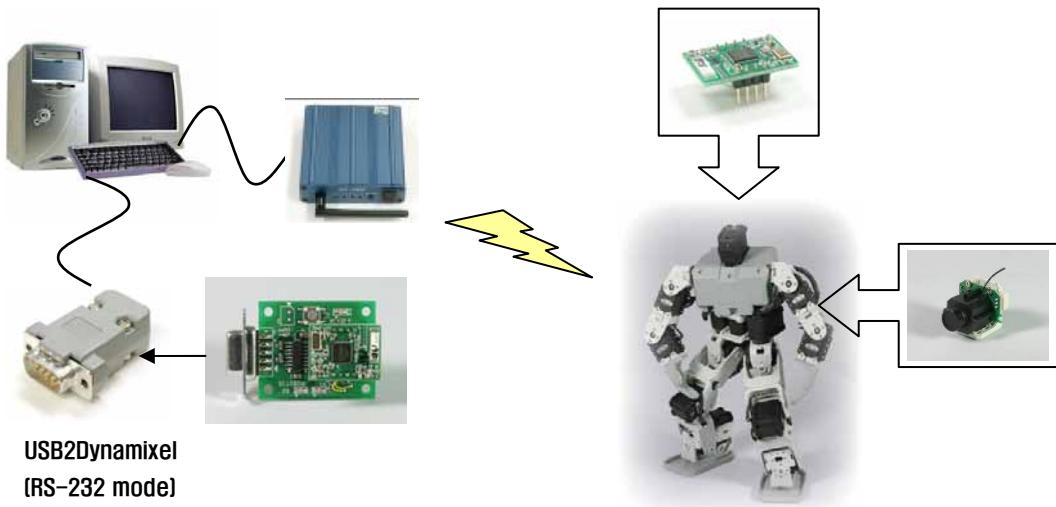
```
if( nDetailedDirection < 0 ) //inverse-round
{
    wLeftSpeed = 256 + wUnitSpeed*(word)(nDetailedDirection*(-1));
    wLeftSpeed = COMPLEMENT_SPEED(wLeftSpeed);
}
else
if( nDetailedDirection > 0 ) //round
{
    wLeftSpeed = 256 + wUnitSpeed*(word)(nDetailedDirection);
}
else
if( nDetailedDirection == 0 )//stop
    wLeftSpeed = 0;
}
else        // Right
{
    wLeftSpeed = 256 + Speed*3;  //round
    nDetailedDirection = nDirection - 8;

    if( nDetailedDirection < 0 ) //inverse-round
    {
        wRightSpeed = 256 + wUnitSpeed*(word)(nDetailedDirection*(-1));
        wRightSpeed = COMPLEMENT_SPEED(wRightSpeed);
    }
    else
    if( nDetailedDirection > 0 ) //round
    {
        wRightSpeed = 256 + wUnitSpeed*(word)(nDetailedDirection);
    }
    else
    if( nDetailedDirection == 0 )//stop
        wRightSpeed = 0;
}

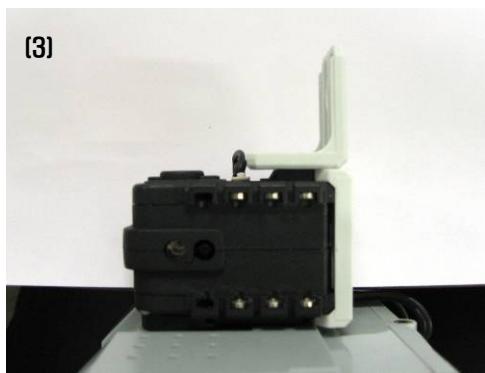
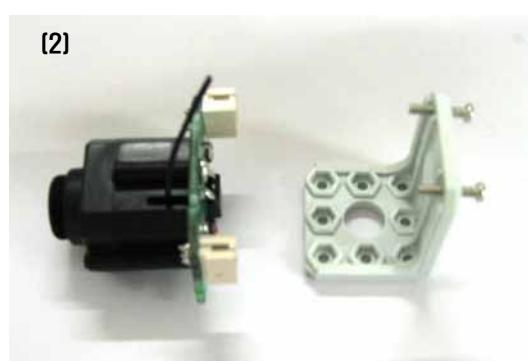
WriteWord( 1, P_GOAL_SPEED_L, wRightSpeed ); // Front-Right Wheel
WriteWord( 2, P_GOAL_SPEED_L, wLeftSpeed ); // Front-Left Wheel
WriteWord( 3, P_GOAL_SPEED_L, wRightSpeed ); // Rear-Right Wheel
WriteWord( 4, P_GOAL_SPEED_L, wLeftSpeed ); // Rear-Left Wheel
}
}
```

3 – 8. Image Recognition and Object Tracking with the Humanoid Robot

In this chapter, the Humanoid, an advanced example shown in the Bioloid Manual (Comprehensive Kit) will be used. The Humanoid used here, shall be equipped with a Zig-100 unit and wireless image transmitter. In addition, the Zig-100 unit shall be set for 1:1 communication mode.



◆ How to install a camera



◆ Program download

- Remote control

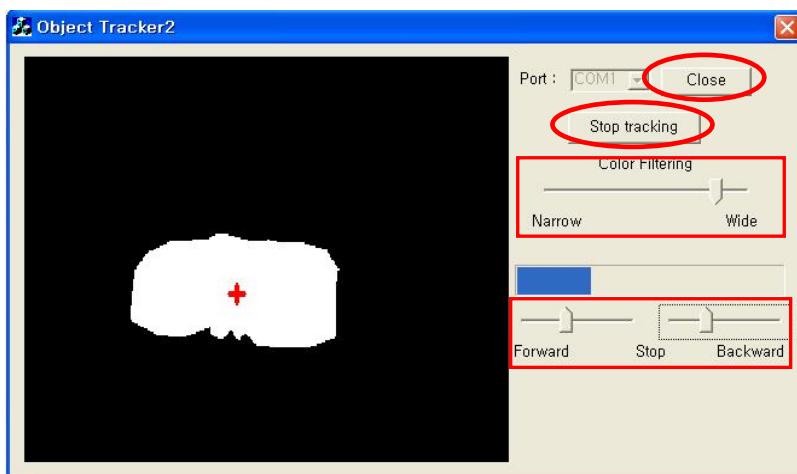
Use “Bioloid SDK \ Windows \ bin \ Object Tracker2.exe”

- Humanoid

Use “Applied Robots \ Advanced \ Humanoid \ DemoExample(Humanoid).mtn” of basic CD and “Bioloid SDK \ Windows \ bin \ example_humanoid2.bpg” of expansion CD.

◆ Robot Operation

- Execute a behavior control program befitting the CM-5 of the Humanoid robot to play mode.
- Execute Object Tracker2.exe.
- Configure communication befitting when Zig2Serial B/D is connected to the COM port of PC
- Start the remote control pressing the “Start tracking” button.
- Create an optimal environment by controlling color filtering and action level.
- When the Humanoid robot detects a reddish object, it will respond depending on the object’s movement. When the reddish object is far, it will go after it and if near, it will step back. Moreover, when the reddish object moves left or right, it will follow the same movement.



○ Analyzing Object Tracker2 program

First, using Visual C++ 6.0, open the Bioloid SDK \Windows \app \Object Tracker2\Object Tracker2.dsw” file from the expansion CD.

Background knowledge in MFC Programming is required for analyzing this example. Refer to reference books on MFC programming for more information.

The Object Tracker2 program is similar to the Object Tracker program shown in the “3-6, Image Recognition Object Tracking Pan/Tilt Camera Robot.” Therefore, topics explained in the previous chapters will not be covered in this chapter and only new information will be provided. Source code can be found in Object Tracker2Dig.h and Object Tracker2Dig.cpp.

Object Tracker and other main routines are embedded in the ImageProcess function.

```
void ImageProcess( unsigned char* pImage, DWORD dwSize )
{
    CDC *pDC;
    CBitmap *pBitmap;
    CRect rtArea;
    VISION_STATE vision_state;
    int object_x, object_y;
    long object_size;
    double percent;

    // Initialize
    pDC = pThis->m_Static_Video.GetDC();
    vision_state = vision_get_state();

    // Processing
    if( pThis->m_bTracking == TRUE )
    {
        RGB24toHSI( pThis->m_pHSI, pImage, vision_state.width, vision_state.height );

        FindRedFromHSI( pThis->m_pBinary, pThis->m_Slider_Range.GetPos()
                        ,pThis->m_pHSI, vision_state.width, vision_state.height );

        Erosion( pThis->m_pTemp, pThis->m_pBinary, vision_state.width,
                 vision_state.height );

        GetCenterPos( &object_x, &object_y, pThis->m_pTemp, vision_state.width,
                      vision_state.height );

        GetShapePiont( pThis->m_pShapePoint, NUMBER_CONTROL_POINT, object_x, object_y
                      ,pThis->m_pTemp, vision_state.width, vision_state.height );

        ClearImage( pThis->m_pTemp, vision_state.width, vision_state.height );

        DrawContour( pThis->m_pShapePoint, NUMBER_CONTROL_POINT
                    ,pThis->m_pTemp, vision_state.width, vision_state.height );

        Dilation( pThis->m_pBinary, pThis->m_pTemp, vision_state.width,
                  vision_state.height );

        FloodFill( object_x, object_y, pThis->m_pBinary, vision_state.width,
                   vision_state.height );
    }
}
```

```
    object_size = GetCenterPos( &object_x, &object_y, pThis->m_pBinary,
vision_state.width, vision_state.height );

    percent = 100 * (double)object_size/(double)(vision_state.width *
vision_state.height);
    pThis->m_Progress_Size.SetPos( (int)percent );
}

// Control Robot
if( pThis->m_bTracking == TRUE )
{
    if( percent > MIN_OBJECT_SIZE )
    {
        if( percent <= pThis->m_Slider_Lower.GetPos() )
            zigbee_send( BIT_SW_UP );
        else if( percent >= pThis->m_Slider_Upper.GetPos() )
            zigbee_send( BIT_SW_DN );
        else
        {
            if( object_x <= (int)(vision_state.width/5) )
                zigbee_send( BIT_SW_LF );
            else if( object_x >= (int)(vision_state.width/5 * 4) )
                zigbee_send( BIT_SW_RT );
            else
                zigbee_send( BIT_SW_START );
        }
    }
}

// Converting bitmap
if( pThis->m_bTracking == TRUE )
    pBitmap = CBitmap::FromHandle( vision_GraytoDDB( pDC->m_hDC, pThis-
>m_pBinary ) );
else
    pBitmap = CBitmap::FromHandle( vision_RGB24toDDB( pDC->m_hDC, pImage ) );

// Drawing bitmap
if( pThis->m_bTracking == TRUE )
{
    CDC MemDC;
    CPen Pen;
    CRect rtArea;
    CBitmap *pOldBitmap;
```

```
// Ready
MemDC.CreateCompatibleDC( pDC );
pOldBitmap = MemDC.SelectObject( pBitmap );

// Drawing object cross
Pen.CreatePen( PS_SOLID, 3, RGB(255,0,0) ); // Red color
MemDC.SelectObject( Pen );
MemDC.MoveTo( object_x, vision_state.height - object_y-5 );
MemDC.LineTo( object_x, vision_state.height - object_y+5 );
MemDC.MoveTo( object_x-5, vision_state.height - object_y );
MemDC.LineTo( object_x+5, vision_state.height - object_y );

MemDC.SelectObject( pOldBitmap );
Pen.DeleteObject();
MemDC.DeleteDC();
}

// Update screen
pThis->m_Static_Video.GetClientRect( rtArea );
vision_DrawDDB( pDC->m_hDC, 0, 0, rtArea.Width(), rtArea.Height(),
(HBITMAP)*pBitmap );
pThis->InvalidateRect( NULL );

// Release
pBitmap->DeleteObject();
pDC->DeleteDC();
}
```

