ROBOTIS PREMIUM

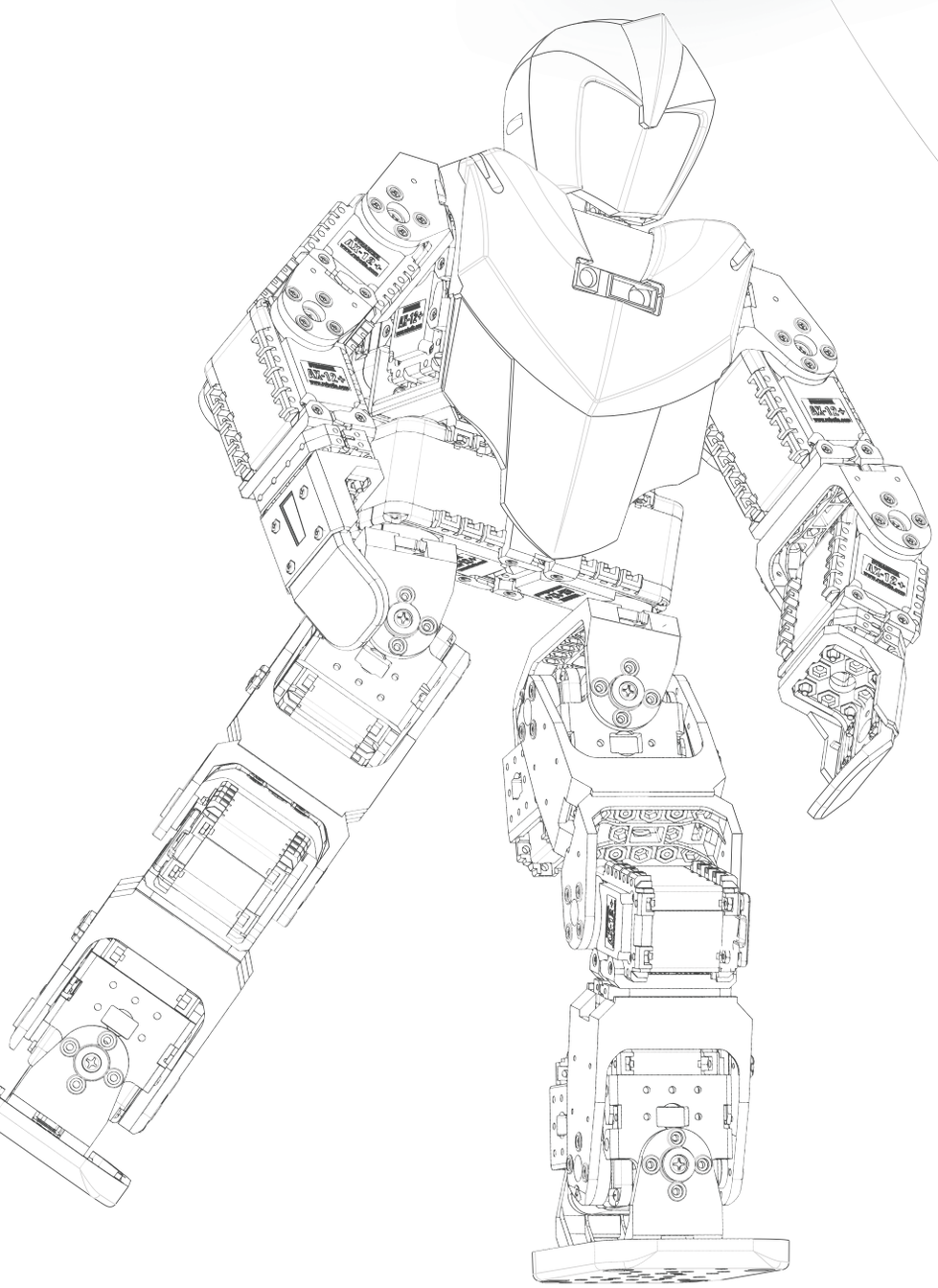
User's Guide

ROBOTIS PREMIUM

1. Includes 26 various types of robot assembly examples
2. Includes a gyro sensor, absolute distance sensor (DMS-80), IR sensor, etc
3. Includes a remote controller (IR-based, Zigbee and Bluetooth optional)
4. RoboPlus software with user-friendly graphical interface
5. Humanoid skin and cable holder
6. Digital packet communication and organized wires with daisy chain topology
7. Daisy-chain topology wire connection with simplified management
8. Versatile expansion capability for building various robots

- For detailed information please visit our e-Manual [http://emanual.robotis.com] or the User's Guide found in RoboPlus.
- Please note robot parts and color may vary from those illustrated in this book.

Content



1. ROBOTIS PREMIUM - Getting Started

1 - 1 Precautions

For your safety

Always exercise personal safety when assembling your robot. ROBOTIS is not liable for any accidents due to user negligence. Please read the instructions carefully before getting started.

- Read the manual carefully before assembly.
- Product recommended for ages 15 and older. Adult supervision required for users under the age of 15.
- Use only the tools provided or listed in this kit. (no knives, cutters, and drills)
- Do not handle product when sick, tired, or under the influence of drugs or alcohol.
- Keep a safe distance from the robot during its activation.
- Keep product of parts of this product out of reach of small children.
- Do not place fingers near the robot's joints.
- Operate the robot indoors only.
- Do not store or operate the robot under direct sunlight.
- Do not store or operate near water, heat, or fire.

Warning

Please read very carefully.

- Do not make your own cable(s) if you are a beginner user.
- Use only recommended screwdrivers.
- Do not use excessive force to insert nuts, bolts, or robotic parts.
- Please turn off the robot immediately when the robot joint(s) is(are) in abnormal position.
- Refer to the assembly manual and follow the instructions from ROBOTIS. Be advised that it takes at least 6 months of training to customize a robot.
- Activate the robot on the floor and not on an elevated surface such as desks. A fee will be applied for repairs caused by user's negligence.
- The gears inside the AX-12A robot joints) are expendable. Gear backlash may occur from any Dynamixel(s) with extensive use.
- When the robot's power is supplied by the SMPS ensure the robot does not fall to the ground to prevent cable damage.

Robot Assembly

Please read the following very carefully before assembly.

- For beginners to not start a robot with more than 10 joints (i.e. advanced examples). Certain amount of experience is required to build more complex robots.
- ROBOTIS recommends users to learn further about this robotic kit. Please check ROBOTIS' homepage (www.robotis.com) for a list of certified instructors and labs. The curricula include building unique and interesting robots as well as contents to supplement this user's guide.

Error

Contact ROBOITS or any certified reseller when of the following occur:

- Smoke emission is visible from the product.
- The LED of robot joints does not blink when it is turned on.
- Water or any other liquid is spilled on the robot.
- A strange odor comes out of the robot.
- The robot is damaged.

Recommended Tools

Use only the tools listed below. The use of any other tools may cause accidents.

- Screw driver : M2 Size
- Flat-headed driver: If the head of bolts for the screwdriver is worn out, use a flat-headed driver to replace the bolt.

1 - 2 Understanding ROBOTIS PREMIUM

What is ROBOTIS PREMIUM?

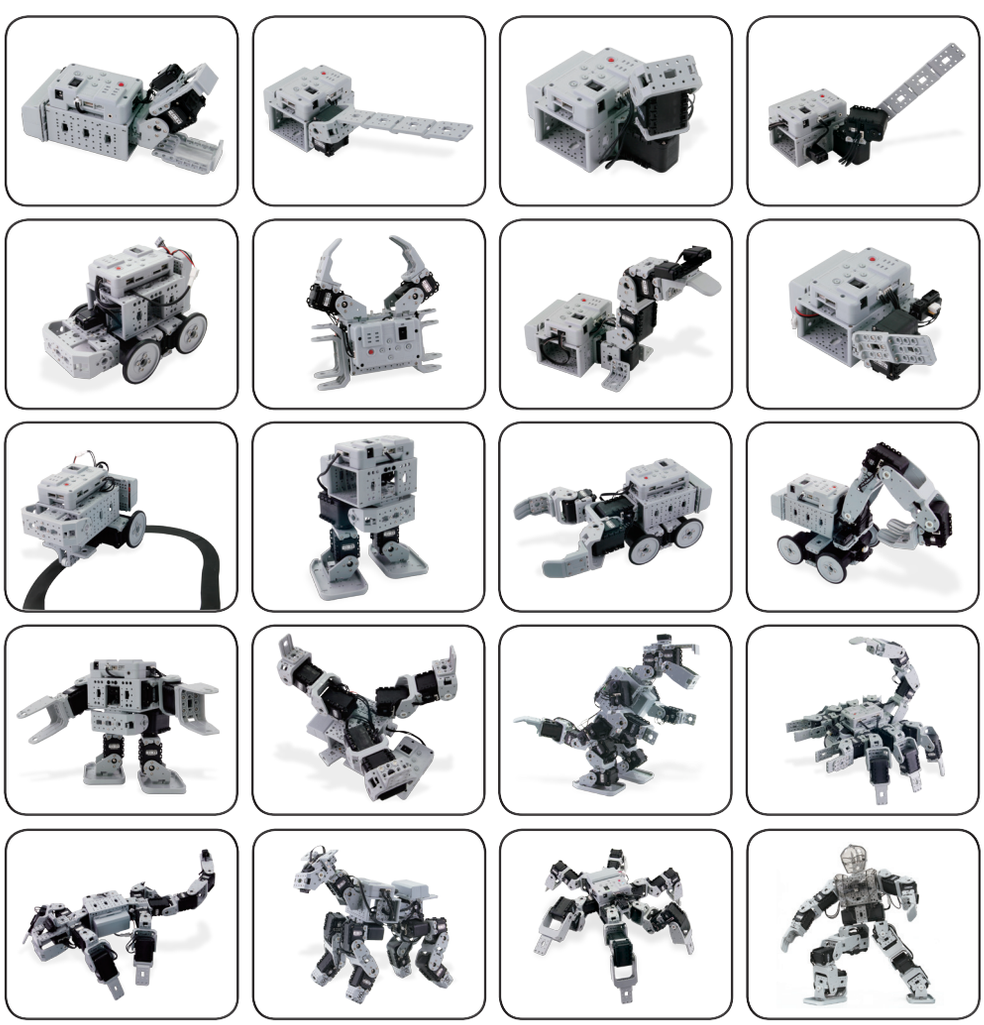
ROBOTIS PREMIUM is a complete robot kit. Like block toys, the user can implement the building parts of this kit create many types of robots. Unlike block toys, user's creation can come to life!

Various Functions

ROBOTIS PREMIUM robot can move autonomously based on information collected by its sensors and joints. For example, a robotic puppy that reacts to claps; a robot that bows to a person; a robot that avoids obstacles; a robot that plays with a ball; a robot that can be remote controlled. The user can realize the abovementioned robot capabilities with the included RoboPlus software.

Various Forms

The following are examples from the ROBOTIS PREMIUM kit.

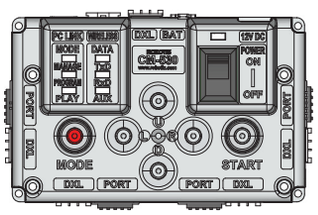


ROBOTIS PREMIUM Robot Examples

1-3 Key Components of ROBOTIS PREMIUM

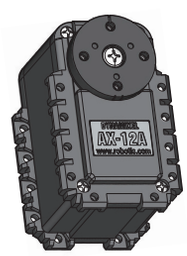
1-3-1 Hardware

CM-530



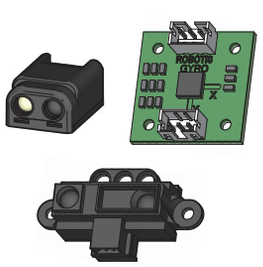
The CM-530 is the main controller for ROBOTIS PREMIUM. It can manage up to 26 AX-12A Dynamixels simultaneously. The buttons can be implemented as user inputs and the controller is capable to hear and emit sound with its embedded microphone and buzzer.

AX-12A



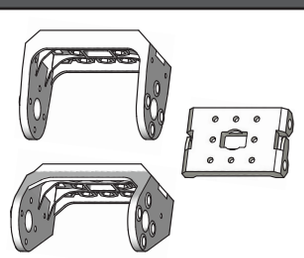
The AX-12A is a servo actuator specifically designed for robot joints. The user can control the speed and position of the servo, as well as detect the temperature and overload. This servo can also be used as wheels in endless rotation mode.

Sensors



Sensors are the eyes and ears that maintain the static equilibrium of the robots. It can detect the distance between objects, brightness of the environment, imbalances, and it is able to distinguish between colors black and white.

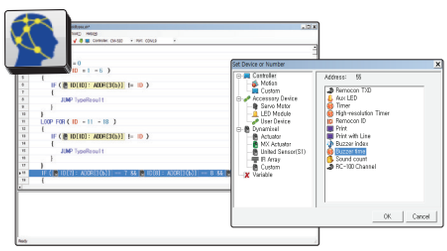
Other Assembly Parts



The parts comprises of frames, cables, and wheels. These parts can be used to connect the CM-530, Dynamixel AX-12A, and sensors with nuts and screws included in the kit.

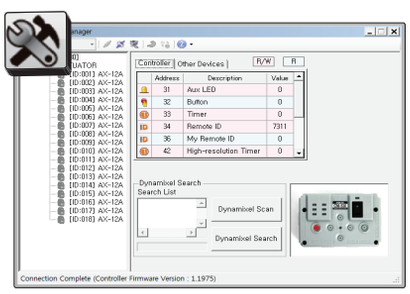
1 -3-2 Software(RoboPlus)

ROBOTIS PREMIUM is bundled with RoboPlus, a dedicated robotic software, for easy robot programming. Advanced users can take advantage of RoboPlus features to create unique robot motions.



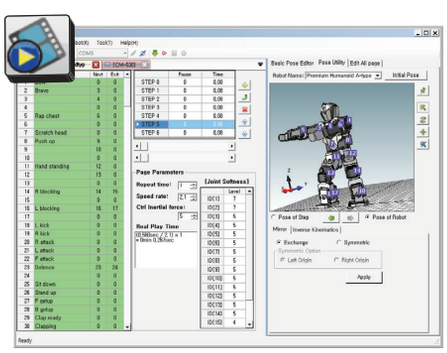
RoboPlus Task

Task software is for creating and downloading robot behavioral algorithms into the robot.



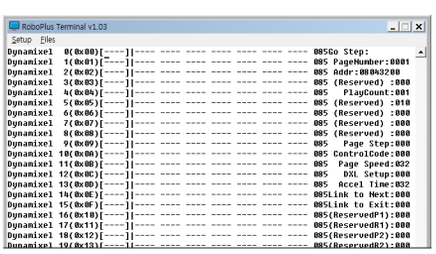
RoboPlus Manager

Manager software is for managing robotics device components



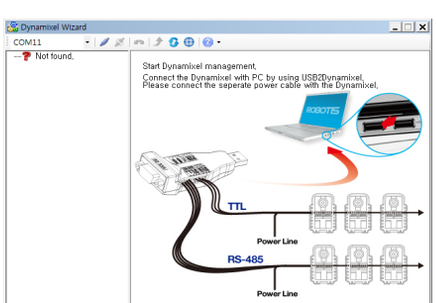
RoboPlus Motion

Motion is software for creation of complex robotic motions easily. Movements created in Motion can be called in Task.



RoboPlus Terminal

Terminal is low-level software for advanced users wishing for greater control of the robot Terminal is a type of serial communications terminal that allow visual transfer of data from computer to robot and robot to computer.



Dynamixel Wizard

Wizard is software to manage Dynamixels. Its main functions are Dynamixel firmware management and check-up.

1-4 Assembly Instructions

Assembly procedures for the ROBOTIS PREMIUM are as follows.