Let's look at the procedures involved in extracting reddish objects from an image and how to determine the size and location of the object. The source code is provided as follows.

```
// Processing
if( pThis->m_bTracking == TRUE )
{
    RGB24toHSI( pThis->m_pHSI, pImage, vision_state.width, vision_state.height );

    FindRedFromHSI( pThis->m_pBinary, pThis->m_Slider_Range.GetPos()
                    ,pThis->m_pHSI, vision_state.width, vision_state.height );

    Erosion( pThis->m_pTemp, pThis->m_pBinary, vision_state.width,
vision_state.height );
```

```
    GetCenterPos( &object_x, &object_y, pThis->m_pTemp, vision_state.width,
vision_state.height );

    GetShapePoint( pThis->m_pShapePoint, NUMBER_CONTROL_POINT, object_x, object_y
                  ,pThis->m_pTemp, vision_state.width, vision_state.height );

    ClearImage( pThis->m_pTemp, vision_state.width, vision_state.height );

    DrawContour( pThis->m_pShapePoint, NUMBER_CONTROL_POINT
                  ,pThis->m_pTemp, vision_state.width, vision_state.height );

    Dilation( pThis->m_pBinary, pThis->m_pTemp, vision_state.width,
vision_state.height );

    FloodFill( object_x, object_y, pThis->m_pBinary, vision_state.width,
vision_state.height );

    object_size = GetCenterPos( &object_x, &object_y, pThis->m_pBinary,
vision_state.width, vision_state.height );
    percent = 100 * (double)object_size/(double)(vision_state.width *
vision_state.height);
    pThis->m_Progress_Size.SetPos( (int)percent );
}
```

< Converting an RGB format image to HSI format >

Refer to Chapter 3-6 for detailed information on the RGB and HIS color models. The main reason for utilizing the HSI color model is to minimize the effect of brightness. Here, the RGB24toHSI function is in charge of conversion.

```
void RGB24toHSI( unsigned char* pHsi, unsigned char* pRGB, int width, int height )
{
    float R,G,B, H,S,I;
    float Min, Angle;
    int iH;
    int x, y, index;

    for( y=0; y<height; y++ )
    {
        for( x=0; x<width; x++ )
        {
```

```
index = width*y + x;

B = (float)pRGB[3*index]/255.0f;
G = (float)pRGB[3*index+1]/255.0f;
R = (float)pRGB[3*index+2]/255.0f;

Min = min(min(R,B),G);

I = (R + G + B)/3.0f;

if( (R == G) && (G == B) )
{
    S = 0.0f;
    H = 0.0f;
}
else
{
    S = 1.0f - (3.0f / (R + G + B)) * Min;

    Angle = (float)((R - G*0.5f - B*0.5f) / sqrt((R-G) * (R-G) + (R-B) *
(G-B)));
    H = (float)acos(Angle) * 57.29577951f;
    if( B > G )
        H = 360.0f - H;
}

iH = (int)(H * 255.0f / 360.0f);
pHSI[3*index] = (unsigned char)iH;
pHSI[3*index+1] = (unsigned char)(S * 255.0f);
pHSI[3*index+2] = (unsigned char)(I * 255.0f);
}
```

< Finding a reddish pixel >

This function was explained in the Chapter 3-6. The actual source code is as follows.

```
void FindRedFromHSI( unsigned char* pBinary, int range, unsigned char* pHSI, int width, int height )
{
    int x, y, index;
    int H, S;
    int radius;

    radius = range / 2;
    for( y=0; y<height; y++ )
    {
        for( x=0; x<width; x++ )
        {
            index = y*width + x;

            H = (int)pHSI[3*index];
            S = (int)pHSI[3*index+1];

            // Red: (0 < H < 30), (H > 225), (80 < S < 175), (I = don't care)
            // Green: (60 < H < 120), (80 < S < 175), (I = don't care)
            // Blue: (140 < H < 200), (80 < S < 175), (I = don't care)
            if( S > 80 && S < 175 )
            {
                if( H <= radius || H >= (255-radius) )
                    pBinary[index] = 255;
                else
                    pBinary[index] = 0;
            }
            else
                pBinary[index] = 0;
        }
    }
}
```

< Eliminating noise >

The Erosion operation method was explained in the Chapter 3-7. The Erosion method is commonly used for eliminating noises. Generally, Erosion and Dilation is used together. However, only the Erosion, not the Dilation, operation will be used here.

```
void Erosion( unsigned char* pResult, unsigned char* pSource, int width, int height )
```

```
{  
    int x, y, index;  
    int ma_x, ma_y, ma_index;  
    int sum = 0;  
  
    // Set filter  
    int szMask_x = 2, szMask_y = 2; // size of mask  
    int threshold = 1;  
  
    for( y=0; y<height; y++ )  
    {  
        for( x=0; x<width; x++ )  
        {  
            index = width*y + x;  
  
            if( y > (height-szMask_y) || x > (width-szMask_x) )  
                pResult[index] = 0;  
            else  
            {  
                // Mask process  
                sum = 0;  
                for( ma_y=0; ma_y<szMask_y; ma_y++ )  
                {  
                    for( ma_x=0; ma_x<szMask_x; ma_x++ )  
                    {  
                        ma_index = width*(y+ma_y) + (x+ma_x);  
                        if( pSource[ma_index] == 0 )  
                            sum++;  
                    }  
                }  
  
                // Determine value  
                if( sum >= threshold )  
                    pResult[index] = 0;  
                else  
                    pResult[index] = 255;  
            }  
        }  
    }  
}
```

< Finding the center coordinate among reddish pixels >

This function is embedded in the GetCenterPos function. There is slight difference with the GetCenterPos function explained in the Chapter 3-6. The GetCenterPos function in this example not only calculates Centroid among the pixels but also identifies the number of pixels as well. The number of pixels is very useful information for determining the dimension of the object.

```
long GetCenterPos( int* pX, int* pY, unsigned char* pBinary, int width, int height )
{
    int x, y, index;
    int count=0;
    long sum_x=0, sum_y=0;

    // Using the centroid method
    for( y=0; y<height; y++ )
    {
        for( x=0; x<width; x++ )
        {
            index = y*width + x;

            if( pBinary[index] == 255 )
            {
                sum_x += x;
                sum_y += y;
                count++;
            }
        }
    }

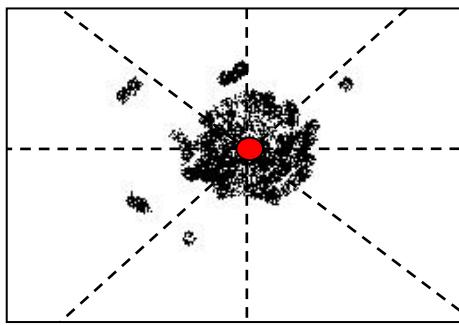
    if( count > 0 )
    {
        *pX = (int)(sum_x / count);
        *pY = (int)(sum_y / count);
    }
    else
    {
        *pX = (int)(width / 2);
        *pY = (int)(height / 2);
    }

    return count;
}
```

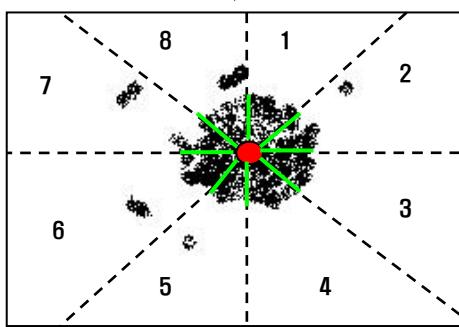
< Extracting Information on the shape of the reddish object >

In this example, not only is the location important, but also the dimension of the reddish object. However, we can confirm that many reddish pixels were lost in the conversion of the image. To accurately calculate the dimension of the object, information on the shape of the object and its internal dimensions must be known.

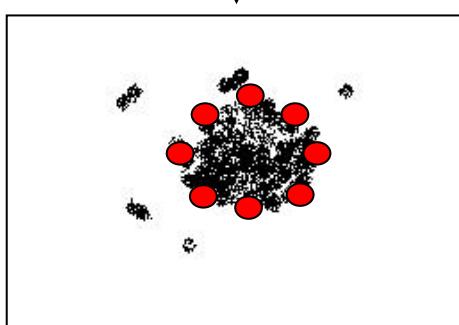
The algorithm is as follows.



Select an initial position and quantity of unique points for searching. The number of unique points must be a multiple of 8. (For example, 8, 16, 24...)



Divide the image in to sectors and find unique points.
To reduce error, the edge of the object is determined by the number of pixels with background colors.



Return coordinates of unique points found.

The following source code is written to simulate the above mentioned theory.

```
void GetShapePoint( POINT* pPoint, int NumPoint, int sx, int sy, unsigned char*
pBinary, int width, int height )
{
#define NUMBER_BACKGROUND_PIXEL 15

    int ix, iy, index, i;
    int count, unit;
    double a, radian, angle, diff_angle, x, y;

    // initialize
    unit = (int)(NumPoint/8);
    diff_angle = 45 / unit;
    radian = 3.141592 / 180.0;

    // search first part of right-upper area
    for( i=0; i<unit; i++ )
    {
        // find gradient of axis
        angle = (90 - diff_angle*i);
        if( angle == 90 )
            a = 0;
        else
            a = tan( angle * radian );

        // search pixel
        iy = sy;
        count = 0;
        while(1)
        {
            if( iy < height && count < NUMBER_BACKGROUND_PIXEL )
            {
                if( a == 0 )
                    x = sx;
                else
                    x = 1/a * (iy - sy) + sx;
                ix = max(0, min(width-1, (int)x));

                index = iy*width + ix;
                if( pBinary[index] == 0 )
                    count++;
            }
        }
    }
}
```

```
        else
            count = 0;

            iy++;
        }

        else if( count >= NUMBER_BACKGROUND_PIXEL )
        {
            iy -= NUMBER_BACKGROUND_PIXEL;
            if( a == 0 )
                x = sx;
            else
                x = 1/a * (iy - sy) + sx;
            ix = max(0, min(width-1, (int)x));

            pPoint[i].x = ix;
            pPoint[i].y = iy;
            break;
        }

        else if( iy >= height )
        {
            iy -= count;
            if( a == 0 )
                x = sx;
            else
                x = 1/a * (iy - sy) + sx;
            ix = max(0, min(width-1, (int)x));

            pPoint[i].x = ix;
            pPoint[i].y = iy;
            break;
        }
    }

}

// search second part of right-upper area
for( i=0; i<unit; i++ )
{
    // find gradient of axis
    angle = (45 - diff_angle*i);
    a = tan( angle * radian );

    // search pixel
```

```
ix = sx;
count = 0;
while(1)
{
    if( ix < width && count < NUMBER_BACKGROUND_PIXEL )
    {
        y = a * (ix - sx) + sy;
        iy = max(0, min(height, (int)y));

        index = iy*width + ix;
        if( pBinary[index] == 0 )
            count++;
        else
            count = 0;

        ix++;
    }
    else if( count >= NUMBER_BACKGROUND_PIXEL )
    {
        ix -= NUMBER_BACKGROUND_PIXEL;
        y = a * (ix - sx) + sy;
        iy = max(0, min(height, (int)y));

        pPoint[unit + i].x = ix;
        pPoint[unit + i].y = iy;
        break;
    }
    else if( ix >= width )
    {
        ix -= count;
        y = a * (ix - sx) + sy;
        iy = max(0, min(height, (int)y));

        pPoint[unit + i].x = ix;
        pPoint[unit + i].y = iy;
        break;
    }
}

// search first part of right-lower area
for( i=0; i<unit; i++ )
```

```
{  
    // find gradient of axis  
    angle = -(diff_angle*i);  
    a = tan( angle * radian );  
  
    // search pixel  
    ix = sx;  
    count = 0;  
    while(1)  
    {  
        if( ix < width && count < NUMBER_BACKGROUND_PIXEL )  
        {  
            y = a * (ix - sx) + sy;  
            iy = max(0, min(height, (int)y));  
  
            index = iy*width + ix;  
            if( pBinary[index] == 0 )  
                count++;  
            else  
                count = 0;  
  
            ix++;  
        }  
        else if( count >= NUMBER_BACKGROUND_PIXEL )  
        {  
            ix -= NUMBER_BACKGROUND_PIXEL;  
            y = a * (ix - sx) + sy;  
            iy = max(0, min(height, (int)y));  
  
            pPoint[unit*2 + i].x = ix;  
            pPoint[unit*2 + i].y = iy;  
            break;  
        }  
        else if( ix >= width )  
        {  
            ix -= count;  
            y = a * (ix - sx) + sy;  
            iy = max(0, min(height, (int)y));  
  
            pPoint[unit*2 + i].x = ix;  
            pPoint[unit*2 + i].y = iy;  
            break;  
        }  
    }  
}
```

```
        }

    }

// search second part of right-lower area
for( i=0; i<unit; i++ )
{
    // find gradient of axis
    angle = -(45 + diff_angle*i);
    a = tan( angle * radian );

    // search pixel
    iy = sy;
    count = 0;
    while(1)
    {
        if( iy >= 0 && count < NUMBER_BACKGROUND_PIXEL )
        {
            x = 1/a * (iy - sy) + sx;
            ix = max(0, min(width-1, (int)x));

            index = iy*width + ix;
            if( pBinary[index] == 0 )
                count++;
            else
                count = 0;

            iy--;
        }
        else if( count >= NUMBER_BACKGROUND_PIXEL )
        {
            iy += NUMBER_BACKGROUND_PIXEL;
            x = 1/a * (iy - sy) + sx;
            ix = max(0, min(width-1, (int)x));

            pPoint[unit*3 + i].x = ix;
            pPoint[unit*3 + i].y = iy;
            break;
        }
        else if( iy < 0 )
        {
            iy += count;
        }
    }
}
```

```
x = 1/a * (iy - sy) + sx;
ix = max(0, min(width-1, (int)x));

pPoint[unit*3 + i].x = ix;
pPoint[unit*3 + i].y = iy;
break;
}
}
}

// search first part of left-lower area
for( i=0; i<unit; i++ )
{
    // find gradient of axis
    angle = -(90 + diff_angle*i);
    if( angle == -90 )
        a = 0;
    else
        a = tan( angle * radian );

    // search pixel
    iy = sy;
    count = 0;
    while(1)
    {
        if( iy >= 0 && count < NUMBER_BACKGROUND_PIXEL )
        {
            if( a == 0 )
                x = sx;
            else
                x = 1/a * (iy - sy) + sx;
            ix = max(0, min(width-1, (int)x));

            index = iy*width + ix;
            if( pBinary[index] == 0 )
                count++;
            else
                count = 0;

            iy--;
        }
        else if( count >= NUMBER_BACKGROUND_PIXEL )
    }
```

```
{  
    iy += NUMBER_BACKGROUND_PIXEL;  
    if( a == 0 )  
        x = sx;  
    else  
        x = 1/a * (iy - sy) + sx;  
    ix = max(0, min(width-1, (int)x));  
  
    pPoint[unit*4 + i].x = ix;  
    pPoint[unit*4 + i].y = iy;  
    break;  
}  
else if( iy < 0 )  
{  
    iy += count;  
    if( a == 0 )  
        x = sx;  
    else  
        x = 1/a * (iy - sy) + sx;  
    ix = max(0, min(width-1, (int)x));  
  
    pPoint[unit*4 + i].x = ix;  
    pPoint[unit*4 + i].y = iy;  
    break;  
}  
}  
}  
  
// search second part of left-lower area  
for( i=0; i<unit; i++ )  
{  
    // find gradient of axis  
    angle = -(135 + diff_angle*i);  
    a = tan( angle * radian );  
  
    // search pixel  
    ix = sx;  
    count = 0;  
    while(1)  
    {  
        if( ix >= 0 && count < NUMBER_BACKGROUND_PIXEL )  
        {
```

```
    y = a * (ix - sx) + sy;
    iy = max(0, min(height, (int)y));

    index = iy*width + ix;
    if( pBinary[index] == 0 )
        count++;
    else
        count = 0;

    ix--;
}
else if( count >= NUMBER_BACKGROUND_PIXEL )
{
    ix += NUMBER_BACKGROUND_PIXEL;
    y = a * (ix - sx) + sy;
    iy = max(0, min(height, (int)y));

    pPoint[unit*5 + i].x = ix;
    pPoint[unit*5 + i].y = iy;
    break;
}
else if( ix < 0 )
{
    ix += count;
    y = a * (ix - sx) + sy;
    iy = max(0, min(height, (int)y));

    pPoint[unit*5 + i].x = ix;
    pPoint[unit*5 + i].y = iy;
    break;
}
}

// search first part of left-upper area
for( i=0; i<unit; i++ )
{
    // find gradient of axis
    angle = 180 - (diff_angle*i);
    if( angle == 180 )
        a = 0;
    else
```

```
a = tan( angle * radian );

// search pixel
ix = sx;
count = 0;
while(1)
{
    if( ix >= 0 && count < NUMBER_BACKGROUND_PIXEL )
    {
        y = a * (ix - sx) + sy;
        iy = max(0, min(height, (int)y));

        index = iy*width + ix;
        if( pBinary[index] == 0 )
            count++;
        else
            count = 0;

        ix--;
    }
    else if( count >= NUMBER_BACKGROUND_PIXEL )
    {
        ix += NUMBER_BACKGROUND_PIXEL;
        y = a * (ix - sx) + sy;
        iy = max(0, min(height, (int)y));

        pPoint[unit*6 + i].x = ix;
        pPoint[unit*6 + i].y = iy;
        break;
    }
    else if( ix < 0 )
    {
        ix += count;
        y = a * (ix - sx) + sy;
        iy = max(0, min(height, (int)y));

        pPoint[unit*6 + i].x = ix;
        pPoint[unit*6 + i].y = iy;
        break;
    }
}
```

```
// search second part of left-upper area
for( i=0; i<unit; i++ )
{
    // find gradient of axis
    angle = 135 - (diff_angle*i);
    a = tan( angle * radian );

    // search pixel
    iy = sy;
    count = 0;
    while(1)
    {
        if( iy < height && count < NUMBER_BACKGROUND_PIXEL )
        {
            x = 1/a * (iy - sy) + sx;
            ix = max(0, min(width-1, (int)x));

            index = iy*width + ix;
            if( pBinary[index] == 0 )
                count++;
            else
                count = 0;

            iy++;
        }
        else if( count >= NUMBER_BACKGROUND_PIXEL )
        {
            iy -= NUMBER_BACKGROUND_PIXEL;
            x = 1/a * (iy - sy) + sx;
            ix = max(0, min(width-1, (int)x));

            pPoint[unit*7 + i].x = ix;
            pPoint[unit*7 + i].y = iy;
            break;
        }
        else if( iy >= height )
        {
            iy -= count;
            x = 1/a * (iy - sy) + sx;
            ix = max(0, min(width-1, (int)x));
        }
    }
}
```

```
    pPoint[unit*7 + i].x = ix;
    pPoint[unit*7 + i].y = iy;
    break;
}
}
}
}
```

< Determining the location and dimension information of the object >

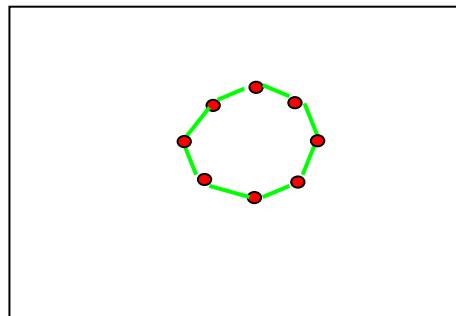
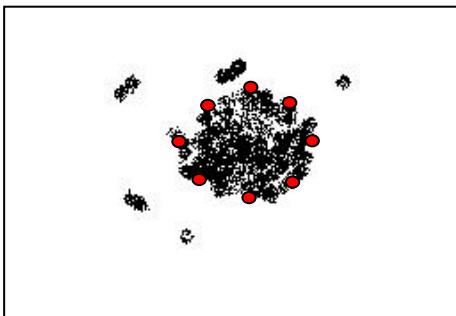
Knowing the shape information you can create an outline of the object using the found edge points. Using the image created, the center position and number of pixels can be determined using GetCenterPos. First, draw an outline of the object using the unique points acquired from the GetShapePoint function.

Prior to drawing the outline, clear the image to be used. The ClearImage function is used for this operation.

```
void ClearImage( unsigned char* pBinary, int width, int height )
{
    int x, y, index;

    for( y=0; y<height; y++ )
    {
        for( x=0; x<width; x++ )
        {
            index = y*width + x;
            pBinary[index] = 0;
        }
    }
}
```

The following shows the DrawContour function used to draw an outline of the object by using the unique points. By inputting the coordinates of the unique points of the object into the function, a polygonal shaped object will be drawn in the given image.



```
void DrawContour( POINT* pCtrlPoint, int NumPoint, unsigned char* pBinary, int width, int height )
{
    int ix, iy, index, i, sx, sy, ex, ey;
    double x, y, dx, dy, a;

    for( i=0; i<NumPoint; i++ )
    {
        // set start position and end position
        sx = pCtrlPoint[i].x;
        sy = pCtrlPoint[i].y;
        if( i == (NumPoint-1) )
        {
            ex = pCtrlPoint[0].x;
            ey = pCtrlPoint[0].y;
        }
        else
        {
            ex = pCtrlPoint[i+1].x;
            ey = pCtrlPoint[i+1].y;
        }

        // find x-difference and y-difference
        dx = ex - sx;
        dy = ey - sy;

        // find gradient of axis
```

```
if( dx != 0 )
    a = dy / dx;

if( abs((int)dx) > abs((int)dy) ) // x based searching
{
    if( dx > 0 )
    {
        for( ix=sx; ix<=ex; ix++ )
        {
            y = a*(ix - sx) + sy;
            iy = (int)max(0, min(height, y));

            index = iy*width + ix;
            pBinary[index] = 255;
        }
    }
    else
    {
        for( ix=sx; ix>=ex; ix-- )
        {
            y = a*(ix - sx) + sy;
            iy = (int)max(0, min(height, y));

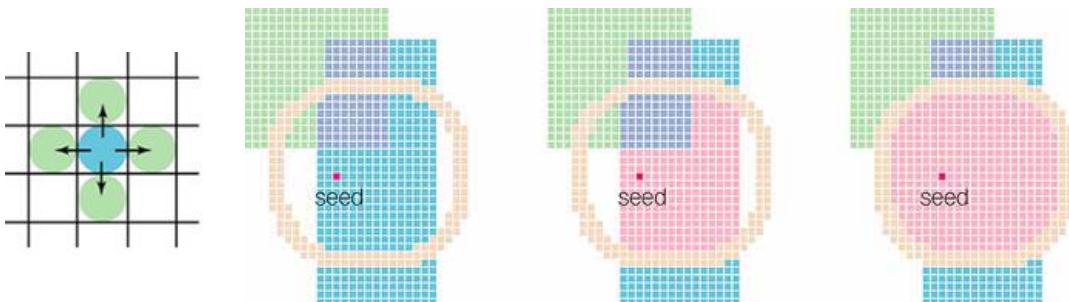
            index = iy*width + ix;
            pBinary[index] = 255;
        }
    }
}
else // y based searching
{
    if( dy > 0 )
    {
        for( iy=sy; iy<=ey; iy++ )
        {
            if( dx == 0 )
                x = sx;
            else
                x = (1/a)*(iy - sy) + sx;
            ix = (int)max(0, min(width, x));

            index = iy*width + ix;
            pBinary[index] = 255;
        }
    }
}
```

```
        }
    }
else
{
    for( iy=sy; iy>=ey; iy-- )
    {
        if( dx == 0 )
            x = sx;
        else
            x = (1/a)*(iy - sy) + sx;
        ix = (int)max(0, min(width, x));

        index = iy*width + ix;
        pBinary[index] = 255;
    }
}
}
```

What we have here is a polygonal shaped outline without the core. To calculate the dimension, internal parts of the polygon need to be filled. The FloodFill algorithm is generally used to fill in the internal area. There are various algorithms for FloodFill. The 4 direction seed fill method is adopted in this example.