Step 1	Read Instructions
Step 2	Decide on the Robot's Shape and Functions
Step 3	Assembly/Wiring
Step 4	Check Assembly
Step 5	Programming with RoboPlus Task
Step 6	Programming with RoboPlus Motion
Step 7	Download/Execute

Step1 Read Instructions

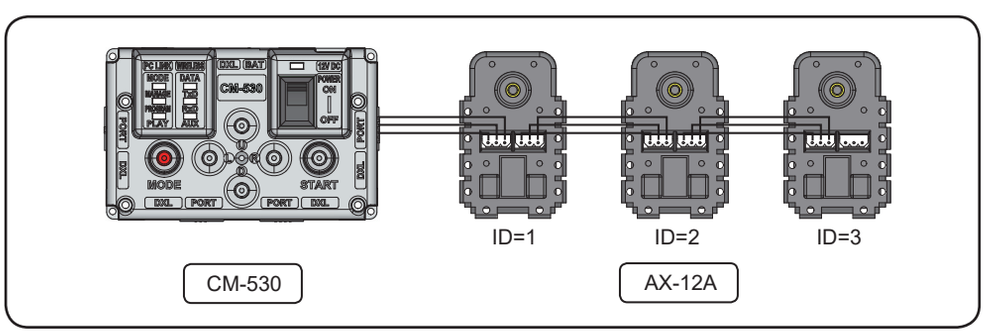
Read this guide thoroughly before assembly. Refer to this guide for detailed assembly instructions and detailed diagrams.

Step 2 Decide on the Robot's Shape and Functions

Decide on the robot to be assembled. The ROBOTIS PREMIUM kit allows reconfiguration of robots. It is recommended to begin with examples sequenced in this guide.

Step 3 Assembly/ Wiring

Assemble the AX-12As, sensors and frames around the CM-530. Then wire each AX-12A to the CM-530 with cables provided in this kit. Refer to the diagram below on how to wire the robot. Connect cables on either connector of the AX-12A to connect op to 26 AX-12As. Pick cables with lengths to allow enough slack for joint movements.



Step 4 Check Assembly

Once robot is built download RoboPlus Task to verify proper assembly (refer to this guide).

Step 5 Programming with RoboPlus Task

Write a behavioral program with Task. Create conditions, such as, “if there is an object in front move backwards…” Refer to chapter 3 for details on Task.

Step 6 Programming with RoboPlus Motion

Create movements with Motion efficiently. Call these motions with Task. Refer to chapter 4 for details on Motion.

Step 7 Download/ Execute

Download the Motion file into the CM-530. Once download is complete press the PLAY button followed by START to execute program.

1 - 5 System Requirements

System Requirements

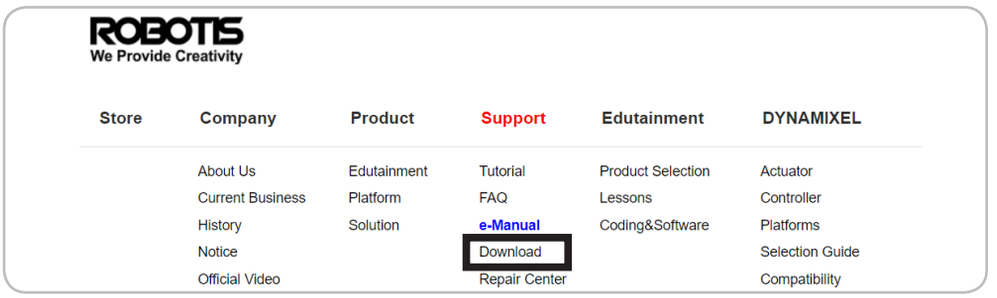
Below are the system requirements for RoboPlus :

- OS : Windows XP Service Pack 2 or above/ Vista/ 7 (32/64bit)
- 800MHz 32 bit (*86) or 64 bit (*64) processors or higher
- Graphic Card that supports 3D accelerator
- System Memory more than 512 MB
- Hard Disk with more than 500 MB of free space

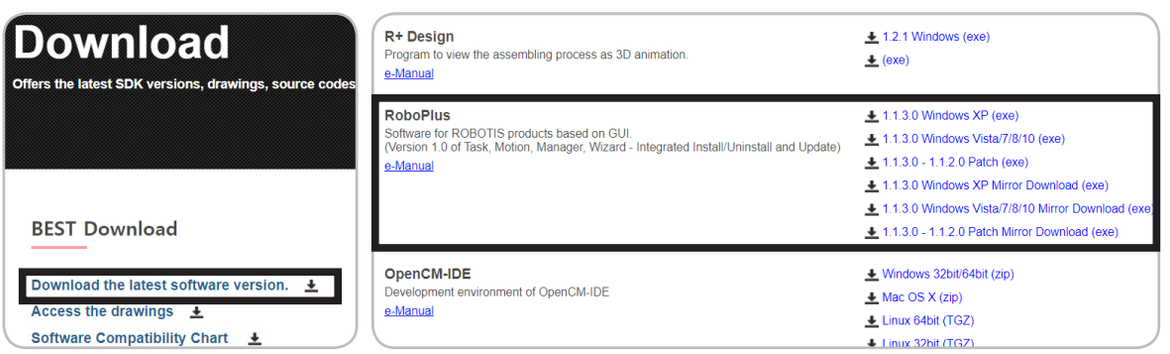
1-6 Installing RoboPlus

RoboPlus is an integrated software that can program all ROBOTIS products.
You can download RoboPlus from the ROBOTIS website. (http://en.robotis.com)

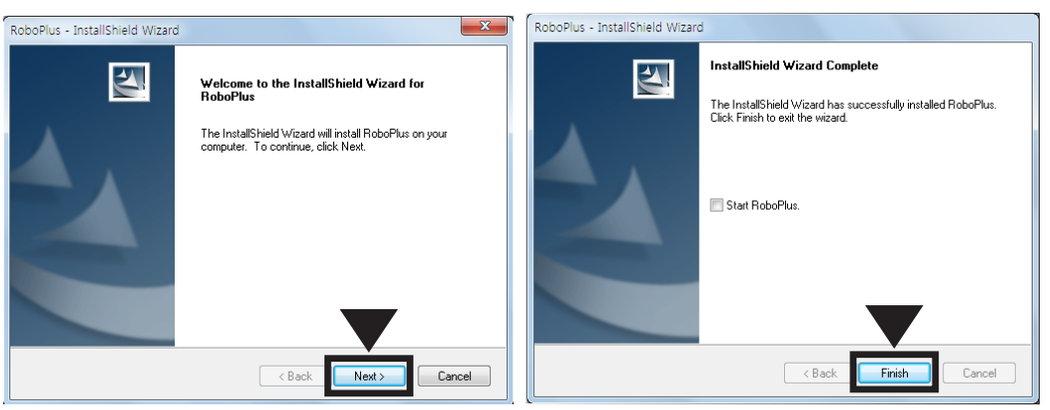
1. Access the ROBOTIS homepage and click **Support > Download** on the ROBOTIS website.



2. From the [Best Download] list, select [Download the latest software version]. Install the appropriate file according to the Windows version of your PC.

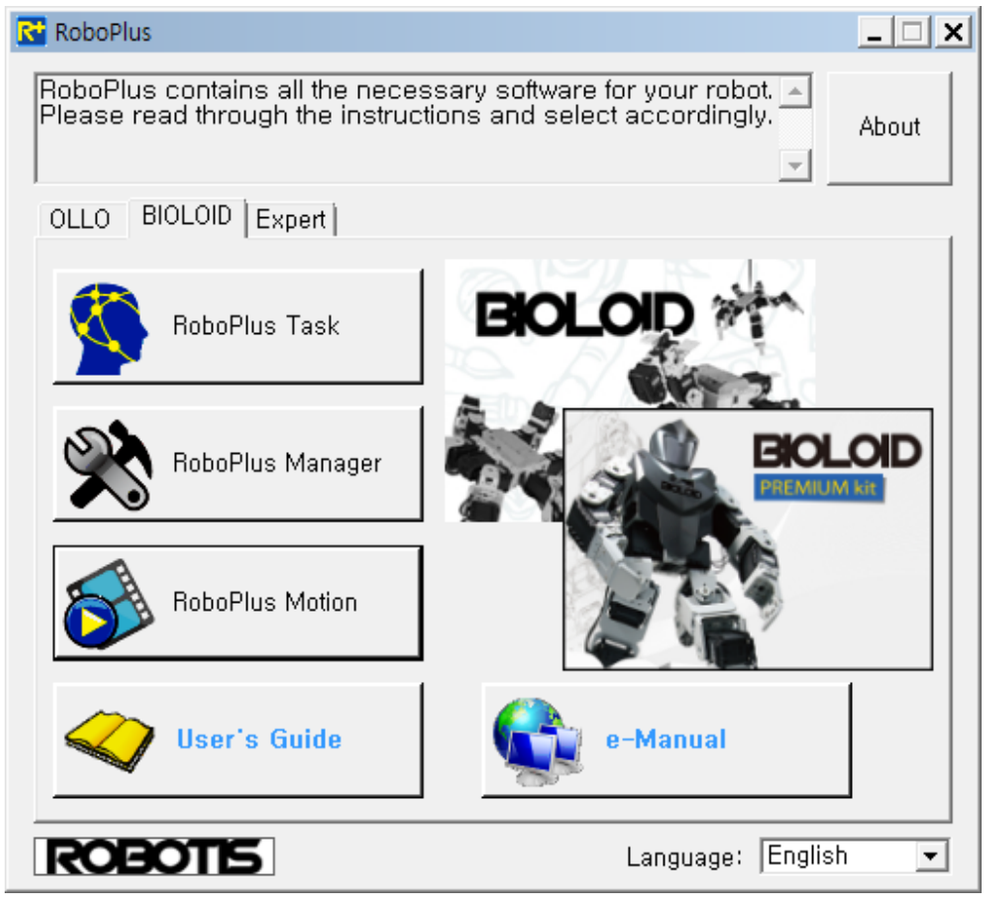


3. Execute the installation file. Follow the instructions and press [Next] to install. Select [Finish] to complete the installation.



Note **NET FrameWork 305 or higher is required to execute RoboPlus. If .NET FrameWork auto-installation fails when installing RoboPlus, install .NET FrameWork manually.**

Once RoboPlus is successfully installed, a shortcut icon should appear on your desktop. Run RoboPlus to execute other included programs.

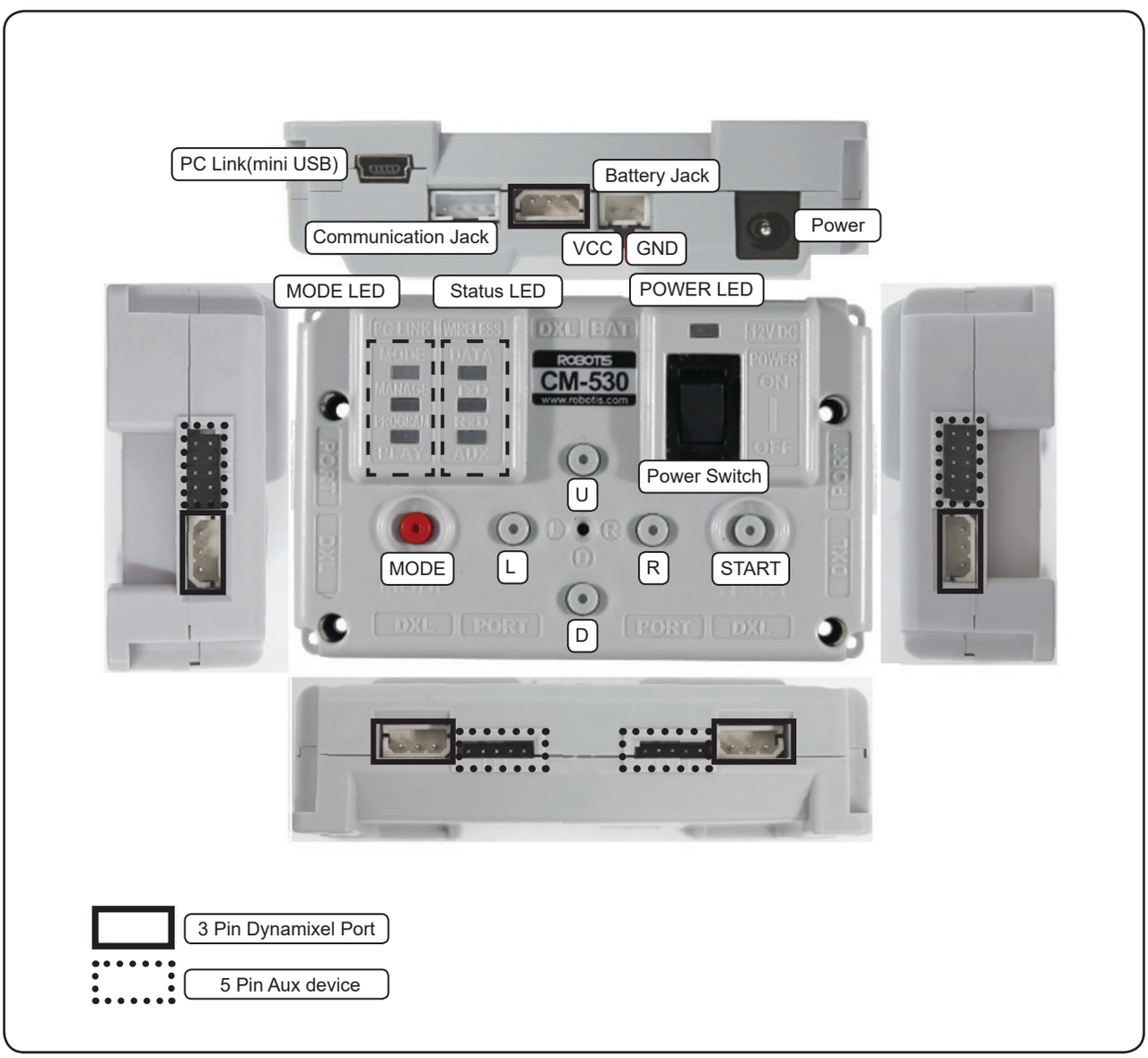


1 - 7 Important Hardware Information

Basic knowledge about the hardware is necessary before assembly. This chapter explains the functions of the CM-530, AX-12A, and sensors that are required to operate the ROBOTIS PREMIUM.

1 - 7 - 1 CM-530

The CM-530 controller controls the AX-12A and sensors. Write programs with Task, movements with Motion, download them into the CM-530, and execute them. The buttons of the CM-530 can be implemented as user inputs.



Power Supply

The SMPS can supply power to the CM-530 when the battery is not needed. When turned on an LED from “Mode” will blink.

Communi-cations

Connect the mini USB cable to the CM-530 and connect the other end of the mini USB cable to a PC USB port.

Mode

There are 3 modes of operation of the CM-530; these are “Manage,” “Program,” and “Play” indicated by its respective LED. When the “Manage” LED blinks the robot is on managing mode. When the “Program” LED blinks the robot is on programming mode. When the “Play” LED blinks the robot is running the downloaded program. Press the MODE button to switch between modes. Press the start button to select the mode.

Manage Mode

Checks the status of the CM-530, AX-12A, sensors, and tests motions. For advanced users, use this mode once the CM-530, AX12A, and sensors are fully understood.

Program Mode

The CM-530 automatically switches to “Program” mode when motion editor is in use. To edit motions with Robot Terminal run Robot Terminal and set the CM-530 to “Program” mode.

Play Mode

Set the CM-530 to “Play.” Set the CM-530 under this mode to run downloaded programs.

START

To start the robot, press the 'Start' button.
If the 'Start' button is pressed while the 'Standby' mode LED is blinking, the robot will start at the respective mode.

PAUSE

To pause the robot, press the 'Mode' button to put it on standby, or turn off the power.