To fill in the internal part of the polygon, the outline has to be made thicker. This is done by the Dilation operation. Refer to the Chapter 3-7 for the Dilation operation.

```
void Dilation( unsigned char* pResult, unsigned char* pSource, int width, int
height )
{
    int x, y, index;
    int ma_x, ma_y, ma_index;
```

```
int sum = 0;

// Set filter
int szMask_x = 2, szMask_y = 2; // size of mask
int threshold = 1;

for( y=0; y<height; y++ )
{
    for( x=0; x<width; x++ )
    {
        index = width*y + x;

        if( y > (height-szMask_y) || x > (width-szMask_x) )
            pResult[index] = 0;
        else
        {
            // Mask process
            sum = 0;
            for( ma_y=0; ma_y<szMask_y; ma_y++ )
            {
                for( ma_x=0; ma_x<szMask_x; ma_x++ )
                {
                    ma_index = width*(y+ma_y) + (x+ma_x);
                    if( pSource[ma_index] == 255 )
                        sum++;
                }
            }

            // Determine value
            if( sum >= threshold )
                pResult[index] = 255;
            else
                pResult[index] = 0;
        }
    }
}
```

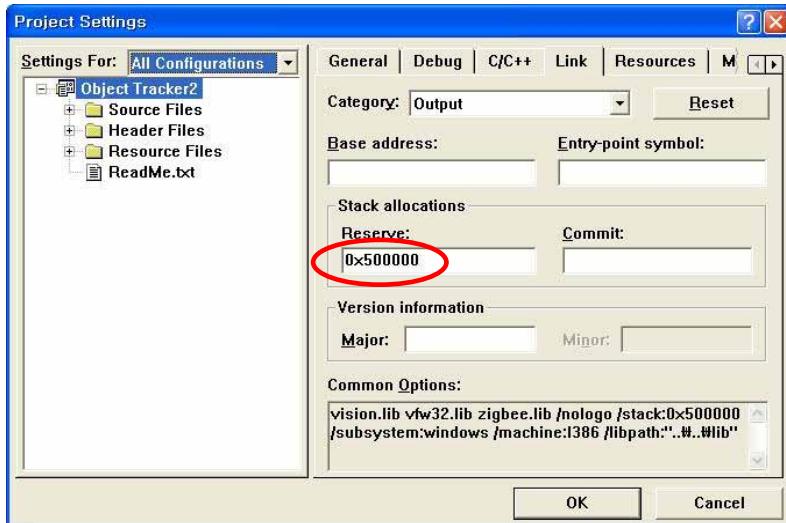
The following is the source code of the FloodFill function.

```
void FloodFill( int x, int y, unsigned char* pBinary, int width, int height )
{
    int index;

    if( (x >= 0) && (x < width) && (y >= 0) && (y < height) )
    {
        index = y*width + x;
        if( pBinary[index] == 0 ) // If present pixel is not filled
        {
            pBinary[index] = 255;

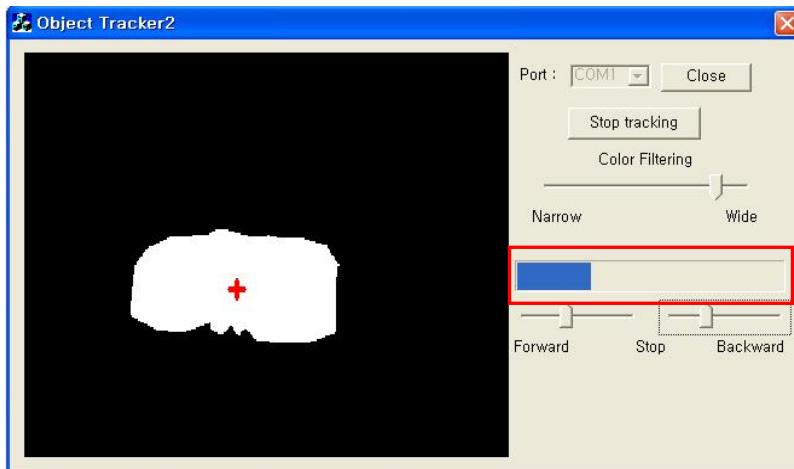
            // Recursive call
            FloodFill( x+1, y, pBinary, width, height );
            FloodFill( x-1, y, pBinary, width, height );
            FloodFill( x, y+1, pBinary, width, height );
            FloodFill( x, y-1, pBinary, width, height );
        }
    }
}
```

The FloodFill function has to be carefully used since Stack memory Overflow may occur using the Recursive call method. To solve this problem, the memory space of the stack must be increased during program compilation. This example has increased the stack allocation from 1MB to 5MB. For using visual images larger than the 320x240 pixels, more than 5MB of stack memory is recommended for allocation.



< Calculating the location and dimension of the restored object >

The GetCenterPos function is used to calculate the location and dimension. The dimension of the object is closely related to its distance. That is, larger sized objects can be thought of as close by, while smaller sized objects can be considered further away. This example represents the dimension in Percentage(%). This is the ratio of the object with respect to the size of the entire image. This value is shown on the Progress Bar in the program.



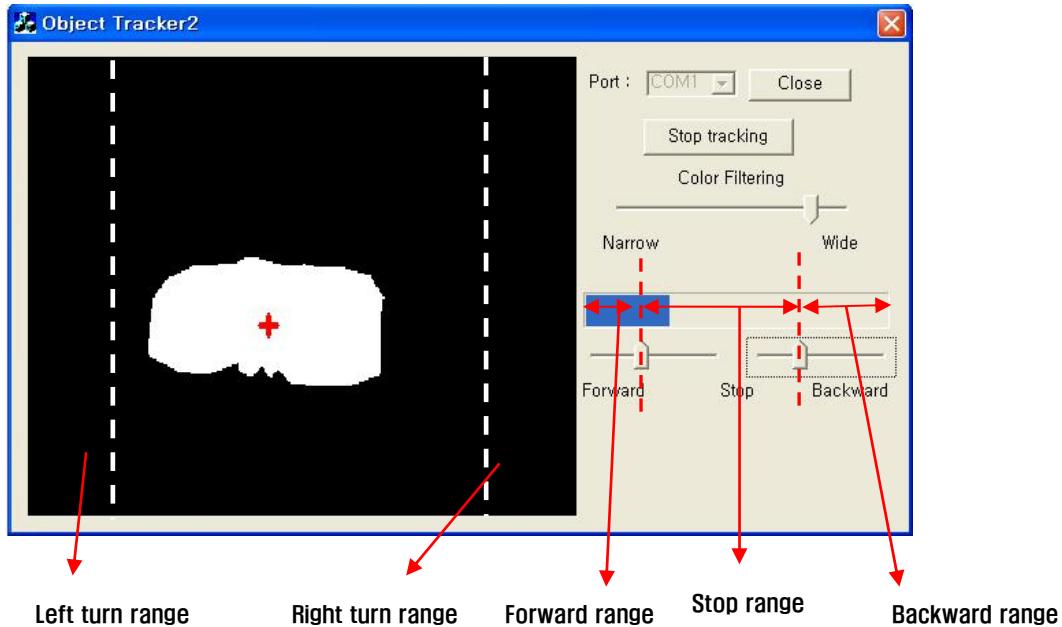
The source code is as follows.

```
object_size = GetCenterPos( &object_x, &object_y, pThis->m_pBinary,  
vision_state.width, vision_state.height );  
percent = 100 * (double)object_size/(double)(vision_state.width *  
vision_state.height);  
pThis->m_Progress_Size.SetPos( (int)percent );
```

< Controlling Humanoid robot >

The output information from Object Tracker2 program is the coordinate and dimension of the object. This will be used to control the Humanoid robot. The movement control program utilized in this chapter, “EG-Humanoid 2.bpg” is similar to “EG-Humanoid.bpg” utilized in Chapter 3-4. The difference between these two is the reception of **START Button command** on the CM-5 that stops the motion.

For the Humanoid robot to detect reddish Objects, the Object Tracker2 Program will transmit five commands to the robot.



Situation	Command	Communication data
Object's area is in forward range.	Forward	CM-5 U Button
Object's area is in backward range.	Backward	CM-5 D Button
Object's area is in stop range and the position is in left turn range.	Left turn	CM-5 L Button
Object's area is in stop range and the position is in right turn range.	Right turn	CM-5 R Button
Above situations are not applicable.	Stop	CM-5 Start Button

The source code is as follows.

```
// CM-5 button state
#define BIT_SW_RT      0x01
#define BIT_SW_LF      0x02
#define BIT_SW_DN      0x04
#define BIT_SW_UP      0x08
#define BIT_SW_START    0x10
```

```
// Control Robot
if( pThis->m_bTracking == TRUE )
{
    if( percent > MIN_OBJECT_SIZE )
```

```
{  
    if( percent <= pThis->m_Slider_Lower.GetPos() )  
        zigbee_send( BIT_SW_UP );  
    else if( percent >= pThis->m_Slider_Upper.GetPos() )  
        zigbee_send( BIT_SW_DN );  
    else  
    {  
        if( object_x <= (int)(vision_state.width/5) )  
            zigbee_send( BIT_SW_LF );  
        else if( object_x >= (int)(vision_state.width/5 * 4) )  
            zigbee_send( BIT_SW_RT );  
        else  
            zigbee_send( BIT_SW_START );  
    }  
}  
}
```

4 . APPENDIX

4 – 1 . LIST OF THE CM-5 LIBRARY FUNCTIONS

4 – 1 – 1 . LED LIBRARY

LEDOn

A function that turns on the 7 LEDs of the CM-5.

```
void LEDOn (
    byte bLed // LED value that you want to turn ON
);
```

Parameters

bLed

An LED's number that you want to turn on. Since it is a bitwise operation, just input the appropriate LED value. If you want to turn on two or more LEDs, use the OR operation. Each LED is defined as shown below.

Value	Meaning
BIT_LED_PWR	POWER LED
BIT_LED_TXD	TXD LED
BIT_LED_RXD	RXD LED
BIT_LED_AUX	AUX LED
BIT_LED_MANA	MANAGE LED
BIT_LED_PROG	PROGRAM LED
BIT_LED_PLAY	PLAY LED
BIT_LED_ALL	ALL LED

Remarks

None

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
LEDOn ( BIT_LED_AUX | BIT_LED_PLAY ); // Turn on AUX, PLAY LED
```

LEDOFF

A function that turns off the 7 LEDs of the CM-5.

```
void LEDOFF (
    byte bLed    // LED value that you want to turn OFF
);
```

Parameters

bLed

An LED's number that you want to turn off. Since it is a bitwise operation, just input the appropriate LED value. If you want to turn off two or more LEDs, use the OR operation. Each LED is defined as shown below.

Value	Meaning
BIT_LED_PWR	POWER LED
BIT_LED_TXD	TXD LED
BIT_LED_RXD	RXD LED
BIT_LED_AUX	AUX LED
BIT_LED_MANA	MANAGE LED
BIT_LED_PROG	PROGRAM LED
BIT_LED_PLAY	PLAY LED
BIT_LED_ALL	ALL LED

Remarks

None

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
LEDOFF ( BIT_LED_AUX | BIT_LED_PLAY );           // Turn off AUX, PLAY LED.
```

4 - 1 - 2 . Button Library

GetButtonState

A function that reads the button state of the CM-5.

```
byte GetButtonState (void);
```

Return Value

Return the current button state as a byte.

The values as shown below return combined as bits.

Value	Meaning
BIT_SW_RT	Right direction button
BIT_SW_LF	Left direction button
BIT_SW_DN	Downward direction button
BIT_SW_UP	Upward direction button
BIT_SW_START	Start button

Remarks

There are macros that have applied the following GetButtonState buttons. Since it returns a value of 1 when the button is pressed, you can use this to check the value of a particular button.

```
#define PUSH_SW_START      ( GetButtonState() & BIT_SW_START )
#define PUSH_SW_UP          ( GetButtonState() & BIT_SW_UP )
#define PUSH_SW_DN          ( GetButtonState() & BIT_SW_DN )
#define PUSH_SW_LF          ( GetButtonState() & BIT_SW_LF )
#define PUSH_SW_RT          ( GetButtonState() & BIT_SW_RT )
```

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
byte bkey;
bkey = GetButtonState();      // Reads the current button state.
switch( bkey )
{
    caseBIT_SW_START:    { } break;
    caseBIT_SW_UP:       { } break;
    caseBIT_SW_DN:       { } break;
    caseBIT_SW_LF:       { } break;
    caseBIT_SW_RT:       { } break;
    case0 :              { }
}
```

WaitButtonChange

A function that waits for a button change and then returns the value of the changed button state of the CM-5.

```
byte WaitButtonChange (void);
```

Return Value

Returns the current button state as a byte type.

The values as shown below return combined as bits.

Value	Meaning
BIT_SW_RT	Right direction button
BIT_SW_LF	Left direction button
BIT_SW_DN	Downward direction button
BIT_SW_UP	Upward direction button
BIT_SW_START	Start button

Remarks

None

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
byte bkey;
bkey = WaitButtonChange(); // Waits until the input of a button.
switch( bkey ) {
    case BIT_SW_START:    { } break;
    case BIT_SW_UP:       { } break;
    case BIT_SW_DN:       { } break;
    case BIT_SW_LF:       { } break;
    case BIT_SW_RT:       { }
}
```

4 - 1 - 3. UART0 (DYNAMIXEL SERIAL COMMUNICATION) LIBRARY

TxD485

A function that transfers 1 byte of data to the HDBRT (Half-Duplex Buffer Rx/Tx) line through the UART0 of the CM-5.

```
void TxD485 (
    byte bTxData    // Tx Data that you want to Transmit on HDBRT
);
```

Parameters

bTxData

Take note if the transferred data is a “byte” type.

Remarks

Take note of the difference with the TxD0Byte function. In order to transfer 1 byte data, this function has to be used.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
TxD485 ( 255 );      // From HDBRT line 0xFF (255) send 1 byte.
```

TxD0Byte

A function that transfers 1 byte of data through the UART 0 of the CM-5.

```
void TxD0Byte (
    byte bTxdData      // Tx Data that you want to Transmit on UART 0
);
```

Parameters

bTxdData

Take note if the transfer data is a “byte” type.

Remarks

Take note of the difference with the TxD485. Unlike the TxD485 function, the HDBRT bus is not activated. Therefore, data used in the TxD0Byte function can not be received by the Dynamixel unit.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
TxD0Byte ( 255 );           // From UART 0 으로 0xFF (255) send 1 byte.
```

RxD0Byte

A function that transfers 1 byte of data through the UART of the CM-5. [Waits until data is received]

```
byte RxD0Byte (void);
```

Return Value

Returns 1 byte of data as a “byte” type.

Remarks

This function receives 1 byte of data through polling method, but since UART 0 generally uses a UART Rx Interrupt, it is rarely used.

Since UART 0 only receives a return packet from the Dynamixel unit, when a Dynamixel control library is used, it automatically checks for the return packet.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

4 - 1 - 4 . Dynamixel Control Library

RxPacket

This function receives a return packet from the Dynamixel unit through the HDBRT(Half-Duplex Buffer Rx/Tx) of the CM-5. It includes basic functions such as a timeout, header, ID, packet length, and checksum error capability.

```
byte RxPacket (
    byte bRxLength           // the expected Packet Length to receive
);
```

Parameters

bRxLength

The length of a received packet. A return packet of a generic Write command is 6 bytes.

Return Value

Returns the length of a return packet. Returns 0 when an error occurs.

Remarks

RxPacket is a function that is made to be used with the TxPacket function. A packet received through this function is saved in the gbpRxBuffer array. Therefore, after the RxPacket function is executed, you can read and check the specific location value of the gbpRxBuffer array.

When the variable of gbErrorPrint is set to 1, the contents of error are sent to the UART so you can check which error occurred.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
// When reading the Firmware Version of ID 2 Dynamixel
byte bID, bTxPacketLength, bRxPacketLength;
bID = 2;
gbpParameter[0] = P_VERSION;           //Address of Firmware Version
gbpParameter[1] = 1;                   //Read Length
```

```
bTxPacketLength = TxPacket(bID,INST_READ,2);  
bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1]);  
TxDString( "\r\n TxD:" );  
PrintBuffer(gbpTxBuffer,bTxPacketLength); //FF FF 02 04 02 02 01 F4  
TxDString( "\r\n RxD:" );  
PrintBuffer(gbpRxBuffer,bRxPacketLength); //FF FF 02 03 00 16 E4  
TxDString( "\r\n Firmware Version:" );  
TxDByte10( gbpRxBuffer[5] ); //0x16 (22)
```

TxPacket

This function transfers an instruction packet to the Dynamixel unit through the HDBRT(Half-Duplex Buffer Rx/Tx) of the CM-5. After the required parameters are set, it automatically calculates the checksum, packet header, and others, and then transfers the data.

```
byte TxPacket (
    byte bID,           // Dynamixel ID
    byte bInstruction, // Instruction byte
    byte bParameterLength // Length of Parameter
);
```

Parameters

bID

The ID of Dynamixel that sends a packet.

bInstruction

Instructions you can use. As shown below, many instructions are possible.

Value	Meaning
INST_PING	Used to check the existence of a Dynamixel unit that has a particular ID
INST_READ	A command that reads data from the Control Table of a Dynamixel unit
INST_WRITE	A command that writes data to the Control Table of a Dynamixel unit
INST_REG_WRITE	Refer to registered instruction related in the Dynamixel manual
INST_ACTION	Refer to action related in the Dynamixel manual. Execute REG_WRITE content
INST_RESET	Initialize the Control Table of Dynamixel to the Factory Default setting
INST_SYNC_WRITE	With one packet, simultaneously control multiple Dynamixels

For more details on each instruction, refer to the user manual of the Dynamixel units.

bParameterLength

The parameter length of a packet.

Return Value

Returns the total packet length that has been sent.

Remarks

A TxPacket is a function made to use the RxPacket function. By using this function, you can transfer a packet to the Dynamixel units. As such, write a parameter in the gbpParameter array and insert the length of the parameter to the TxPacket as a parameter.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
// When reading the Firmware Version of ID 2 Dynamixel
byte bID, bTxPacketLength, bRxPacketLength;
bID = 2;
gbpParameter[0] = P_VERSION;           //Address of Firmware Version
gbpParameter[1] = 1;                   //Read Length
bTxPacketLength = TxPacket(bID,INST_READ,2);
bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1]);
TxDString("\r\n TxD:");
PrintBuffer(gbpTxBuffer,bTxPacketLength);      //FF FF 02 04 02 02 01 F4
TxDString("\r\n RxD:");
PrintBuffer(gbpRxBuffer,bRxPacketLength);      //FF FF 02 03 00 16 E4
TxDString("\r\n Firmware Version:");
TxDByte10( gbpRxBuffer[5] );                //0x16 (22)
```

PrintBuffer

A function that outputs the contents of the memory buffer on the terminal screen (UART). It is printed as hexadecimal.

```
void PrintBuffer (
    byte *bpPrintBuffer, // Buffer Pointer that you want to print
    byte bLength        // Length of Buffer Array
);
```

Parameters

*bpPrintBuffer

Pointer of a buffer that will be printed (array name).

bLength

Length of the buffer that will be printed (only has to print out the received /sent amount)

Remarks

PrintBuffer is a function that is useful in debugging since a developer can directly check to see if a packet has been generated or when they want to confirm the contents of a packet.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
// When reading the Firmware Version of ID 2 Dynamixel
byte bID, bTxPacketLength, bRxPacketLength;
bID = 2;
gbpParameter[0] = P_VERSION;           //Address of Firmware Version
gbpParameter[1] = 1;                   //Read Length
bTxPacketLength = TxPacket(bID, INST_READ, 2);
bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1]);
TxDString("\r\n TxD:");
PrintBuffer(gbpTxBuffer,bTxPacketLength);      //FF FF 02 04 02 02 01 F4
TxDString("\r\n RxD:");
PrintBuffer(gbpRxBuffer,bRxPacketLength);      //FF FF 02 03 00 16 E4
TxDString("\r\n Firmware Version:"); TxDByte10( gbpRxBuffer[5] ); //0x16 (22)
```

TxRxPacket

A function that combines TxPacket and RxPacket. Its usage method is the same as the TxPacket.

```
byte TxRxPacket (
    byte bID,           // Dynamixel ID
    byte bInst,          // Instruction Byte
    byte bTxParaLen     // Length of Parameter
);
```

Parameters

bID

The ID of Dynamixel that sends a packet.

bInstruction

Instructions you can use. As shown below, many instructions are possible. For more details on each instruction, refer to the user manual of the Dynamixel units.

Value	Meaning
INST_PING	Used to check the existence of Dynamixel that has a particular ID
INST_READ	A command that reads data from Control Table of Dynamixel
INST_WRITE	A command that writes data to Control Table of Dynamixel
INST_REG_WRITE	Refer to registered instruction related in Dynamixel manual
INST_ACTION	Refer to action related in Dynamixel manual. Execute REG_WRITE content
INST_RESET	Initialize Control Table of Dynamixel to Factory Default
INST_SYNC_WRITE	With one time packet, simultaneously control multiple Dynamixels.

bParameterLength

The parameter length of a packet.

Return Value

Returns 0 if there are errors in the return packet and returns 1 if there are no errors.

Remarks

TxRxPacket is a function that combines the function of TxPacket and RxPacket, reducing the hassle of executing the RxPacket after TxPacket. When the gbErrorPrint variable is set to 1, the content of error is sent to UART and you can check the source of the error. When the gbPacketPrint variable is set to 1, it sends the contents of the buffer that has been sent and received.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
// When reading the Firmware Version of ID 2 Dynamixel
byte bID, bError;
bID = 2;
gbParameter[0] = P_VERSION;           //Address of Firmware Version
gbParameter[1] = 1;                  //Read Length
bError = TxRxPacket( bID, INST_READ, 2 );
if( bError == 0 ) { /* Error */ }
```

ReadByte

A function where you can read 1 byte data from the control table of Dynamixel.

```
byte ReadByte (
    byte bID,          // Dynamixel ID
    byte bAddress      // Address of Control Table
);
```

Parameters

bID

The ID of the Dynamixel unit you want to read.

bAddress

The address of control table where you want to read.

Return Value

When there is no error, it returns the read value, but for errors, it returns 0xFF (255). So, if you read “255,” you need to check if there is an error. [for example, the existence of the Dynamixel unit, its communication speed, etc.]

Remarks

It is a function where you can read a value from the control table of the Dynamixel unit. It is used when you want to read a byte type. Since it uses the TxRxPacket function internally, if gbPrintPacket or gbErrorPrint value is set to 1, you can output the source or content of the error through the terminal screen.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
// When reading the Firmware Version of ID 2 Dynamixel
ReadByte( 2, P_VERSION );
```

ReadWord

A function where you can read 2 bytes (word type) of data from the control table of the Dynamixel unit.

```
word ReadWord (
    byte bID,      // Dynamixel ID
    byte bAddress  // Address of Control Table
);
```

Parameters

bID

The ID of the Dynamixel unit you want to read.

bAddress

The address of control table where you want to read.

Return Value

When there is no error, it returns the read value, but for errors, it returns 0xFFFF (65535). So, if you read “65535,” you need to check if there is an error. [for example, the existence of the Dynamixel unit, its communication speed, etc.]

Remarks

It is a function where you can read a value from the control table of a Dynamixel unit. It is used when you want to read a word type. Since it uses theTxRxPacket function internally, if gbPrintPacket or gbErrorPrint is set to 1, you can output the source or content of the error through the terminal screen.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
// When reading the current location value of ID 2 Dynamixel
ReadWord( 2, P_PRESENT_POSITION_L );
```

WriteByte

A function where you can write 1 byte data to the control table of a Dynamixel unit.

```
byte WriteByte (
    byte bID,      // Dynamixel ID
    byte bAddress, // Address of Control Table
    byte bData     // New Data
);
```

Parameters

bID

The ID of the Dynamixel unit you want to write to.

bAddress

The address of the control table where you want to write to

bData

Data you want to write

Return Value

When there is no error, it returns 1, but for errors, it returns 0.