Status indication LED

Status LEDs indicate the status of the CM-530. Each LED corresponds the folloinwg:

TXD: Turns on when the CM-530 is transmitting data
RXD: Turns on when the CM-530 is receiving the data
AUX: User-programmable LED

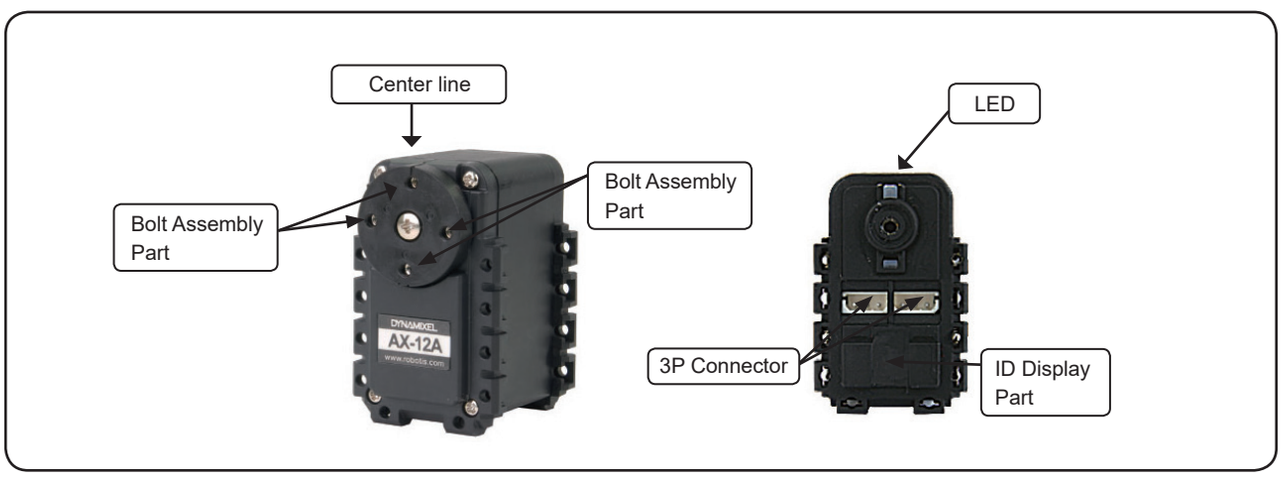
The AUX LED can be turned on and off with Task.

U, D, L, R Buttons

Buttons assigned for user input when running a robot program.

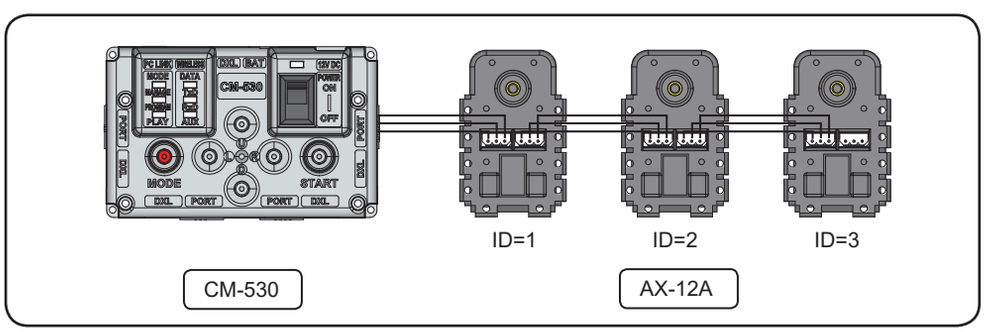
1-7-2 AX-12A

AX-12A is a servo actuator specifically designed for robot's joints. The user can control the position value of AX-12A by 0.29 degrees and assign the specific motor speed. It can be used as wheels by setting it to rotate endlessly. It has a function to detect temperature / over current / over voltage as well. For detailed instructions, please refer to '2-4. Test and Set Up.'



ID

IID is a number assigned to the AX-12A. An AX-12A ID number is unique to distinguish from other connected AX-12As to the CM-530.



Characteristic of ID

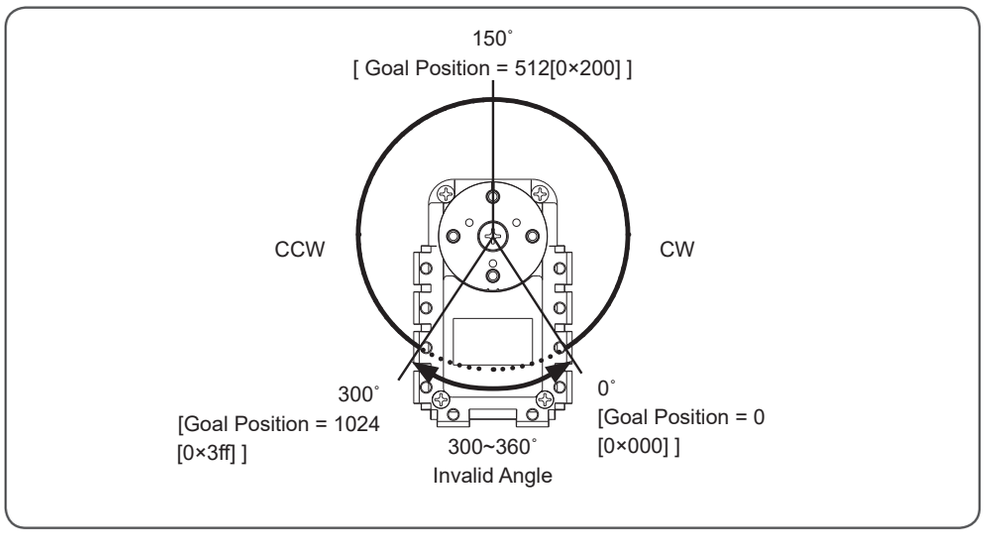
The ID can be manipulated by the user. The AX-12As included in this kit have unique numbers pre-assigned in consecutive order. However, AX-12As sold individually have a preset ID of 1. When including AX-12A(s) apart from this kit the ID must be changed. To change ID refer to "2-4 Test and Set-up."

Address

The AX-12A has many functions assigned in "addresses." A function can be manipulated by modifying the value of its address. Refer to "2-4 Test and Set-up" for more detailed information.

Use as a joints

Under Joint Mode the AX-12A can move from 0 to 300 degree positions. Position range is controlled with values from 0 to 1023. The image below illustrates the angle position with its corresponding value.



Use as a wheels

Under Wheel Mode the AX-12A can rotate continuously under a speed value. Speed values range from 0 to 1023. Refer to “2-4 Test and Set-up” for more detailed information.

1 - 7 -3 Sensors

When a human senses an external stimulation, the signal is sent to the brain to make decisions and moves the muscles accordingly. Likewise, the ROBOTIS PREMIUM recognizes an external stimulation with its sensors, sends the signal to the CM-530 to make a decision, and moves the AX-12A accordingly.

IR Sensor

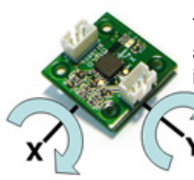
Used to detect nearby objects or color (black / white)

DMS-80

Used to detect and measure the distance from close to fairly far (10cm~80cm)

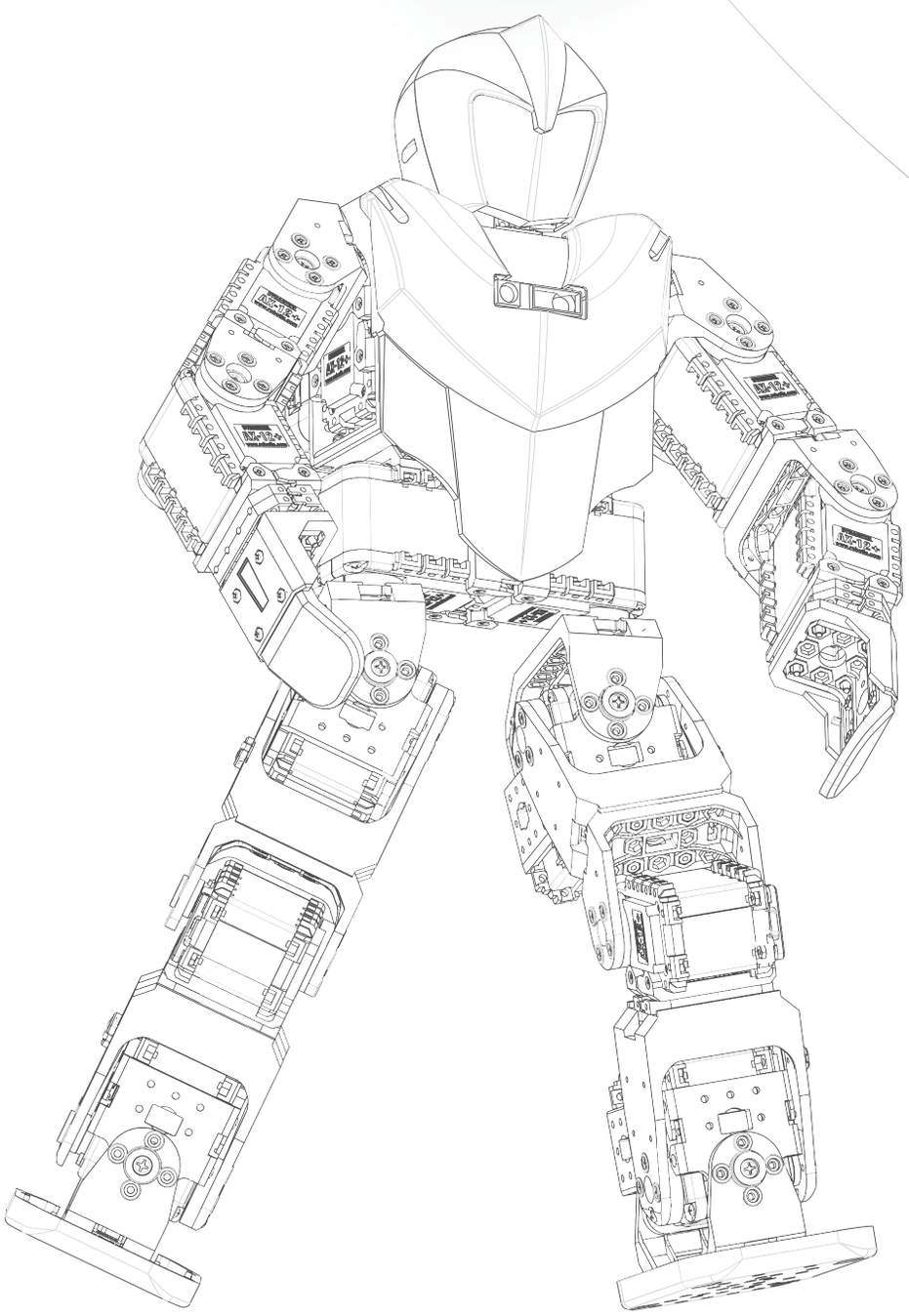
Gyro Sensor

Used to detect the momentary balance of the robot (angular velocity). It defines both X-axis and Y-axis values.



The direction where Y axial angular velocity increases

The direction where X axial angular velocity increases



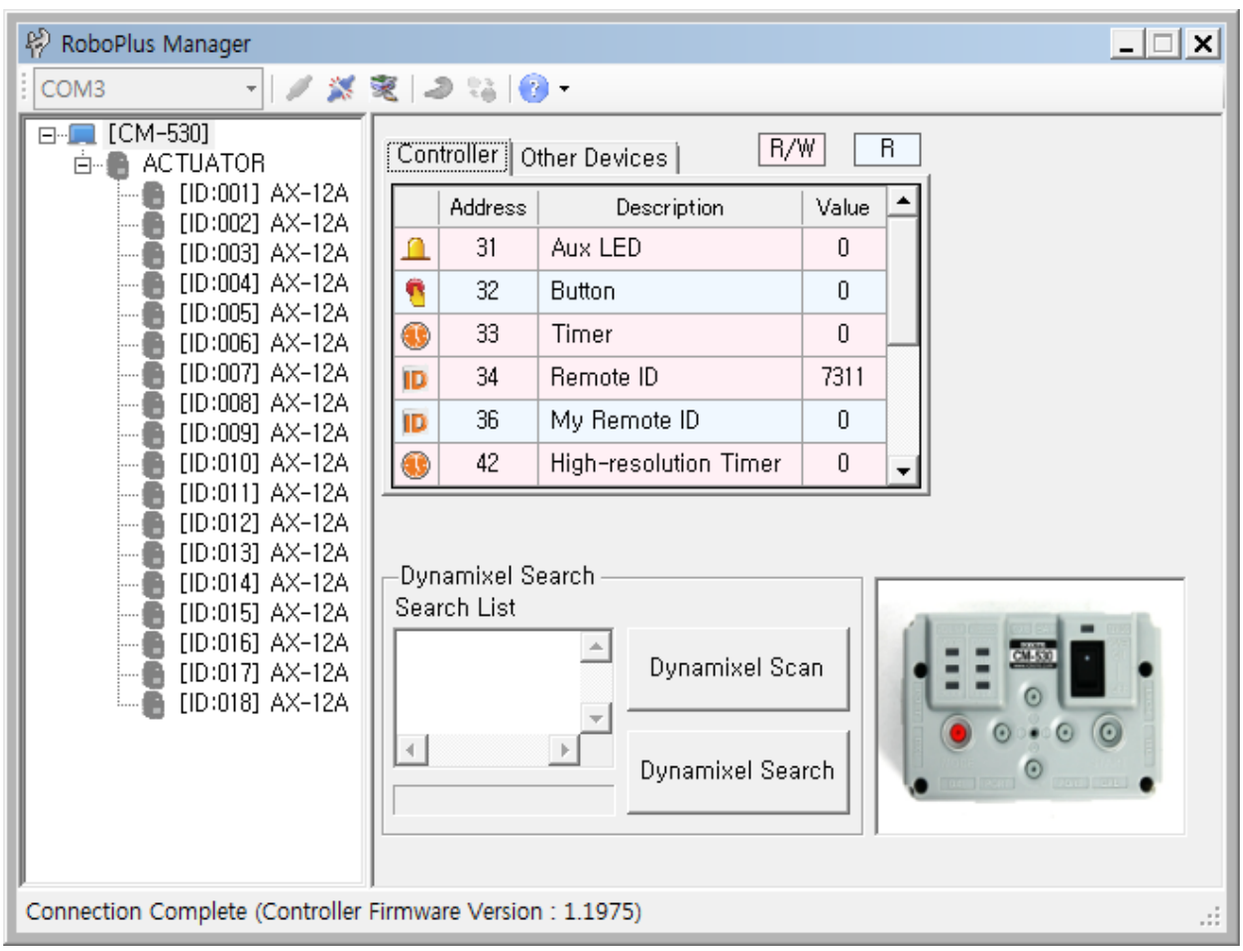
2. RoboPlus Manager

2 - 1 What is RoboPlus Manager?

RoboPlus Manager

RoboPlus Manager manages the devices that Comprise the robot.

- Manages the firmware. (Updates and restores functions)
- Examines the main controllers and other devices. (Tests functions)
- Sets the required modes. (Settings)



2-2 Getting Started

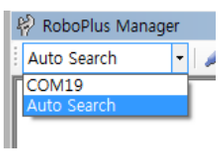
Connect to CM-530

[STEP 1]

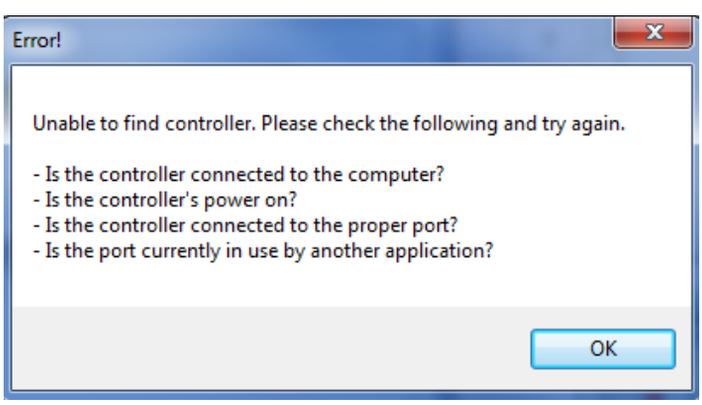
Connect the robot to the PC via mini USB cable. Connect the mini USB end to the CM-530 and the opposite end to any USB port of the PC.