Remarks

It is a function where you can write a value to the control table of a Dynamixel unit. It is used when you want to write a byte type. Since it uses the TxRxPacket function internally, if gbPrintPacket or gbErrorPrint is set to 1, you can output the source or content of the error through the terminal screen.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
// When changing ID 2 Dynamixel to ID 1
WriteByte( 2, P_ID, 1 );
```

WriteWord

A function where you can write 2 bytes (word type) data to the control table of a Dynamixel unit

```
byte WriteWord (
    byte bID,      // Dynamixel ID
    byte bAddress, // Address of Control Table
    word wData     // New Data
);
```

Parameters

bID

The ID of the Dynamixel unit you want to write to.

bAddress

The address of the control table where you want to write to.

wData

Data you want to write.

Return Value

When there is no error, it returns 1, but for errors, it returns 0.

Remarks

It is a function where you can write a value to the control table of the Dynamixel unit. It is used when you want to write a word type. Since it uses the TxRxPacket function internally, if gbPrintPacket or gbErrorPrint is set to 1, you can output the source or content of the error through the terminal screen.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
// When changing ID 2 Dynamixel's Goal Position to 512
WriteWord( 2, P_GOAL_POSITION_1, 512 );
```

4-1-5. UART1 (PC SERIAL COMMUNICATION) LIBRARY

TxD1Byte

A function that transfers 1 byte of data through the UART 1 of the CM-5.

```
void TxD1Byte (
    byte bTxdData // Tx Data that you want to Transmit on UART 1
);
```

Parameters

bTxdData

Take note whether the transfer data is a “byte” type or not.

Remarks

UART 1 connects to the PC and outputs through the PC’s serial terminal. It is appropriate for debugging purposes and various libraries are provided so that numbers and strings can be easily displayed.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
TxD1Byte ( 'A' ); // Send 1 byte of character 'A' to UART 1
```

TxDByte16

A function that outputs 1 byte of data as a hexadecimal through the UART 1 of the CM-5.

```
void TxDByte16 (
    byte bSentData      // Tx Data (will print in Hexadecimal)
);
```

Parameters

bSentData

Take note whether the transferred data is a “byte” type or not..

Remarks

Outputs 1 byte of a number as a hexadecimal through the UART 1.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
TxDByte16 ( 65 ); // Outputs number 65 to hexadecimal to the UART 1 -> 41
```

TxDByte10

A function that outputs 1 byte of data as a decimal through the UART 1 of the CM-5.

```
void TxDByte10 (
    byte bByte           // Tx Data (will print in Decimal)
);
```

Parameters

bByte

Take note whether the transferred data is a “byte” type or not. .

Remarks

Outputs 1 byte of a number as a decimal through the UART 1. Empty digit places are filled as 0.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
TxDByte10 ( 0x41 ); // Outputs the number 0x41 as a decimal through UART 1 -> 065
```

TxDWord16

A function that outputs 2 bytes of data as a hexadecimal through the UART 1 of the CM-5.

```
void TxDWord16 (
    word wSentData           // Tx Data (will print in Hexadecimal)
);
```

Parameters

wSentData

Take note whether the transferred data is a “word” type or not.

Remarks

Outputs 2 bytes of a number as a hexadecimal through the UART 1.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
TxDWord16 ( 20000 );// Outputs the number 20000 as a hexadecimal through UART 1 -> 4E20
```

TxDWord10

A function that outputs 2 bytes of data as a decimal through the UART 1 of the CM-5.

```
void TxDWord10 (
    word wData           // Tx Data (will print in Decimal)
);
```

Parameters

wData

Take note whether the transferred data is a “word” type or not.

Remarks

Outputs 2 bytes of a number as a decimal through the UART 1. Empty digit places are filled as 0.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
TxDWord10 ( 512 );//Outputs the number 512 as a decimal through UART 1-> ..512 ("."
is empty)
```

TxDLong16

A function that outputs 2 bytes of data as a hexadecimal through the UART 1 of the CM-5.

```
void TxDLong16 (
    long lSentData           // Tx Data (will print in Hexadecimal)
);
```

Parameters

lSentData

Take note whether the transfer data is a “long” type or not.

Remarks

Outputs 4 bytes of a number as a hexadecimal through the UART 1.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
TxDLong16 ( 0x12345 ); // Outputs the number 0x12345 as a hexadecimal through UART 1
-> 00012345
```

TxDLong10

A function that outputs 4 bytes of data as a decimal through the UART 1 of the CM-5.

```
void TxDLong10 (
    long lLong           // Tx Data (will print in Decimal)
);
```

Parameters

lLong

Take note whether the transferred data is a “long” type or not.

Remarks

Outputs 4 bytes of a number as a decimal through the UART 1. Empty digit places are filled as 0.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
TxDLong10 ( 123456 );// Outputs the number 123456 as a decimal through UART 1 -> 123456
```

TxD8DecHex

A function that outputs 1 byte of data as a decimal and hexadecimal through the UART 1 of the CM-5.

```
void TxD8DecHex (
    byte bByte          // Tx Data (will print in Decimal(Hexadecimal))
);
```

Parameters

bByte

Take note whether the transferred data is a “byte” type or not..

Remarks

Outputs 1 byte of a number as a decimal and hexadecimal through the UART 1.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
TxD8DecHex ( 120 ); // Outputs the number 120 as a decimal (hexadecimal) through UART
1 -> 120(0x78)
```

TxDString

A function that outputs a string through the UART 1 of the CM-5

```
void TxDString (
    byte *bData      // Tx Data (will print in Decimal(Hexadecimal))
);
```

Parameters

*bData

A pointer of the string that will be sent.

Remarks

A function that outputs the string. Its function is similar to “puts” of stdio in C language.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
TxDString ("Hello Robotis"); // Prints string -> Hello Robotis
```

RxD8Interrupt

A function that reads 1 byte of data that is on standby from the Rx Interrupt Buffer of the UART 1 of the CM-5. (It is on standby until data is received.)

```
byte RxD8Interrupt (void);
```

Return Value

Returns 1 byte data as a “byte” type.

Remarks

This function receives 1 byte data by the interrupt method from the UART 1. Since data that is received via the UART 1 is a packet received from either a PC or Zigbee Module, you can use the Zigbee packet check function to automatically check whether new data has arrived.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

4 - 1 - 6. Wireless Data Communication Library

GetHexWord

A function that receives a hexadecimal 4-digits number (2 bytes) through the UART 1 of the CM-5.

```
word GetHexWord ( void );
```

Return Value

Number data of “word” type (2 bytes)

Remarks

A function generally used in ZigbeeSet; receives hexadecimal 4-digit numbers and converts and returns them as 2 byte number data.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

ZigbeeSet

A function that reads or changes the setting of the Zigbee Module.

```
byte ZigbeeSet (
    byte Function,    // Zigbee Setup Function
    word newDest,     // New Destination Zigbee ID Address
    byte newWait      // New Waitmode value
);
```

Parameters

bFunction

The Zigbee setting functions are shown below.

Function Value	Meaning
0	Read Src, Destination, Wait value
1	Change Destination value
2	Change Wait value
3	Change Destination, Wait value

newDest

The destination address that you want to change

newWait

Waitmode Value that you want to change

Return Value

If it finishes without any errors, it returns 1, but if the ZIG-100 module is not found, it returns 0.

Remarks

It reads or changes the setting of the Zigbee. In gwMyZigbeeID, it saves its own Zigbee ID and in gwYourZigbeeID, it saves the ID of the opposing robot.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
////Zigbee setting function example//////////  
// ZigbeeSet(0,0,0); // Read  
// ZigbeeSet(1,0x3243,0); // Set Dest Address  
// ZigbeeSet(2,0,1); // Set Wait Mode  
// ZigbeeSet(3,0x00F8,1); // Set Dest Address, Wait Mode  
//////////  
  
/// Reading the setting of Zigbee, when there is no setting, it outputs error; if  
not, it outputs the address of its own and a counterpart. ///  
if ( ZigbeeSet(0,0,0) == 0 ) TxDString(" Zigbee not Found!");  
else  
{  
    TxDString(" My Address:"); TxDWord10( gwMyZigbeeID );  
    TxDString("Dest Address:"); TxDWord10( gwYourZigbeeID );  
}  
  
/// Change to Broadcasting Mode.  
ZigbeeSet ( 1,0xFFFF,0 );
```

ZigbeePacketCheck

A function that checks whether Zigbee communication data has arrived at the UART 1.

```
void ZigbeePacketCheck ( void );
```

Remarks

By analyzing the UART 1 Interrupt Buffer, it checks whether a Zigbee communication packet has arrived.

When new data arrives, gbNewPacket is changed to 1 and the data is saved to the variable gwZigbeeRxData.

Take note that after reading, gbNewPacket must be initialized back to 0.

When using this function, you have to use the loop statement to check the exact point of packet's arrival. Moreover, for the UART 1 case, since it can receive a packet from either a PC or Zigbee module, the RXD_ZIGBEE function must be performed so that it can receive a packet from the Zigbee module.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
RXD_ZIGBEE;
while(1)
{
    ZigbeePacketCheck ( );

    if( gbNewPacket == 1 )    // New Zigbee Data Arrived
    {
        TxDWord10 (gwZigbeeRxData); // Processing the new data
        gbNewPacket = 0;
    }
}
```

ZigbeePacketTransfer

A function that sends 2 bytes of data through a Zigbee module.

```
void ZigbeePacketTransfer (
    word wData           // New TxD Data
);
```

Parameters

wData

2 bytes of data that will be sent through the Zigbee modlue.

Remarks

It sends 2 bytes of data conforming to the Zigbee data format through the UART 1.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
ZigbeePacketTransfer ( 1000 ); // Transfers '1000' data
```

4 - 1 - 7. Battery Charge Library

ChargeInInterruptInitialize

Initializes the charge routine, and other related resources.

```
void ChargeInInterruptInitialize ( void );
```

Remarks

To charge, initialize timer interrupt, AD converter and various routines.

This function should be, if possible, executed only during the beginning part of a program.

To display the current charge state (charge mode) and to start the charge, the variable gbBatteryChargeMode must be set to BATTERY_CHARGE_MODE_ING.

The following shows the values of gbBatteryChargeMode.

Value	Meaning
BATTERY_CHARGE_MODE_NONE	Initial value. Non-charging state.
BATTERY_CHARGE_MODE_ING	Charging or started to charge.
BATTERY_CHARGE_MODE_ERROR	Error occurred.
BATTERY_CHARGE_MODE_FULL	Charge completed.

Since the charge state automatically updates itself, you only have to initialize the charge in the beginning.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
int main( void )
{
    PortInitialize ();
    TimerInitialize (TIMER0, 128, 1);           //Timer Interrupt Enable
    SerialInitialize (SERIAL_PORT0,1,RX_INTERRUPT); //HDBRT
    SerialInitialize (SERIAL_PORT1,DEFAULT_BAUD_RATE,RX_INTERRUPT); //232
    ChargeInInterruptInitialize ();             // Charge Initialize
    sei();                                     // Enable Interrupt
    gbBatteryChargeMode = BATTERY_CHARGE_MODE_ING; // Battery Charge Start
    . .
    // to do
    . .
    return 0;
}
```

4 - 1 - 8. Other Essential Library

SerialInitialize

Initializes the serial port.