[STEP 2]

Select the COM port number of the CM-530. Select “Automatic Search” to automatically select the COM port number.



If RoboPlus Manager is unable to find the controller the following error message can appear :



- Check if the CM-530 and PC are connected.
- Check if the CM-530 is turned on.
- Check if you selected the correct communication port.

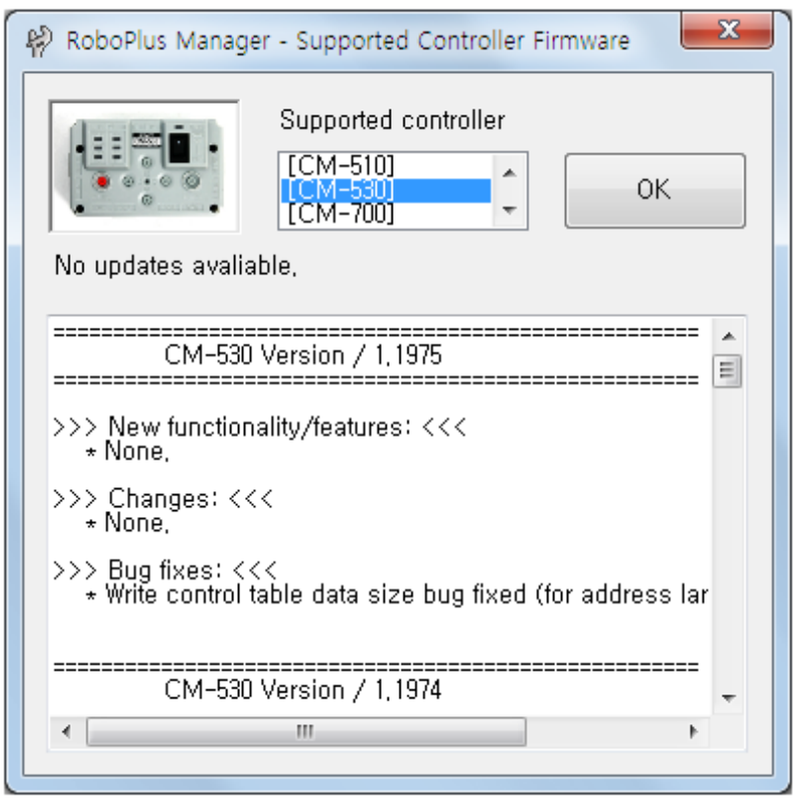
[STEP 3]

Start Managing

2-3 Firmware Management

2-3-1 Firmware Update

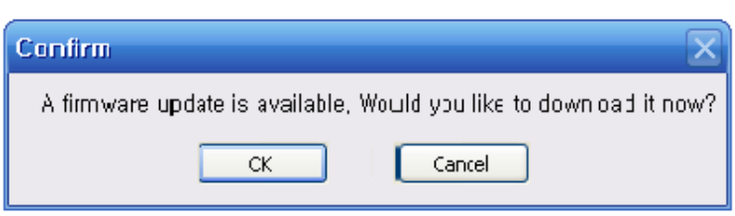
Firmware is code installed to the CM-530 to enable execution of Task programs or controller management. Manager automatically detects the latest firmware via internet.



How to update

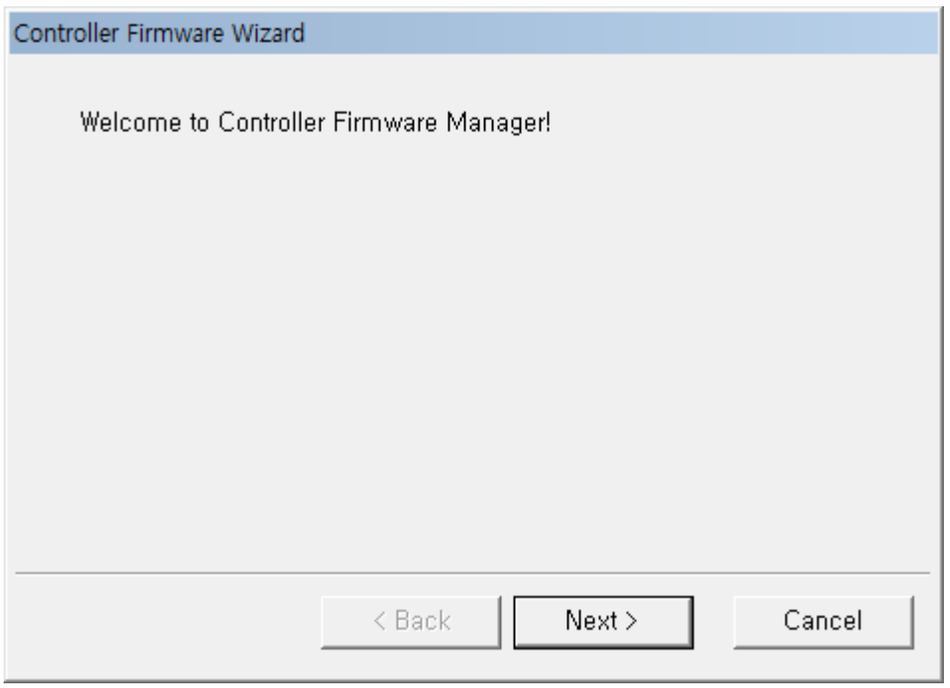
[STEP 1]

When the CM-530 is connected, RoboPlus Manager automatically examines firmware version. When a newer firmware version is available a message will appear asking whether or not to update firmware.



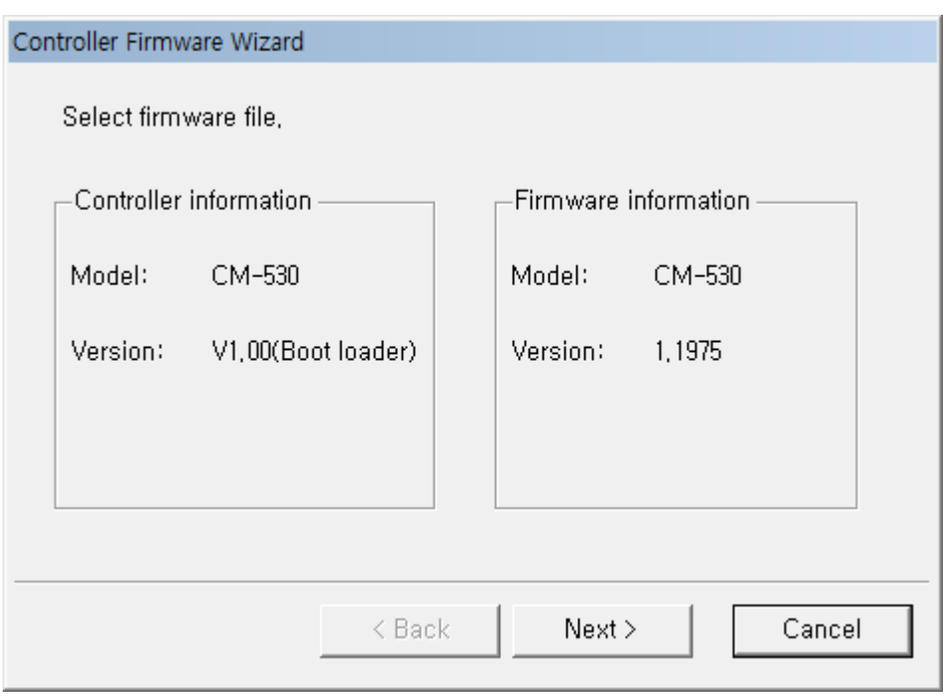
[STEP 2]

Press 'Next' to start the Firmware Update Wizard.



[STEP 3]

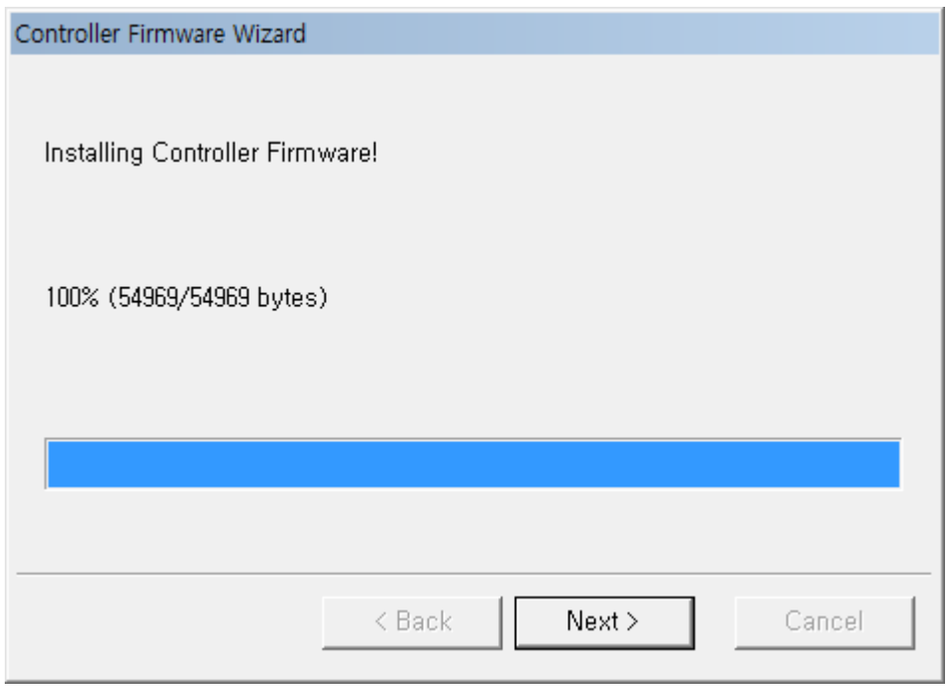
The model number of the CM-530 and the version of the firmware will appear.



[STEP 4]

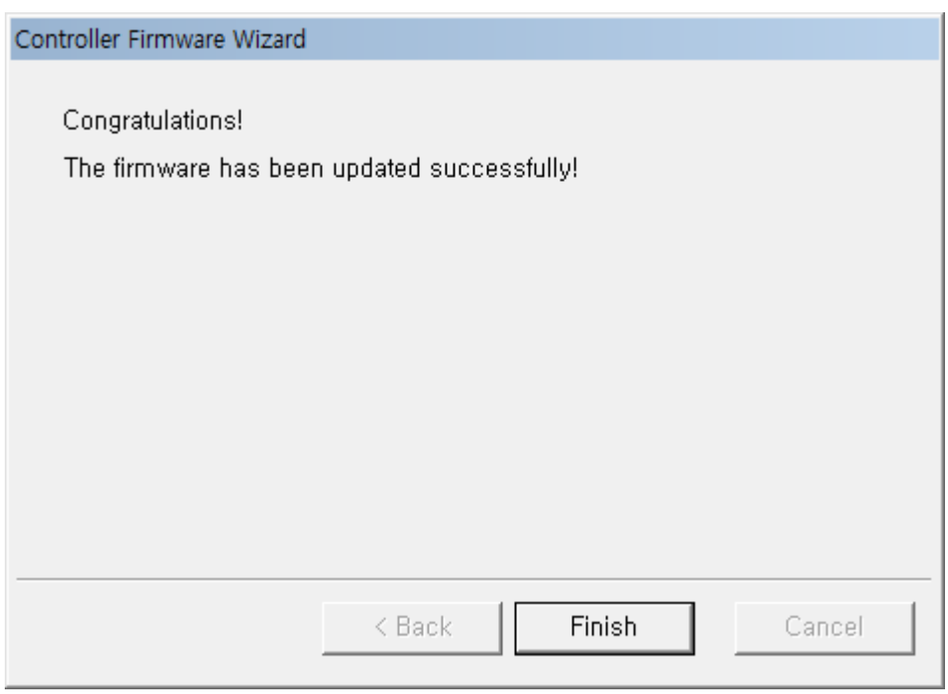
Press 'Next' to start the update.

Do not turn off the power or disconnect the cable during this process.



[STEP 5]

Check for the result of the firmware update



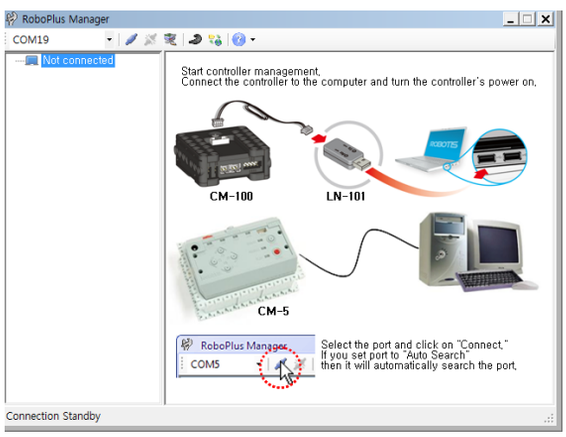
2-3-2 Firmware Restoration

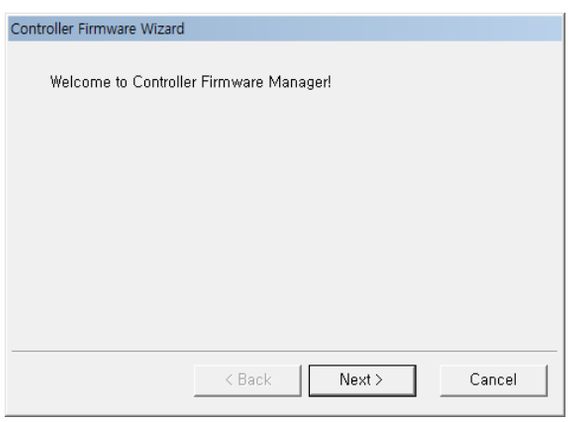
If the controller has firmware issues, RoboPlus Manager can help you restore its firmware.

How to restore

[STEP 1]

Go to Tools>Restore the Controller to start 'Controller Firmware Wizard' (or click on the 'Controller Firmware Management' icon)

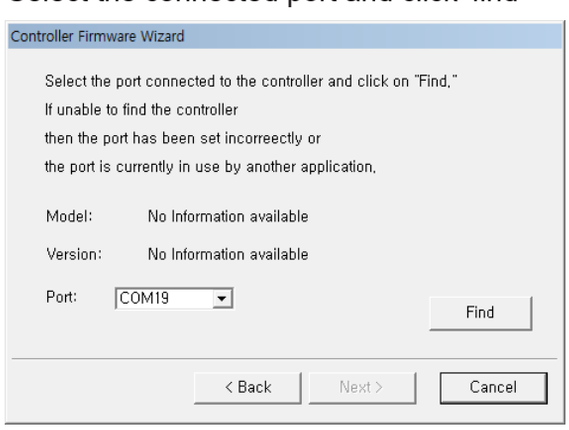




[STEP 2]

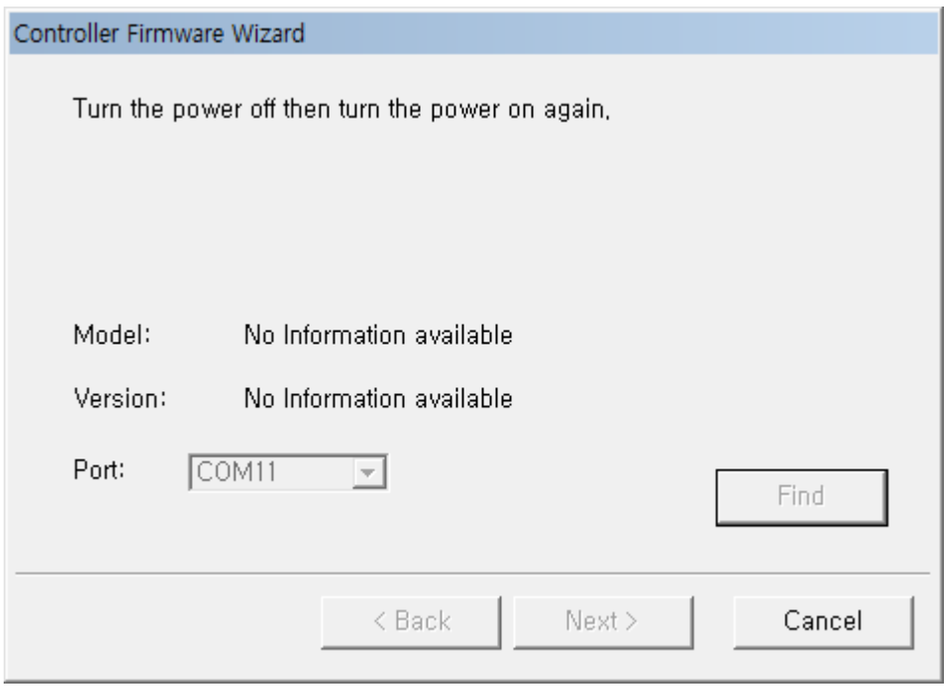
Manager cannot recognize the current firmware and unable to automatically select the COM port of the CM-530. Select the correct port manually.

Select the connected port and click 'find'



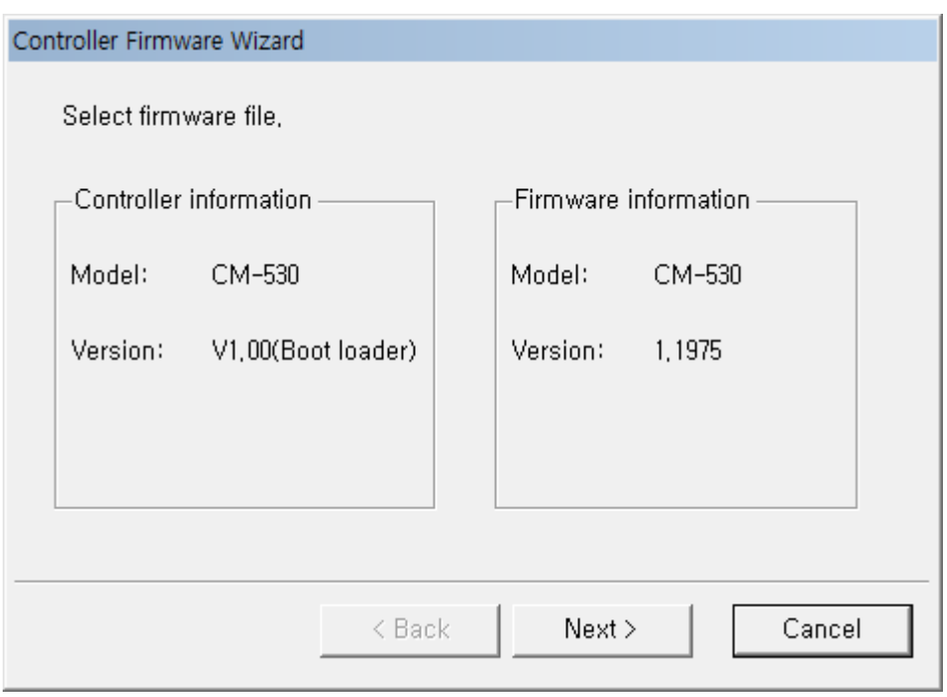
[STEP 3]

To detect the CM-530, turn the power off and then turn back on.



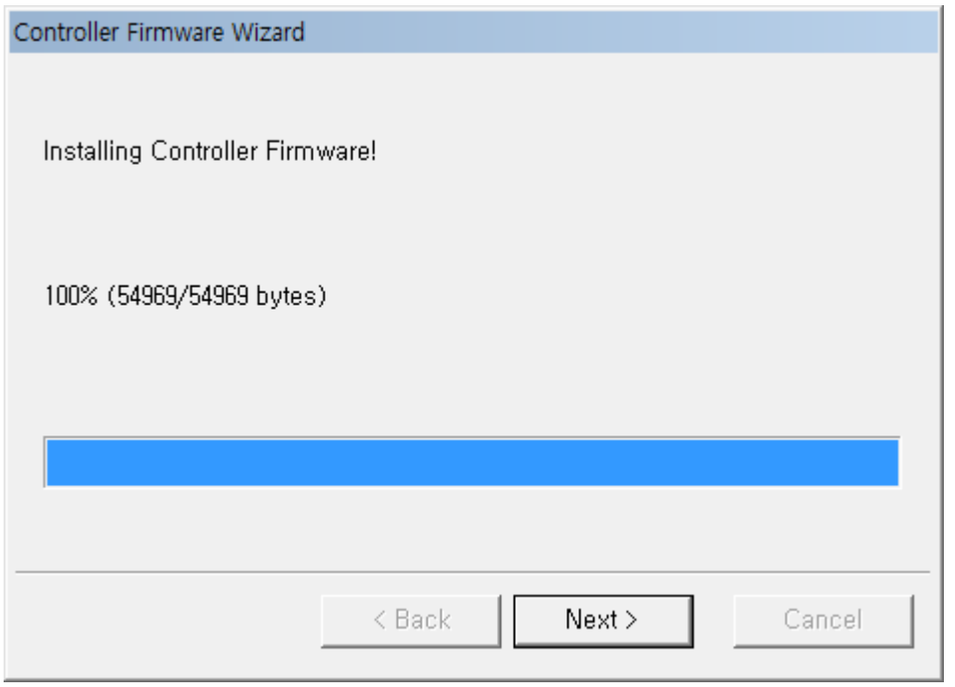
[STEP 4]

Once the CM-530 is detected, the controller's information and downloadable firmware information will appear. Verify if this information is correct. Please note: the version of the CM-530 information shown here refers to the Boot Loader version.



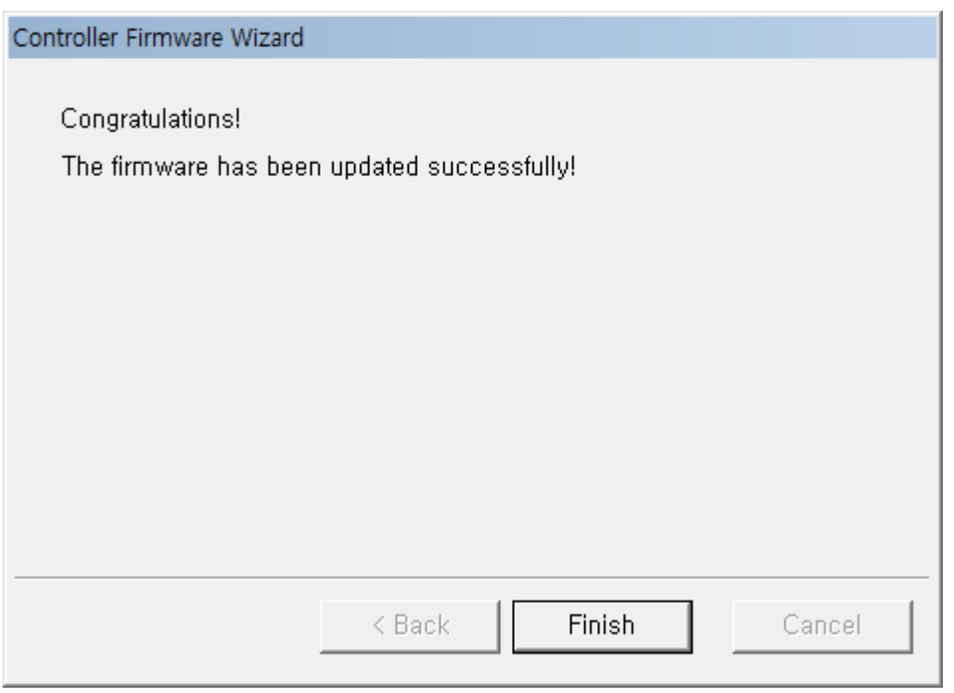
[STEP 5]

Click the 'Next' button to start the Firmware restoration process. Do not turn off the power or disconnect the cable during this process.



[STEP 6]

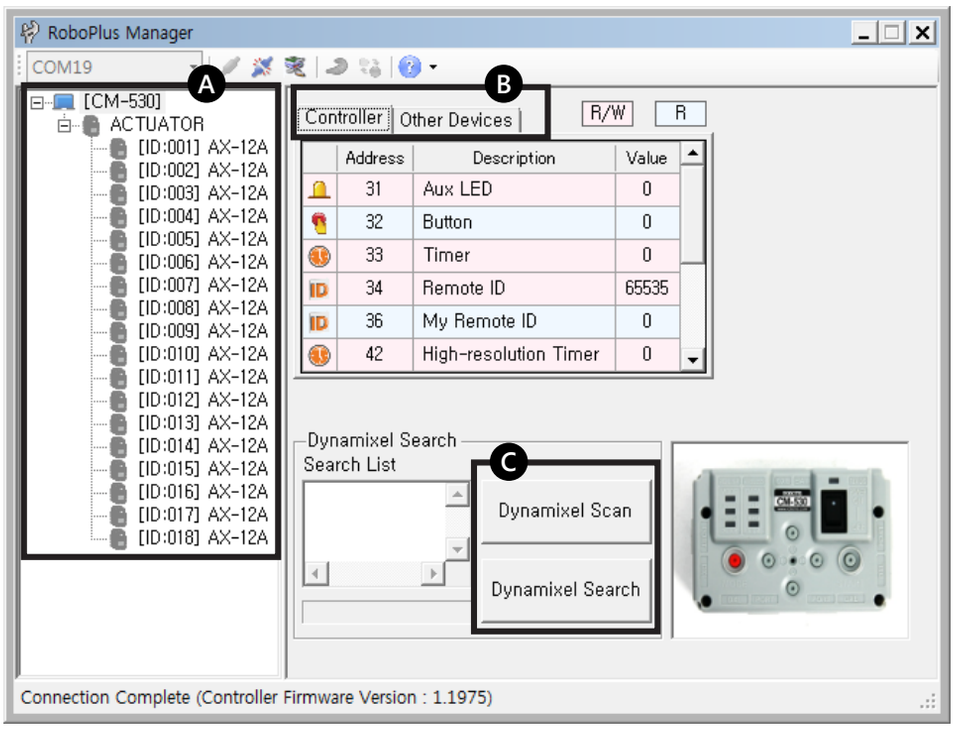
Check for the result of firmware restoration.



2-4 Test and Set Up

2-4-1 Introduction of the default window for RoboPlus Manager

Once the CM-530 is connected to RoboPlus Manager, you will see the following window.



A and B list the CM-530 and its devices. If the CM-530 has an older firmware than Manager it will ask to update. Please refer to “How to update CM-530 firmware.”

In window C RoboPlus Manager supports two types of search modes.

General Search

- Searches for the AX-12A connected to the CM-530 at 1 Mbps.
- Search speed is fast because it searches only for Dynamixels connected at 1Mbps.
- AX-12A connected at other speeds will not be detected.

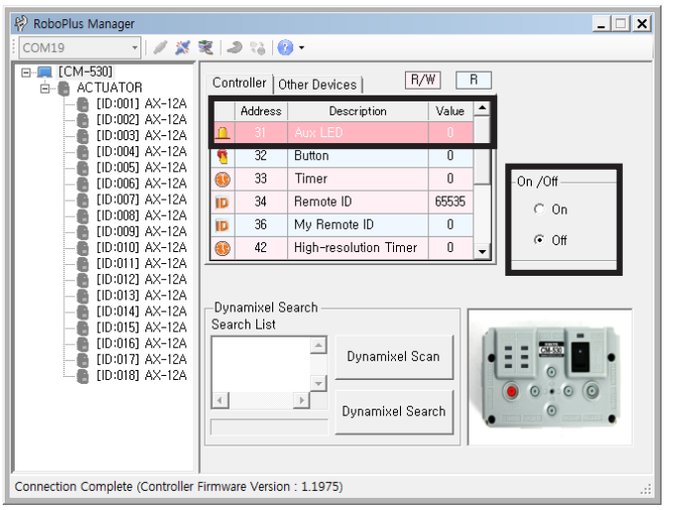
Detailed Search

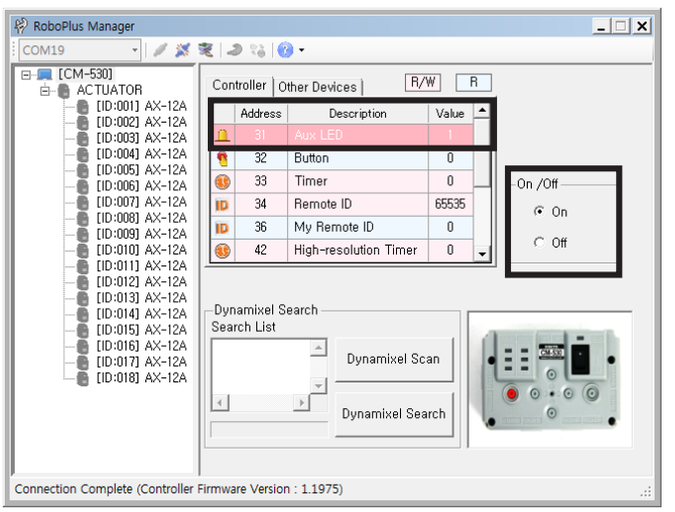
- Searches for all the AX-12A connected to the CM-530 at every possible communication speed.
- Search speed is slower than General search because it searches for all Dynamixels connected at different communication speeds.
- AX-12As that are not connected at 1Mbps are automatically adjusted to 1Mbps.

Aux LED

Controls the AUX LED of the CM-530.

- Operated AUX LED on or off.





Button

Reads the CM-530's button status (pressed or not).

When a button is pressed it will appear on the window.

Timer

Reads and sets internal timer of the CM-530.

- Sets up the timer value and checks the actual time as well.
- The timer value is 0~255.
- When the timer is set, it will start counting every 0.128 seconds.

Please note

If an AX-12A is not recognized after a detailed search ID information may have crashed as result of AX-12As having the same ID number. In such case connect only 1 unrecognized AX-12A and change ID number.

Wireless ID of your opponent's robot (Remo on ID)

Can read and set the opponent's Zigbee communications ID or read the preset ID value.

- Only works when the Zigbee module is connected.
- Wireless ID value can be set from 0~65535
- When the opponent's ID is 65535 then it sends data to all detected modules (broadcasting).
- For 1:1 communication the ID of both Zigbee modules should be an exact match.

MY robot's Wireless ID (My ID)

Reads the ID value of Zigbee Communications module installed in the robot.

- If the Zigbee module is not installed, it is read as '0'
- If the Zigbee module is installed, it identifies with a unique ID value (numeric number from 1 to 65534)

Number of detected Noise (Sound ount)

Detects noise through the embedded microphone of the CM-530.

- Can record the count of detected claps near the CM-530.
- Count up to 255.
- If no noise is detected for a certain amount of time, the number of detected sounds will be input into the "Sound Count" parameter.
- Noise count not automatically initialized. Count needs to be set to 0.

Current number of detected noise (Current Sound Count)

Checks the detected sound through the embedded microphone of the CM-530.

- Counts the number of detected noise instances up to 255.
- For every detection instance count increases by 1 value.
- If no noise is detected after a certain period the sound count is sent to the "Sound Count" parameter then "Current Sound Count" reset to 0 automatically

Buzzer Index

Sets the musical note played by the embedded buzzer of the CM-530.