```
void SerialInitialize (
    byte bPort,           // UART Port Number
    byte bBaudrate,       // Baud rate (Division Factor)
    byte bInterrupt       // Interrupt
);
```

Parameter

bPort

Serial port number.

Value	Meaning
SERIAL_PORT0	UART 0 (HDBRT)
SERIAL_PORT1	UART 1 (PC, ZIGBEE)

bBaudrate

It indicates the division factor that calculates baud rate.

That is, “ $bBaudrate = (2000000 / \text{desired baudrate}) - 1$ ”

It is 34 for 57600bps and 1 for 1Mbps.

< Main Communication Speed >		
No.	Actual speed(bps)	Goal speed(bps)
1	1000000	1000000
3	500000	500000
4	400000	400000
7	250000	250000
9	200000	200000
16	117647.1	115200
34	57142.9	57600
103	19230.8	19200
207	9615.4	9600

bInterrupt

Checks whether the interrupt is being used. It can be either 0 or the RX_INTERRUPT value.

Remarks

It is a routine that initializes a serial port and to use the library effectively, it must be executed in the beginning. Normally, the Ax series operates at 1Mbps, and the PC (Zigbee) 57600bps, and thus, UART 0 is generally set to 1Mbps, and UART 1 to 57600bps.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
int main( void )
{
    PortInitialize ();
    TimerInitialize (TIMER0, 128, 1);                                //Timer Interrupt Enable
    SerialInitialize (SERIAL_PORT0,1,RX_INTERRUPT);                      //HDBRT
    SerialInitialize (SERIAL_PORT1,DEFAULT_BAUD_RATE,RX_INTERRUPT);     //232
    ChargeInInterruptInitialize ();                                     // Charge Initialize
    sei();                                                               // Enable Interrupt
    . .
    // to do
    . .
    return 0;
}
```

PortInitialize

Initializes each port of the ATMEGA128.

```
void PortInitialize (void);
```

Remarks

Initializes each port of the ATMEGA128, an MCU that is a core component of the CM-5.
In order to use the library effectively, it must be executed in the beginning.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
int main( void )
{
    PortInitialize ();

    TimerInitialize (TIMER0, 128, 1);           //Timer Interrupt Enable
    SerialInitialize (SERIAL_PORT0,1,RX_INTERRUPT); //HDBRT
    SerialInitialize (SERIAL_PORT1,DEFAULT_BAUD_RATE,RX_INTERRUPT); //232
    ChargeInInterruptInitialize ();             // Charge Initialize
    sei();                                     // Enable Interrupt
    . .
    // to do
    . .
    return 0;
}
```

TimerInitialize

Initializes the timer interrupt routine. Uses Timer 0 (Timer 0 Output Compare Interrupt).

```
byte TimerInitialize (
    byte bWhatTimer,      // Timer number
    word wFreq,           // Frequency
    byte bInterrupt       // Interrupt
);
```

Parameter

bWhatTimer

Indicates what timer to initialize. Make it TIMER0.

wFreq

Indicates frequency of the timer. The optimum frequency is 128 Hz.

bInterrupt

Indicates whether the interrupt is being used. Input 1.

Remarks

Initializes the timer interrupt. On the charge routine or during charge, started either from the power LED control or in the CM-5 library, it is optimized to operate at 128Hz by using Timer 0. Therefore, it must be immediately executed at the beginning of main function and we recommend that you use the following example

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
int main( void )
{
    PortInitialize ();                                //Port Initialize
    TimerInitialize (TIMER0, 128, 1);                //Timer Interrupt Enable
    SerialInitialize (SERIAL_PORT0,1,RX_INTERRUPT);   //HDBRT
    SerialInitialize (SERIAL_PORT1,DEFAULT_BAUD_RATE,RX_INTERRUPT); //232
```

```
ChargeInInterruptInitialize ();           // Charge Initialize
sei();                                     // Enable Interrupt
. . .
// to do
. . .
return 0;
}
```

MiliSec

A function that delays time in a program.

```
void MiliSec (
    word wDelayTime      // Delay time in Milisecond
);
```

Parameter

wDelayTime

The time that will be delayed in units of 1/1000 second, (Millisecond)

Remarks

It is used to as a time delay in a program. The time unit is 1/1000 second (Millisecond) and since it is word type, ranges from 0 to 65535.

Information

- Import library : libCM-5.a
- Header file : libCM-5.h

Example

```
//// Turn on and off AUX LED for 1 second.
LEDOn( BIT_LED_AUX );
MiliSec( 1000 );
LEDOff( BIT_LED_AUX );
```

4 – 2 . The List of Window Library Function

4 – 2 – 1 . Dynamixel Control Library

dxl_instruction

A structure that includes the content of the Instruction packet, a packet used for Dynamixel communication.

```
struct dxl_instruction
{
    BYTE id;                                // Dynamixel ID
    BYTE length;                             // Packet length
    BYTE instruction;                        // Instruction code
    BYTE address;                            // Address of Dynamixel control table
    BYTE parameter[MAX_PARAM_NUM];          // Parameters
};
```

Members

id

The Dynamixel unit ID that will receive the Instruction packet. The range of IDs is 0~253, with 254 being a special ID reserved for broadcasting. The broadcasting ID is declared as a BROADCASTING_ID macro in Dynamixel.h.

length

Instruction packet's number of bytes. Calculating the packet length is as follows.

Length = the number of parameters + 3

However, in case of INST_PING and INST_ACTION, INST_RESET, only the ID and instruction are required, means the length is 2.

instruction

The instruction code used in Dynamixel units. It is declared as a macro in Dynamixel.h.

Value	Meaning
INST_PING	Ping command
INST_READ	Read command
INST_WRITE	Write command
INST_REG_WRITE	Registered writing command
INST_ACTION	Action command
INST_RESET	Reset command (Initialize the value as factory set)
INST_SYNC_WRITE	Synchronous writing command

Address

value of the control table of a Dynamixel unit. It is declared as a macro in Dynamixel.h.

		Value	Meaning
Common EEPROM Range	AX-12	P_MODEL_NUMBER_L	Low byte of model number
		P_MODEL_NUMBER_H	High byte of model number
		P_VERSION	Firmware version information
		P_ID	Dynamixel ID
		P_BAUD_RATE	Dynamixel communication speed (Baud rate)
		P_RETURN_DELAY_TIME	Return delay time
		P_LIMIT_TEMPERATURE	Internal temperature limit
		P_DOWN_LIMIT_VOLTAGE	Minimum voltage limit
		P_UP_LIMIT_VOLTAGE	Maximum voltage limit
		P_RETURN_LEVEL	Return level
AX-S1	AX-12	P_CW_ANGLE_LIMIT_L	Low byte of clockwise angle limit value
		P_CW_ANGLE_LIMIT_H	High byte of clockwise angle limit value
		P_CCW_ANGLE_LIMIT_L	Low byte of counter-clockwise angle limit value
		P_CCW_ANGLE_LIMIT_H	High byte of counter-clockwise angle limit value
		P_MAX_TORQUE_L	High byte of torque limit value
		P_MAX_TORQUE_H	Low byte of torque limit value
		P_ALARM_LED	Alarm LED
		P_ALARM_SHUTDOWN	Alarm shut down
		P_EEPROM_IR_DETECT_COMPARE	IR sensor comparison value
		P_EEPROM_LIGHT_DETECT_COMPARE	Light sensor comparison value
RAM Range	Common	P_PRESENT_VOLTAGE	Present voltage
		P_PRESENT_TEMPERATURE	Present temperature
		P_REGISTERED_INSTRUCTION	Question on instruction register
		P_LOCK	EEPROM lock
	AX-12	P_TORQUE_ENABLE	Torque On/Off
		P_LED	LED On/Off
		P_CW_COMPLIANCE_MARGIN	CW Compliance margin
		P_CCW_COMPLIANCE_MARGIN	CCW Compliance margin
		P_CW_COMPLIANCE_SLOPE	CW Compliance slope
		P_CCW_COMPLIANCE_SLOPE	CCW Compliance slope
		P_GOAL_POSITION_L	Low byte of goal position value
		P_GOAL_POSITION_H	High byte of goal position value
		P_GOAL_SPEED_L	Low byte of goal speed value
		P_GOAL_SPEED_H	High byte of goal speed value
		P_TORQUE_LIMIT_L	Low byte of torque limit value
		P_TORQUE_LIMIT_H	High byte of torque limit value

		Value	Meaning
RAM Range	AX-12	P_PRESENT_POSITION_L	Low byte of present position value
		P_PRESENT_POSITION_H	High byte of present position value
		P_PRESENT_SPEED_L	Low byte of present speed value
		P_PRESENT_SPEED_H	High byte of present speed value
		P_PRESENT_LOAD_L	Low byte of present load value
		P_PRESENT_LOAD_H	High byte of present load value
		P_MOVING	Movement Yes/No
		P_PUNCH_L	Low byte of punch value
		P_PUNCH_H	High byte of punch value
		P_IR_LEFT_FIRE_DATA	Left IR sensor data
RAM Range	AX-S1	P_IR_CENTER_FIRE_DATA	Center IR sensor data
		P_IR_RIGHT_FIRE_DATA	Right IR sensor data
		P_LIGHT_LEFT_DATA	Left light sensor data
		P_LIGHT_CENTER_DATA	Center light sensor data
		P_LIGHT_RIGHT_DATA	Right light sensor data
		P_IR_OBSTACLE_DETECTED	IR sensor Yes/No
		P_LIGHT_DETECTED	Light sensor Yes/No
		P_SOUND_DATA	Sound sensor data
		P_SOUND_DATA_MAX_HOLD	Maximum value of sound sensor data
		P_SOUND_DETECTED_COUNT	Number of sound detection
		P_SOUND_DETECTED_TIME_L	Low byte of sound detected time value
		P_SOUND_DETECTED_TIME_H	High byte of sound detected time value
		P_BUZZER_DATA0	Buzzer data 0
		P_BUZZER_DATA1	Buzzer data 1
		P_REMOTE_ARRIVED	IR remote data arrival Yes/No
		P_REMOTE_RXDATA0	Low byte of IR remote received data
		P_REMOTE_RXDATA1	High byte of IR remote received data
		P_REMOTE_TXDATA0	Low byte of IR remote sent data
		P_REMOTE_TXDATA1	High byte of IR remote sent data
		P_IR_DETECT_COMPARE	IR sensor comparison value of RAM range
		P_LIGHT_DETECT_COMPARE	Light sensor comparison value of RAM r

For more details on control table, refer to the Dynamixel manual.

parameter

An array where the parameter value required for the instruction packet is included. The maximum size of the array in Dynamixel.h is defined as follows.

```
#define MAX_PARAM_NUM      255
```

Remarks

If you want more details on Instruction packets and on each member, check the Dynamixel manual.

Information

- Import library : Dynamixel.lib
- Header file : Dynamixel.h

dxl_status

A structure that includes the content of the Status packet, a packet used for Dynamixel communication.

```
struct dxl_status
{
    BYTE id;                                // Dynamixel ID
    BYTE length;                             // Packet length
    BYTE error;                              // Error code
    BYTE parameter[MAX_PARAM_NUM];          // Parameters
};
```

Members

id

The Dynamixel unit ID that responds to the Instruction packet.

length

Status packet's number of bytes. Calculating packet length is as follows.

Length = the number of parameter + 2

Here, 2 represents the addition of 1 byte for the ID and 1 byte for the Error.

error

Includes information on the Dynamixel units' errors. The error code is declared as a macro in Dynamixel.h.

Value	Meaning
ERROR_NONE	There is no error.
ERROR_VOLTAGE	Error in input voltage.
ERROR_ANGLE	Angle value out of range.
ERROR_OVERHEAT	Temperature value out of range.
ERROR_RANGE	Parameter value out of valid range.
ERROR_CHECKSUM	Error checksum in instruction packet.
ERROR_OVERLOAD	Not able to control current load with the maximum torque.
ERROR_INSTRUCTION	Error in instruction code.

The source for finding the above values is as follows.

```
[ Example ]  
  
if( dxl_status.error == ERROR_NONE )  
    // No error.  
else if( dxl_status.error & ERROR_VOLTAGE )  
    // ERROR_VOLTAGE  
else if( dxl_status.error & ERROR_ANGLE )  
    // ERROR_ANGLE  
else if( dxl_status.error & ERROR_OVERHEAT )  
    // ERROR_OVERHEAT  
else if( dxl_status.error & ERROR_RANGE )  
    // ERROR_RANGE  
else if( dxl_status.error & ERROR_CHECKSUM )  
    // ERROR_CHECKSUM  
else if( dxl_status.error & ERROR_OVERLOAD )  
    // ERROR_OVERLOAD  
else if( dxl_status.error & ERROR_INSTRUCTION )  
    // ERROR_INSTRUCTION
```

parameter

An array where requested data from the Instruction packet is included.

The maximum size of the array in Dynamixel.h is defined as follows.

```
#define MAX_PARAM_NUM    255
```

Remarks

dxl_status is not created by the users. When you create the contents of the Instruction packet and send it to the Dynamixel unit, check to see whether the content of the status packet is included in the dxl_status.

Information

- Import library : Dynamixel.lib
- Header file : Dynamixel.h

dxl_open

A function that initializes USB2Dynamixel.

```
bool dxl_open(  
    char* port_name,      // Port name  
    int baud_number       // Index of baudrate  
) ;
```

Parameters

port_name

A port name of the PC where Dynamixel units are connected. It is used as a string, “COM1”, “COM2”, etc.

baud_number

It is a baud number of the Dynamixel units. Instead of the actual baud speed, an index is used with a range of 1~255.