- Plays either scales or melodies according to the value of "Buzzer Time" parameter.
- If "Buzzer Time" parameter is not in melody mode then it will not be played.
- Musical notes will be played for the length of time from buzzer timer parameter value.

- In Special Melody Mode melody plays completely.
- There is a total of 16 melodies (0-15) and 27 musical notes (0-26).
- If buzzer timer is 0 then its automatically set to 3 and plays the scale.

Buzzer time

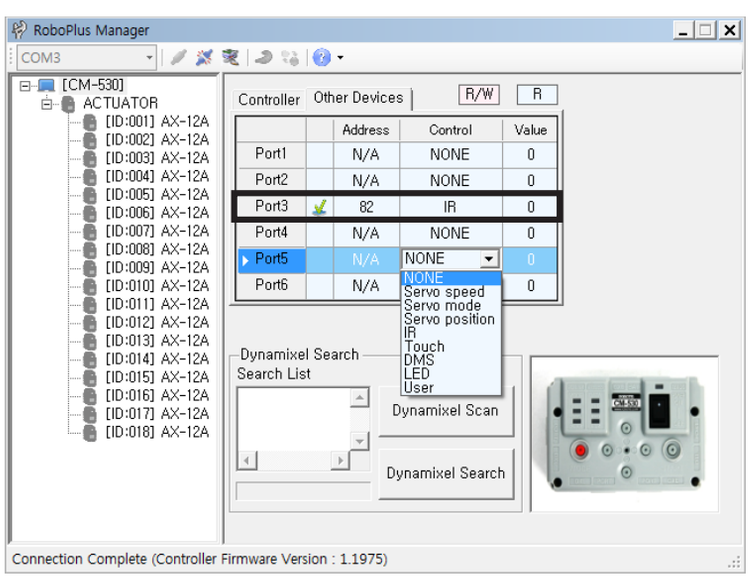
Sets the sound mode and length of playing time.

- If buzzer timer value is between 0 and 50 it plays the musical note and the duration of play can be verified.
- If buzzer time value is 255 it plays a special melody.
- When a scale or melody is finished playing buzzer timer is set to 0.

External I/O

The controller supports external I/O devices, such as IR Sensor, Touch Sensor, Distance Sensor, etc.

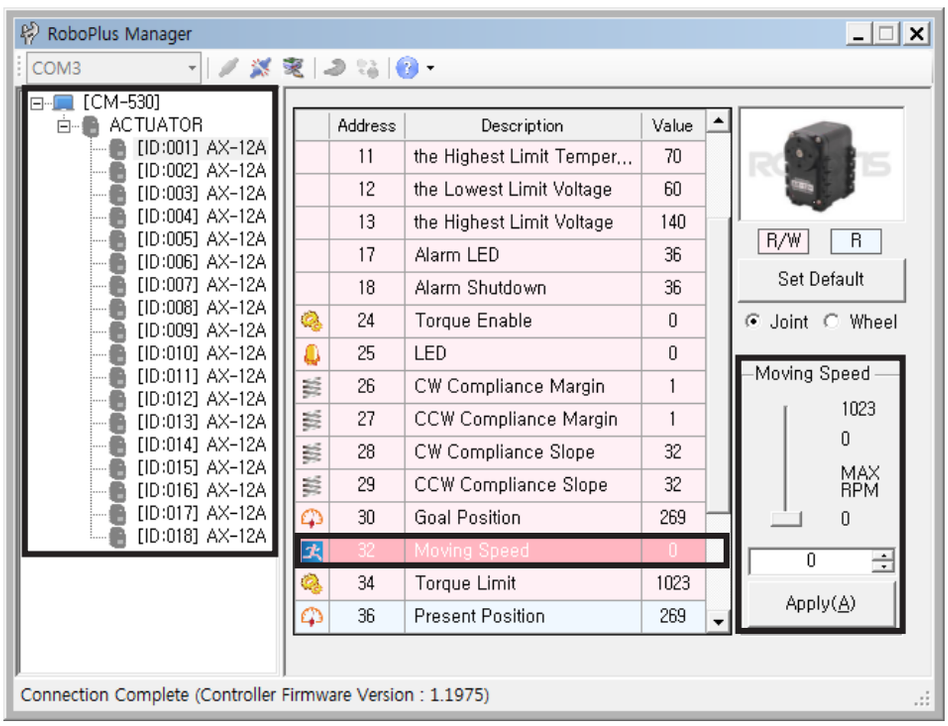
- The controller usually can only read values from I/O devices; but, writing may be possible if I/O device allows it.
- Connect the I/O device, set the type, and check the read value.
- Device output values can be set
- I/O devices are not recognized automatically; they need to be set according to the port number they are connected in.
- Supported devices are IR, distance, touch, and other user devices.
- If a device is not connected then the port will output random values erroneously.
- When the port with it respective device are set the controller sets the address value.



The image above illustrates an IR sensor connected to Port 3. Value will change depending on the distance of the IR sensor.

2-4-2 AX-12A

The image shown is to test and set up the AX-12A



Scan/Search function will list connected AX-12A(s) to the CM-530.
Click on the AX-12A from the list to manage the corresponding AX-12A. The sub-window list the addresses of the corresponding Dynamixel.

The image above is a screenshot of Manager with Moving Speed selected. The sub-window on the lower right corresponds to Moving Speed. Double-clicking on the ID of a AX-12A will cause the LED of the corresponding AX-12A to light up. This feature can be useful to locate and the AX-12A.

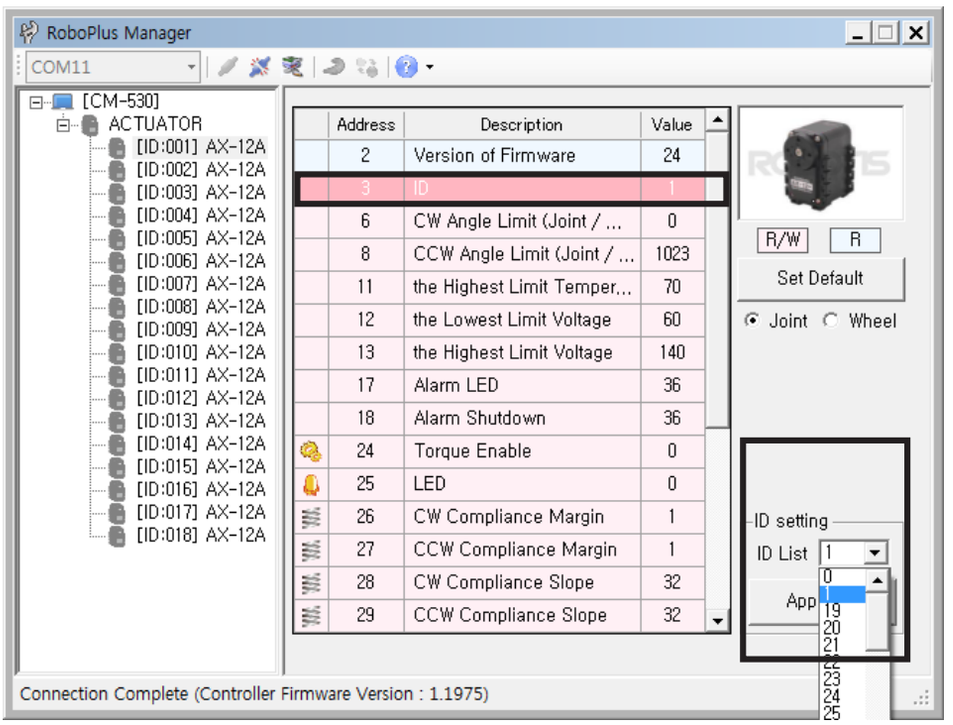
AX-12A set value

ID, position limitation, critical temperature, critical voltage
LED Alarm set, Shutdown Set, Turning on the motor, LED Compliance Margin, Compliance Slope, Goal Position, Moving Speed, Torque Limit
Present Position, Present Speed, Present Load, Present Voltage, Present Temperature, Moving

ID

AX-12A has an unique ID, which the CM-530 uses to control. With this command, you can manage the IDs of the AX-12A

- When you set values, the values are saved even when the power is off.
- Click to check the list of ID to see the other changeable ID.
- If an AX-12A is not listed its most likely due to having duplicate ID number.
- ID numbers range from 0 to 253. Number 200 is reserved for the CM-530.



Position Limit

Sets position limits of the AX-12A or set movement mode

- Values saved even after powered off.
- Position limits for the CW and CCW positions; Goal position values will depend according to the CW an CCW limits.
- You can check the actual angle in accordance with the set value.

Motion Mode

Sets motion mode of the AX-12A

- Wheel Mode
  Set CW and CCW limits to set the AX-12A to Wheel mode.
  In Wheel mode position value is automatically set to 0.
- Joint Mode
  In Joint mode when not in Wheel mode.

Temperature Setting

You can set the limit of the temperature for operation of the AX-12A

- Values saved even after powered off.
- You can check the actual temperature in accordance with the set value.
- When the actual temperature exceeds set temperature the LED will blink and the AX-12A automatically halts mechanical operations as a safety measure.
- Unless necessary it is recommended to leave temperature values to default value.

Present Temperature

Reads current temperature of the AX-12A

Voltage Setting

Sets the range of operating voltage limits of the AX-12A

- Values saved even after powered off.
- You can check the actual voltage in accordance with the set value.
- Can set min and max operating voltages.
- When the actual voltage exceeds the operating range the LED will blink and the AX-12A automatically halts mechanical operations as a safety measure.
- Unless necessary it is recommended to leave voltage values to default values.

Present Voltage

Reads current voltage of the AX-12A

Alarm and Shutdown Setting

Sets the LED Alarm/Shutdown to on/off in specific situations

- Values saved even after powered off.
- The AX-12A can be set to stop automatically through the Shutdown setting.
- Shutdown is useful to protect the AX12A from potential failures such as overheating, overload, etc.
- It notifies the user by flickering the LED or shutdown the movement in the following circumstances :

Instruction Error : Received a wrong command
Overload Error : Weight overload in AX-12A
Checksum Error : Checksum error in the received command
Range Error : Received a command that is out of the range of position value of AX-12A
Overheating Error : Current temperature exceeded the limit
Angle Limit Error : Out of position limit value
Input Voltage Error : Input Voltage exceeded the limit

Turn on Motor

Turns the torque of the AX-12A on/off

- You can easily turn on/off the torque with on/off command.
- If the value is 0, torque is off, if 1, torque is on.
- When target position has been set while torque is 0 the AX-12A will not move until torque is set to 1.

LED

Turns the LED of the AX-12A on/off

- If the value is 0, LED is off, if 1, LED is on.
- You can easily turn on/off the LED with on/off command.
- The LED cannot be manipulated when its activated from an alarm.

Compliance Margin

Sets compliance margin of the AX-12A

- The margin designates the area around the goal position that receives no torque.
- Set appropriate values for "Compliance slope," "Torque limit," and "Compliance margin," for smoother movements.

Compliance SlopeMargin

Sets compliance slope of the AX-12A

- Slope values created at both CW and CCW directions and output level set when approaching Goal position.
- When you set a lower value, it barely reduces the initial power as reaches to the goal position. When you set a higher value, it reduces the strength as it reaches the goal position.
- At a lower value, it resists with the maximum strength not to stray from the goal position.
- Even if you set a higher value, it will resist with more power if it is strayed too much from goal position.
- Compliance slope will change 7 representative steps according to input data. For example, an input of 25 will have a representative data of 16.

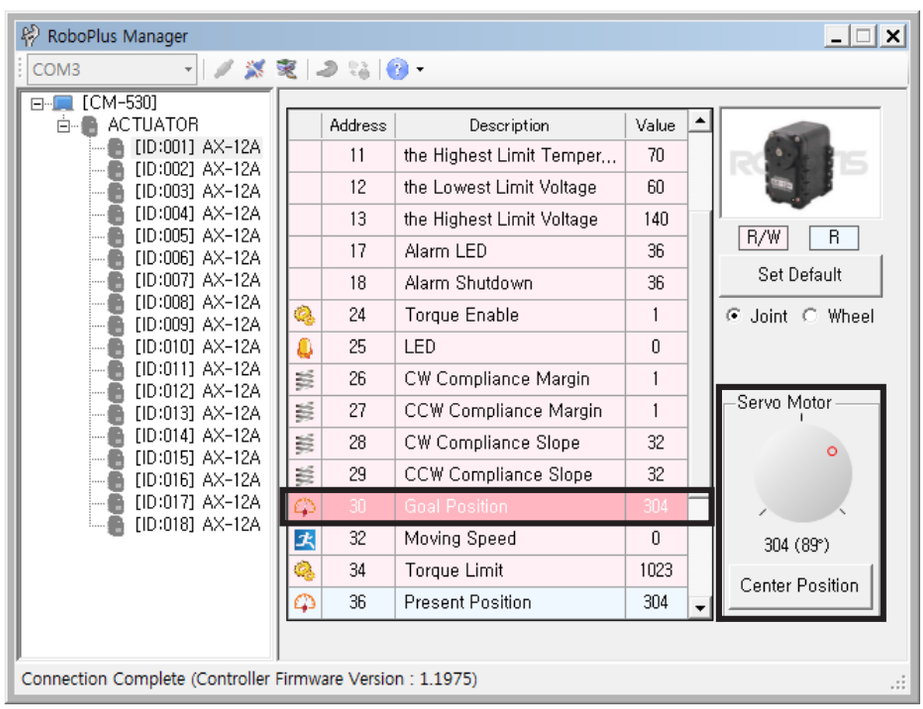
Steps	Data Value	Main Data Value
1	0 (0x00) ~ 3(0x03)	2 (0x02)
2	4(0x04) ~ 7(0x07)	4 (0x04)
3	8(0x08)~15(0x0F)	8 (0x08)
4	16(0x10)~31(0x1F)	16 (0x10)
5	32(0x20)~63(0x3F)	32 (0x20)
6	64(0x40)~127(0x7F)	64 (0x40)
7	128(0x80)~254(0xFE)	128 (0x80)

- Set appropriate values for the "compliance slope," "torque limit," and "compliance margin," for smoother movements.

Goal position

Sets Goal position of the AX-12A

- The value can be set using a jog dial.
- When this value is set, the actuator will immediately move to the goal position.
- The value is affected by "moving speed, position limitation," "compliance slope," and "compliance margin" parameters.
- When torque is off the AX-12A does not move; but, it will move as soon as torque is on.
- When the "center position" button is pressed, the value is set to the center position.



Moving Speed

Sets the speed of the AX-12A

- In joint mode, the speed is affected by the "compliance slope" and "compliance margin" values.
- In joint mode, set the value as 0 to output maximum power.
- In wheel mode (endless rotation mode), the speed and rotating direction depend on the "moving speed" value.
- The movement mode can be set using the "position limitation" parameter.
- The control used to set the speed is different for each mode.

Torque Limit

Sets the max load of the AX-12A

- The alarm LED may be triggered or movement may stop depending on torque limit, LED alarm, or shutdown settings.

Present Position

Reads the current position of the AX-12A

- Values change as the position of the AX-12A change.

Present Speed

Reads the current speed of the AX-12A

- Values change as the AX-12A moves.

Present Load

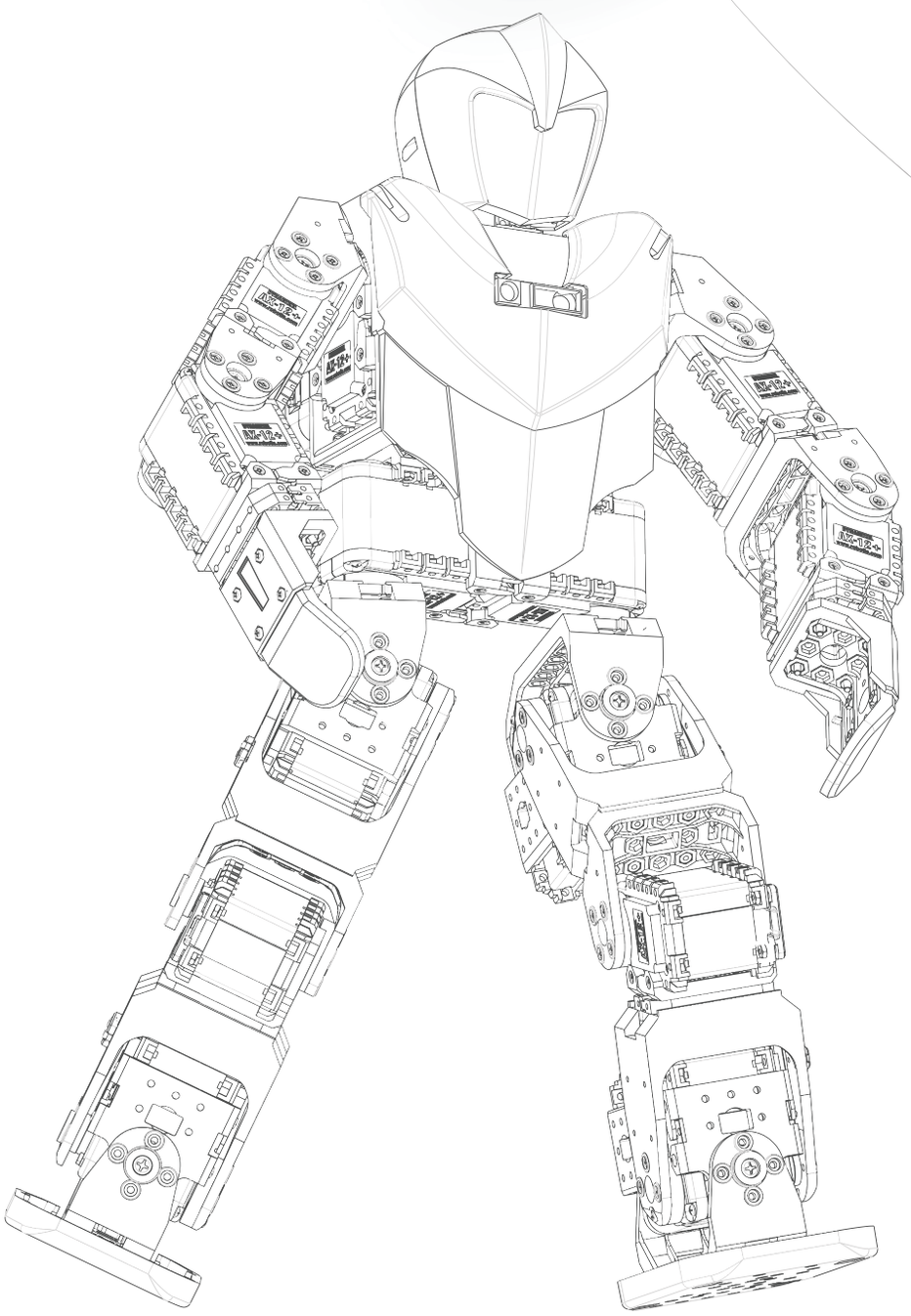
Reads the current load of the AX-12A

- Values change as the AX-12A moves
- Value not relevant when motor is turned off.

Moving

Determines whether the AX-12A is currently moving

- If the value is 0, it is not moving; if the value is 1, it is moving



3. RoboPlus Task

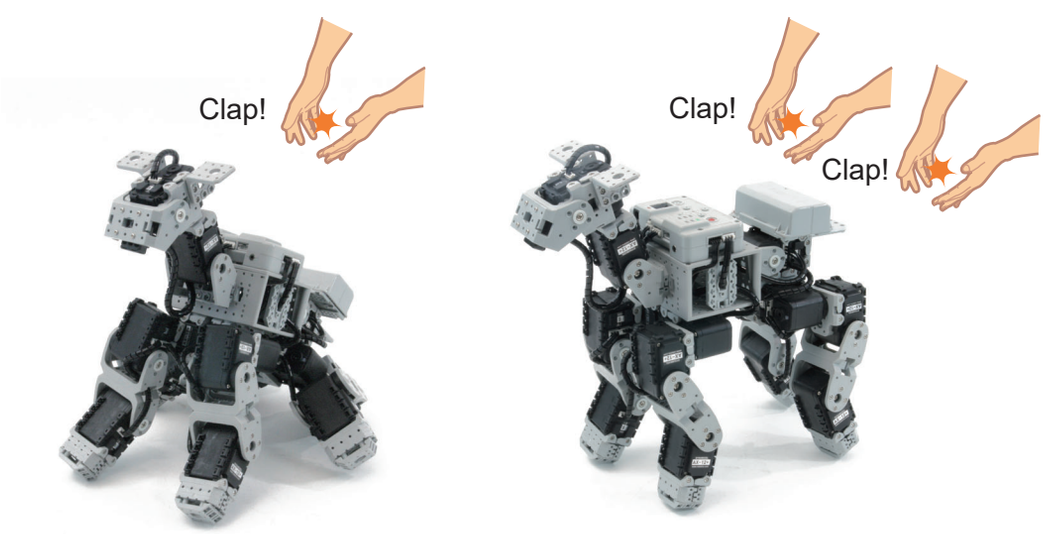
3-1 What is RoboPlus Task?

RoboPlus Task is a software that assigns certain robot behavioral actions given a condition. Task allows a convenient way to create behavioral code.

Input and Output

Connecting an input to an output is the basic function of Task. For example, a robot puppy stands up after one clap and sits down after two claps. The input parameter is the clapping sound and the outputs are sitting and standing movements.

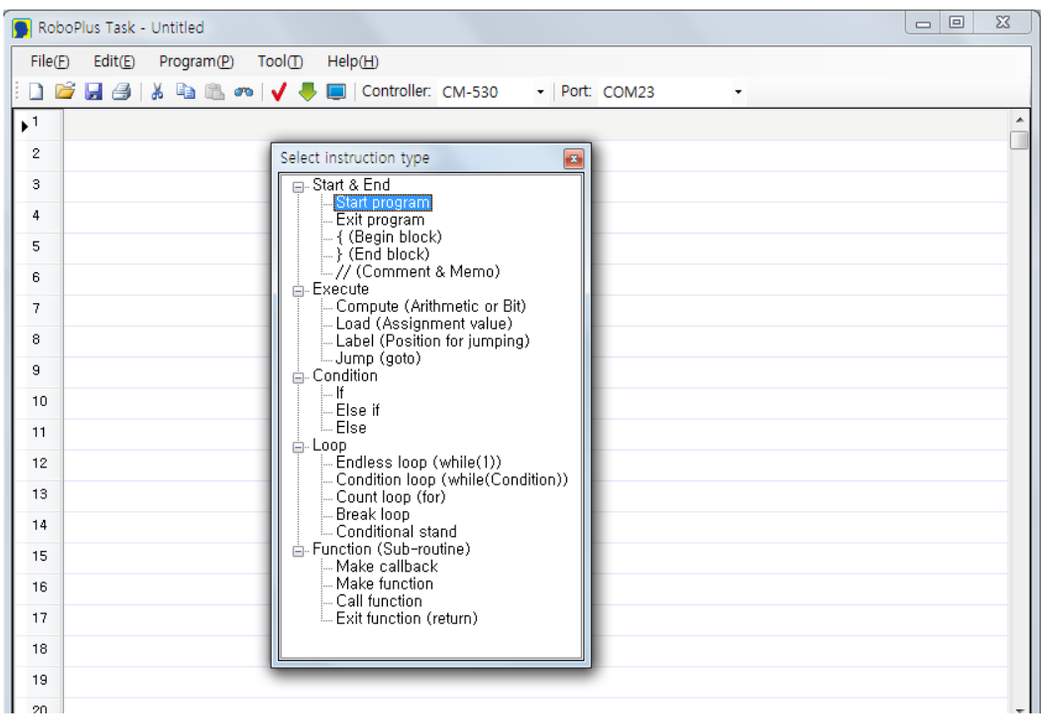
Therefore, to write a RoboPlus Task program, you need to know the input/output parameters. You can say learning RoboPlus Task program means learning these parameters. This chapter explains how to program with the parameters provided with ROBOTIS PREMIUM.



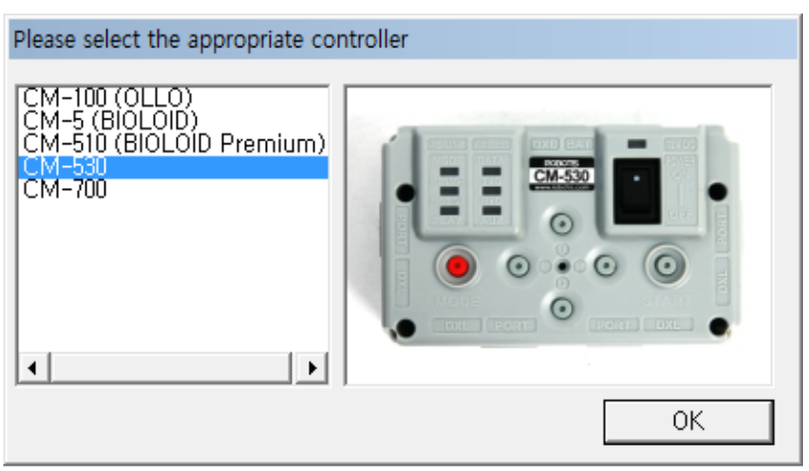
3-2 Getting Started

3-2-1 Create Command line

Double click on a blank line or click on a line and press enter. Choose a command from the list of commands supported by the selected controller.

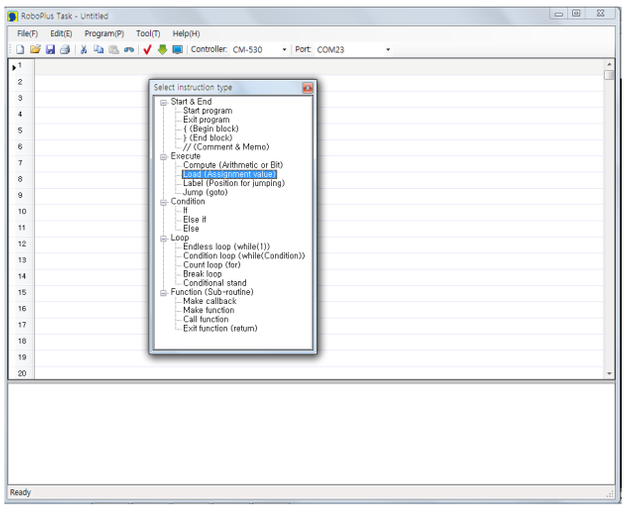


If the controller has not been selected yet, the program will ask you to choose the type of controller that will be used for the current program. Select CM-530.

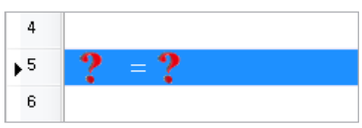


3-2-2 Create Parameter

"Parameter" refers to required fields or information to execute commands.

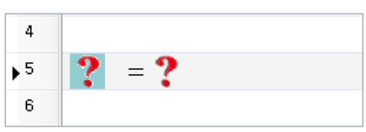


A question mark (?) indicates that a parameter has not been set.



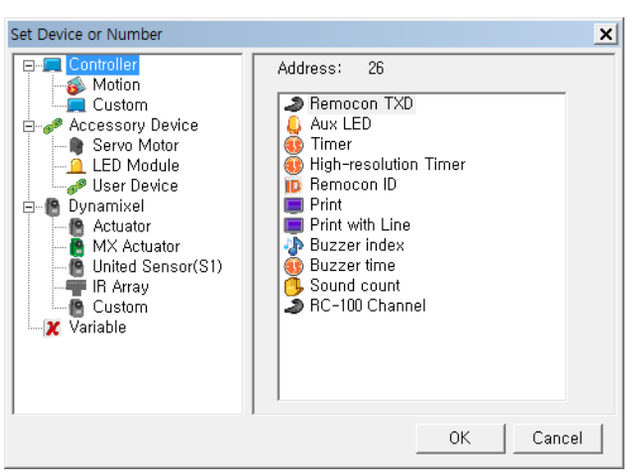
After selecting a command you must designate a parameter to complete the command line.

1. Go to 'edit mode' by double clicking the mouse or pressing the enter key.



2. Choose the parameter to create by pressing left/right arrow keys or by clicking on the question mark.

3. Press enter or double click to see the parameter selection window.



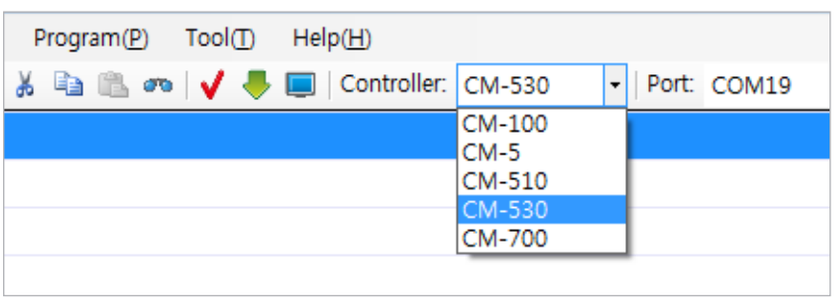
4. Choose the appropriate parameter. It is very important to learn and understand the functions of parameter.

3-2-3 Download

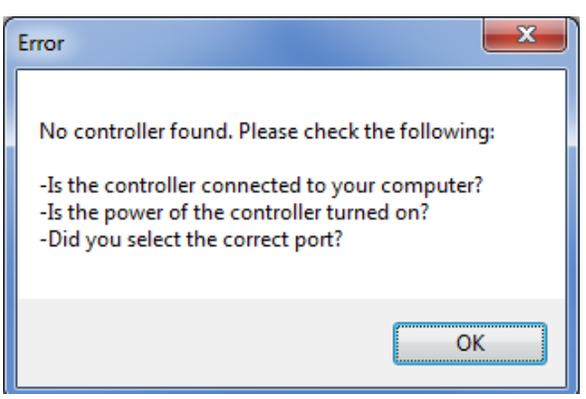
Download the Task code to the CM-530. Downloading the code once is enough and the Task code will reside in the CM530's memory until overwritten with a new code.

1. The controller must be connected to the PC.
   To download the task code the controller must be connected to the PC.

2. Select the correct communication port
   Use the "Automatic Search" function to easily find the appropriate port.

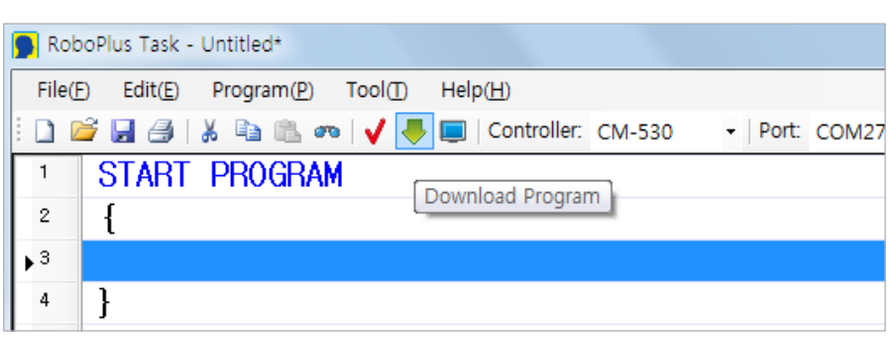


   If RoboPlus Task is unable to find a controller one of the following error messages will appear.



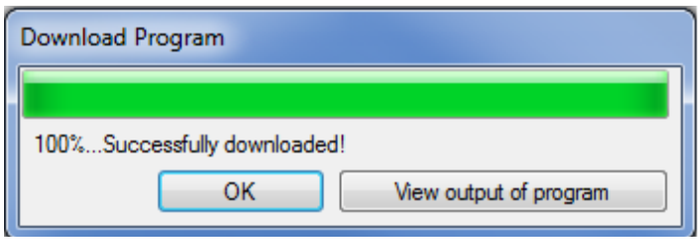
- Check if the CM-530 and PC are connected.
- Check if the CM-530 is turned on.
- Check if you selected the correct communication port.