The calculation method is as follows.

Actual Baud rate = $2000000 / (\text{baud_number} + 1)$

< Main Communication Speed >		
No.	Actual speed(bps)	Goal speed(bps)
1	1000000	1000000
3	500000	500000
4	400000	400000
7	250000	250000
9	200000	200000
16	117647.1	115200
34	57142.9	57600
103	19230.8	19200
207	9615.4	9600

Return Values

If a return value is true, the initialization is successful, if not, it returns false

Remarks

It must call this function in order to control Dynamixel units.

Information

- Import library : Dynamixel.lib
- Header file : Dynamixel.h

dxl_close

A function that terminates USB2Dynamixel.

```
void dxl_close( );
```

Parameters

None

Return Values

None

Remarks

After initializing the USB2Dynamixel using `dxl_open`, this function must be used to terminate it.

Information

- Import library : Dynamixel.lib
- Header file : Dynamixel.h

dxl_control

A function that sends Instruction packets to Dynamixel units and receives Status packets.

```
int dxl_control(  
    dxl_instruction* pinst_packet, // Instruction packet  
    dxl_status* pstatus_packet, // Status packet  
    bool bWait                 // Option of Waiting  
) ;
```

Parameters

pinst_packet

A data input, it is a `dxl_instruction` structure pointer. You directly create the contents of `dxl_instruction`.

pstatus_packet

A data output, it is a `dxl_instruction` structure pointer. Communicating Dynamixel units create the contents of `dxl_status`.

bWait

Communicating Dynamixel units determine whether to wait for the Status packet

Value	Meaning
true	Waits for Status packet.
false	Does not wait for Status packet.

Return Values

Returns error code.

Value	Meaning
ERR_CODE_SUCCESS	Communication success.
ERR_CODE_SEND_FAIL	Could not send instruction packet from PC.
ERR_CODE_PACKET_LOST	Part of Status packet is lost.
ERR_CODE_TIMEOUT	Status packet not delivered.

Remarks

None

Information

- Import library : Dynamixel.lib
- Header file : Dynamixel.h

Other Useful Macros

MAKE_WORD(a, b)

Creates one “word” type with a high and low byte. “a” is a low byte and “b” is a high byte.

[Example]

```
WORD word;
BYTE high, low;

word = MAKE_WORD(low, high);
```

HIGH_BYTE(w)

From the word type data, it only outputs the high byte.

[Example]

```
WORD word;
BYTE high;

high = HIGH_BYTE(word);
```

LOW_BYTE(w)

From the word type data, it only outputs the low byte.

[Example]

```
WORD word;
BYTE high;

low = LOW_BYTE(word);
```

MAKE_VALUE(d, v)

Among the control table of the Dynamixel units, it is used for the data of Speed and Load items.

“d” is the 11th bit value indicating a direction among the word type data, and v is the 10bit value indicating data.

[Example]

```
BYTE direction;  
WORD value;  
WORD data;  
  
data = MAKE_VALUE(direction, value);
```

DIRECTION(w)

Among the control table of the Dynamixel units, it is used for the data of Speed and Load items.

It outputs the 11th bit value from the word type data. Companion macros are as follows.

Value	Meaning
CW_DIRECTION	Clockwise
CCW_DIRECTION	Counter-clockwise

[Example]

```
BYTE direction;  
WORD data;  
  
direction = DIRECTION(data);  
if( direction == CW_DIRECTION )  
    // if clockwise  
else if( direction == CCW_DIRECTION )  
    // if counter-clockwise
```

VALUE(w)

Among the control table of the Dynamixel units, it is used for the data of Speed and Load items.

It outputs low the 10bit value from the word type data..

[Example]

```
WORD value;  
WORD data;  
  
value = VALUE(data);
```

Information

- Import library : Dynamixel.lib
- Header file : Dynamixel.h

4 - 2 - 2. Wireless Data Communication Library

zigbee_open

A function that initializes Zig2Serial.

```
bool zigbee_open(  
    char* port_name,      // Port name  
    HWND hParent,         // Handle of Parent window  
    UINT message          // Using message of communication  
) ;
```

Parameters

port_name

A port name of the PC where the Zigbee module is connected. It is used as a string, “COM1,” “COM2,” etc.

hParent

A Window handle that will receive a data message from the Zigbee library.

message

Message value that will be used for data receipt purposes.

[Example]

```
#define WM_ZIGBEE_RECEIVE (WM_USER + 1)
```

Return Values

If a return value is true, the initialization is successful, if not, it returns false.

Remarks

It must be called in order to do wireless data communication.

Information

- Import library : zigbee.lib
- Header file : zigbee.h

zigbee_close

A function that terminates the Zigbee library.

```
void zigbee_close( );
```

Parameters

None

Return Values

None

Remarks

After initializing the Zig2Serial using zigbee_open, you must use this function to terminate it.

Information

- Import library : zigbee.lib
- Header file : zigbee.h

zigbee_clear

A function that empties the buffer in the Zigbee library.

```
void zigbee_clear( );
```

Parameters

None

Return Values

If a return value is true, the initialization is successful, if not, it returns false.

Remarks

None

Information

- Import library : zigbee.lib
- Header file : zigbee.h

zigbee_send

A function that is used when Zigbee communication is performed.

```
bool zigbee_send(  
    WORD send_data      // Send data  
) ;
```

Parameters

send_data

Data that will be sent for Zigbee communication.

Return Values

If a return value is true, the initialization is successful, if not, it returns false.

Remarks

None

Information

- Import library : zigbee.lib
- Header file : zigbee.h

Zigbee Communication Method

WM_USER_MESSAGE

When using the zigbee_open function, you must first declare a data received notice message. When Zigbee data arrives, the Zigbee library notifies the application program with a message. Windows then transfers necessary data to the message parameter.

wParam	Word type Zigbee received data
lParam	Not used

[Example]

```
WORD ReceiveData;  
  
ReceiveData = (WORD)wParam;
```

4 - 2 - 3. Image handling and recognition library

VISION_PARAM

A structure that includes the content used for when initializing a wireless image receiver.

```
struct VISION_PARAM
{
    int width;          // Image width
    int height;         // Image height
    void (*pProcessFunc) (unsigned char*, DWORD );
                           // User function for image processing
};
```

Members

width

The resolution width of a wireless image receiver.

height

The resolution height size of a wireless image receiver

pProcessFunc

It is a user function pointer that will call each frame from the wireless image receiver. When you register a function that will handle images from the user application program with this value, anytime the wireless image receiver operates, it calls each frame. This function transfers captured images as an unsigned char type, and gives the total number of bytes of the image data through the DWORD type.

[Example]

```
void ImageProcess( unsigned char* pImageData, DWORD dwSize );

VISION_PARAM.pProcessFunc = ImageProcess;
```

Remarks

This function must be set when initializing the wireless image receiver as a structure that will be used as a parameter in the vision_open function.

Information

- Import library : vision.lib
- Header file : vision.h

VISION_STATE

A structure that includes the content used when showing the state of a wireless image receiver

```
struct VISION_STATE
{
    int width;           // Image width
    int height;          // Image height
    int fps;             // fps(Frame Per Second)
    float capTime_ms;   // Term of capturing image
    float procTime_ms;  // Term of processing algorithm
};
```

Members

width

The resolution width that is used in a vision library. .

height

The resolution height size that is used in a vision library.

fps

The fps (Frame Per Second) value that represents the data handling speed of the captured image from the vision library.

capTime_ms

A value (in ms) that dictates the update period of captured images from the vision library.

procTime_ms

A value (in ms) that shows the user function handling time of captured images from the vision library.

Remarks

It is used to find out the state of the wireless image receiver using the `vision_get_state` function.

Information

- Import library : `vision.lib`
- Header file : `vision.h`

vision_open

A function that initializes the wireless image receiver

```
bool vision_open(  
    VISION_PARAM* pVisionParam; // Pointer of VISION_PARAM structure  
) ;
```

Parameters

pVisionParam

VISION_PARAM structure's pointer

Return Values

If a return value is true, the initialization is successful, if not, it returns false

Remarks

Before using the wireless image receiver, this function must be called. When this function is called, the wireless receiver dialog box is displayed.

Information

- Import library : vision.lib
- Header file : vision.h

vision_close

A function that terminates the wireless image receiver.

```
void vision_close( );
```

Parameters

None

Return Values

None

Remarks

After initializing with the `vision_open` function, before closing the application program, this function must be used to terminate the wireless image receiver.

Information

- Import library : `vision.lib`
- Header file : `vision.h`

vision_start

A function that enables the wireless image receiver to start capturing images.

```
bool vision_start( );
```

Parameters

None

Return Values

If the return value is true, the initialization is successful, if not, it returns false.

Remarks

After the wireless image receiver is initialized using `vision_open`, this function must be called to start the image capture. If this function is not called, the application program will not start.

Information

- Import library : `vision.lib`
- Header file : `vision.h`

vision_get_state

A function that checks the state of the wireless image receiver.

```
VISION_STATE vision_get_state( );
```

Parameters

None

Return Values

Returns VISION_STATE structure. By checking this structure, you can check the state of the wireless image receiver.

Remarks

The wireless image receiver must use the vision_start function to start capturing images in order to check for correct information.

Information

- Import library : vision.lib
- Header file : vision.h

vision_RGB24toDDB

A function that generates a bitmap handle that will be used in a Win32 application program.

```
HBITMAP vision_RGB24toDDB(  
    HDC  hDC;           // Handle of DC  
    unsigned char* pRGB24; // RGB24 format image  
) ;
```

Parameters

hDC

DC (Device Context) handle that is used in outputting to a screen in Win32 programming.
In an application program,
Use Window HDC that will be outputted in a Win32 program.

pRGB24

It is an image data pointer of RGB24 format that will convert to a bitmap handle for display purposes.

Return Values

DDB(Device Dependent Bitmap) handle that is used generally in outputting to a screen in Win32 programming.

For more information on HBITMAP, refer to reference books on Win32 programming.

Remarks

The resolution of pRGB24 data is used as a set data in VISION_PARAM.

Information

- Import library : vision.lib
- Header file : vision.h

vision_GraytoDDB

A function that generates a bitmap handle that will be used in a Win32 application program.

```
HBITMAP vision_GraytoDDB(  
    HDC  hDC;           // Handle of DC  
    unsigned char* pGray; //Gray format image  
) ;
```

Parameters

hDC

DC (Device Context) handle that is used in outputting to a screen in Win32 programming.
In application program,
Use Window HDC that will be outputted in Win32 program.

pGray

It is an image data pointer of Gray format that will convert to a bitmap handle for display purposes.

Return Values

DDB(Device Dependent Bitmap) handle that is used generally in outputting to a screen in Win32 programming.

For more information on HBITMAP, refer to reference books on Win32 programming.

Remarks

The resolution of pGray data is used as a set data in VISION_PARAM.

Information

- Import library : vision.lib
- Header file : vision.h

vision_DrawDDB

A function that will output DDB(Device Dependent Bitmap) to the screen.

```
bool vision_DrawDDB(  
    HDC  hDC;           // Handle of DC  
    Int  x;             // Position x of drawing area  
    Int  y;             // Position y of drawing area  
    Int  width;         // Width of drawing area  
    Int  height;        // Height of drawing area  
    HBITMAP  DDB;       // DDB  
) ;
```

Parameters

hDC

DC (Device Context) handle that is used in outputting to a screen in Win32 programming.
In application program,
Use Window HDC that will be outputted in Win32 program.

x

X coordinate range that will be displayed on the actual screen.

y

Y coordinate range that will be displayed on the actual screen.

width

Width size range that will be displayed on the actual screen.

height

Height size range that will be displayed on the actual screen.

DDB

DDB(Device Dependent Bitmap) is used for outputting to a screen in Win32 programming.

Return Values

If the return value is true, the initialization is successful, if not, it returns false.

Remarks

Since the DDB's width and height size information is included in the function itself, the DDB that is not used in vision library can be displayed on the screen.

Information

- Import library : vision.lib
- Header file : vision.h