3. Select the download menu.



If the program has an error find the error and correct it. See “rule check error messages”

4. Download the program.

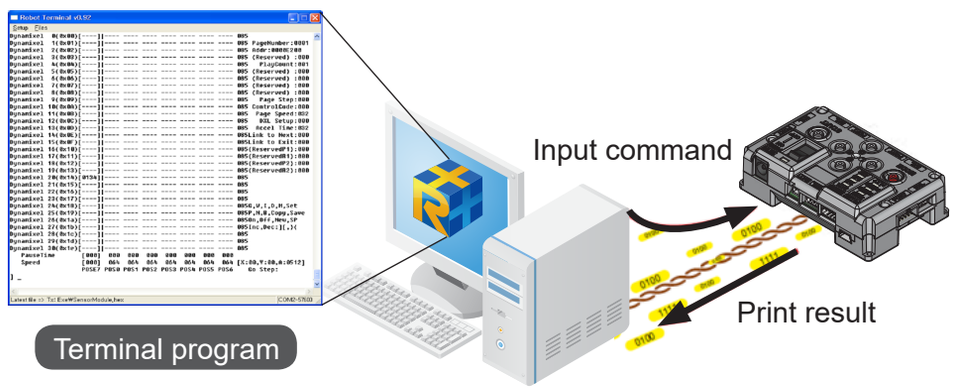


If the downloading fails it will automatically try again from the beginning.

5. Execute the task code → Your robot will move.
Turn on the controller and execute the downloaded task code.

3-2-4 Print Program Output

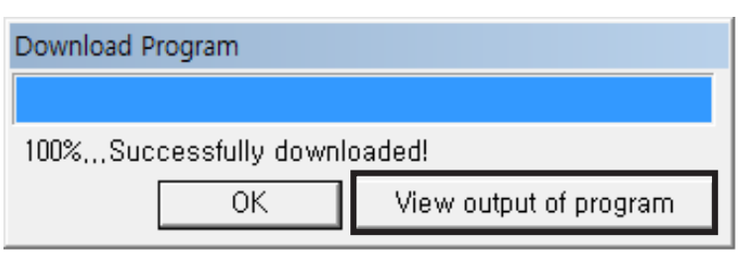
In general, the CM-530 does not have display devices the way a PC has a monitor, so it is hard to keep track of what goes on in a controller. Therefore, a "terminal" is used to "borrow" the PC's monitor.



Open the Program Output Monitor

Used to see the output of the program. Before executing the program open the program output monitor. There are three ways to open the program output monitor.

- Click the 'View Print of Program' on the Program download window.



- Click the "View Print of Program" button ( ) in the tool bar.
- Press F5 or click on "View Print of Program (V)" menu under Program (P).

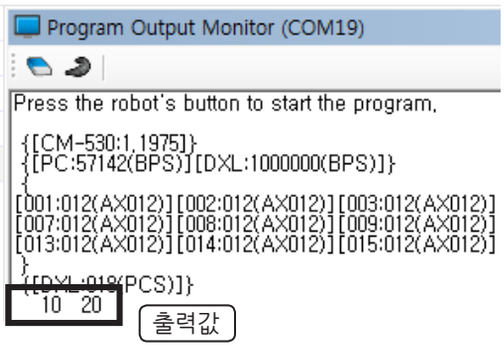
Use the "Print" parameter in your task code to see desired values on the screen.

```
1  START PROGRAM
2  {
3      Print  = ?
4
5      Print with Line  = ?
6  }
```

Print Screen

Print the value and then move the cursor to next line.

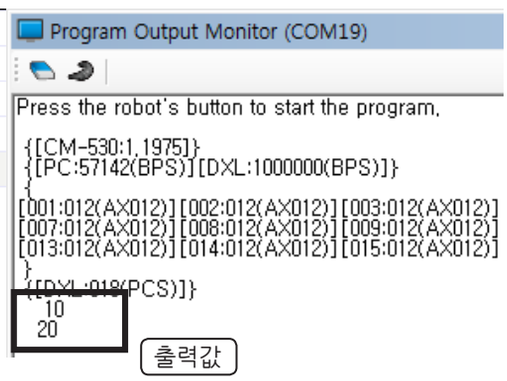
```
1  START PROGRAM
2  {
3      Print  = 10
4
5      Print  = 20
6  }
```



Change Line after Print Screen

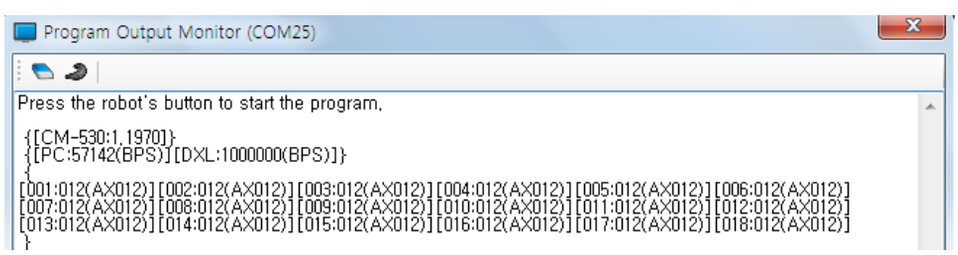
Print with New Line Print out the value and move the cursor to next line.

```
1  START PROGRAM
2  {
3      Print with Line  = 10
4
5      Print with Line  = 20
6  }
```

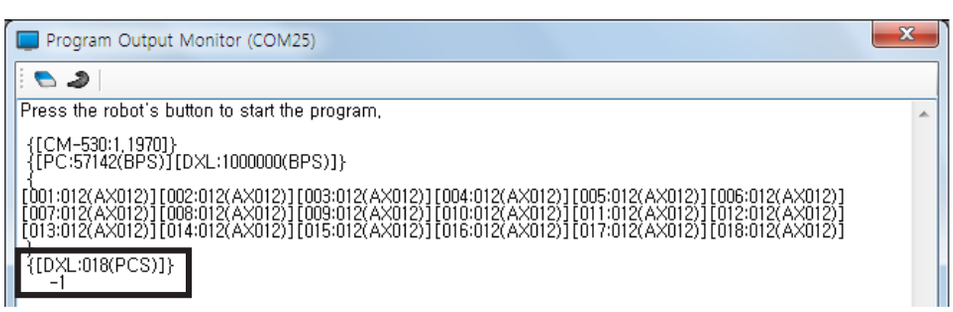


Print Screen Contents

- A welcome screen appears when the program starts



- Error m essages during program operation (See types of error messages)

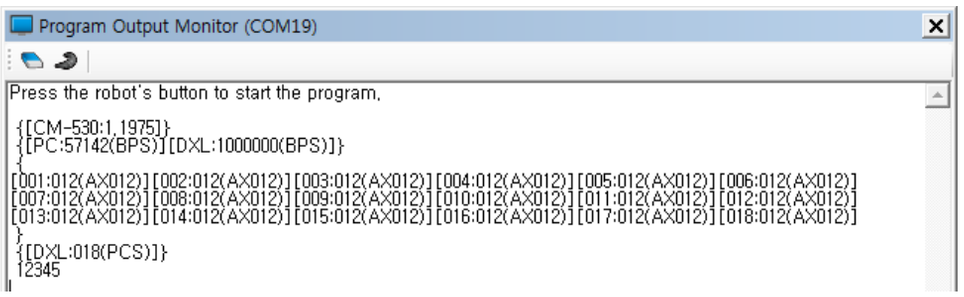


Output contents from the task code

All decimal numbers between -32767 and +32767 can be displayed.

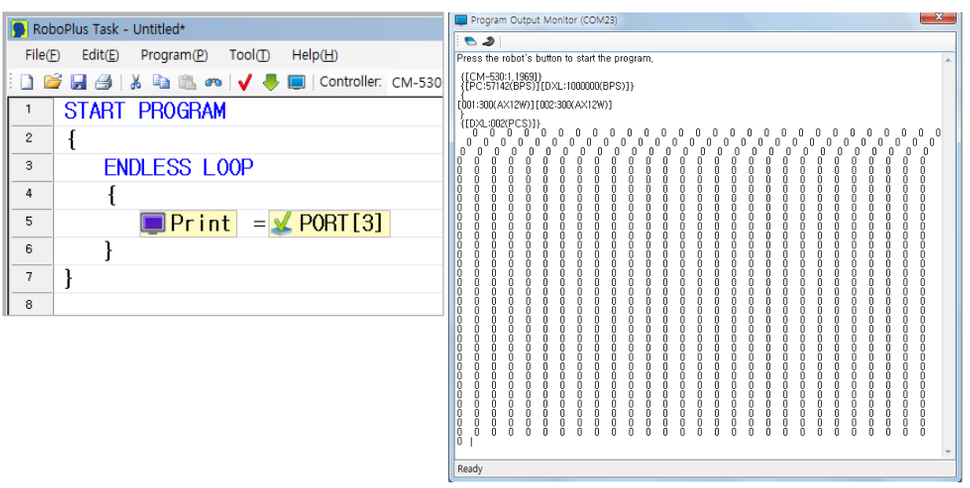
Printing characters or custom messages is not possible.

When printing numbers



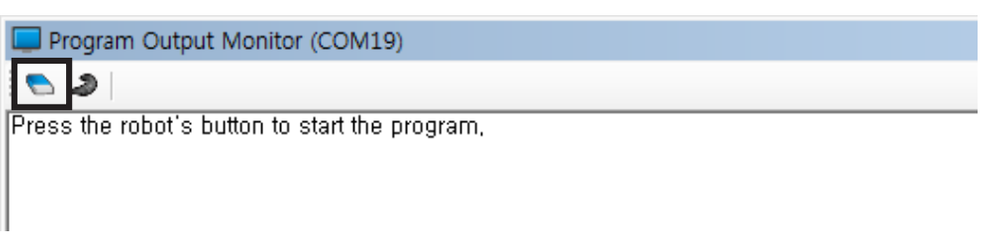
When printing sensors values

CM-530 out info



Clear Screen

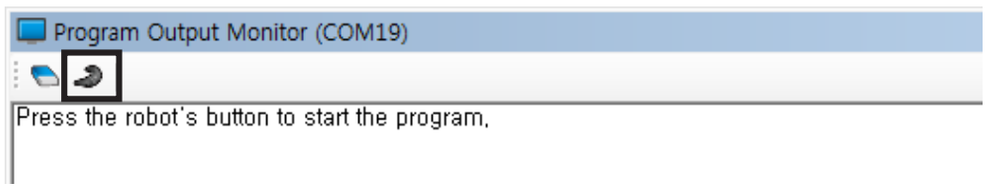
You may erase everything on the screen.

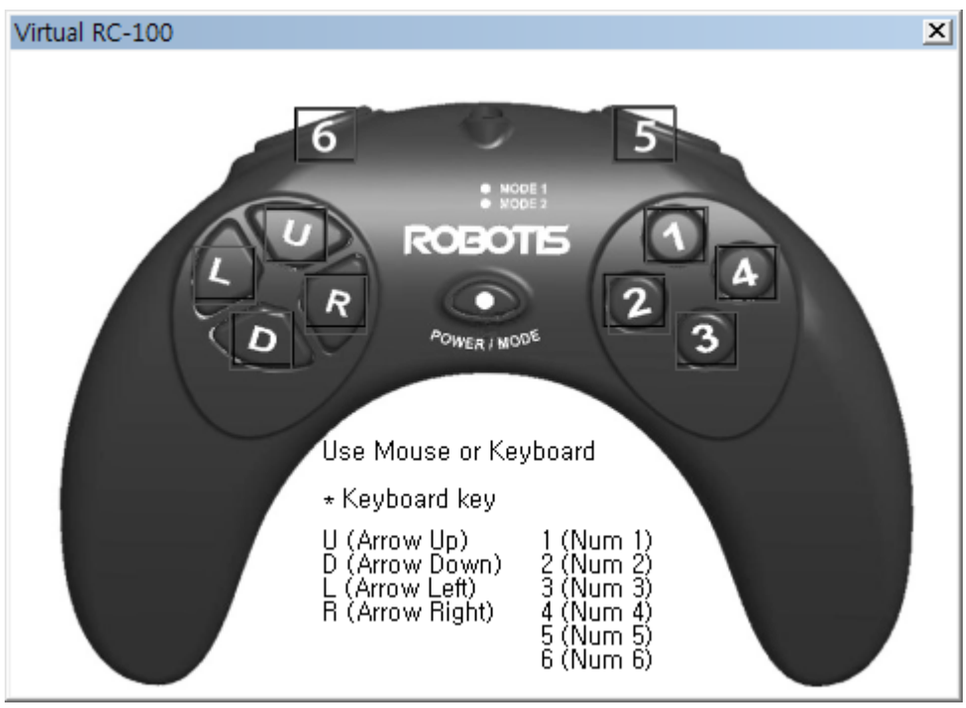


3-2-5 Virtual Robot Control

Virtual Robot Controller

RoboPlus Task supports virtual robot control, which makes controlling of robot possible without remote controller devices like RC-100B. Click the appropriate button with the mouse or press the appropriate key.





3-3 Programming 1

3-3-1 Edit

Below are the editting methods used in RoboPlus Task.

Select Multiple Lines

Used to select and edit (cut, copy, delete, comment, etc) multiple lines of program code.

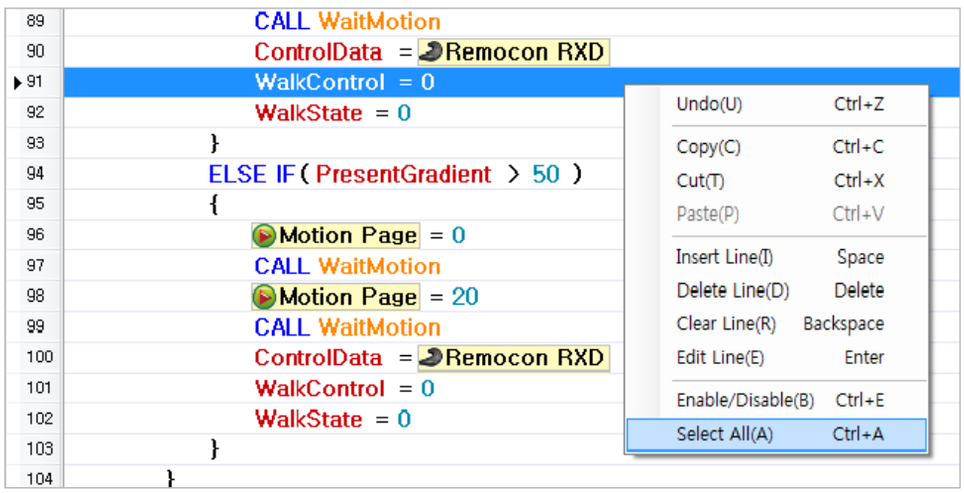
- While pressing the Ctrl key click on the lines with the mouse.

```
89          CALL WaitMotion
90          ControlData = Remocon RXD
91          WalkControl = 0
92          WalkState = 0
93      }
94      ELSE IF ( PresentGradient > 50 )
95      {
96          Motion Page = 0
97          CALL WaitMotion
98          Motion Page = 20
99          CALL WaitMotion
100         ControlData = Remocon RXD
101         WalkControl = 0
102         WalkState = 0
103     }
104 }
```

- Click on the first line, and while holding the Shift key click on the last line. The lines between the two lines will be selected. Or, click and drag on the lines you want to select.

```
88      {
89          CALL WaitMotion
90          ControlData = Remocon RXD
91          WalkControl = 0
92          WalkState = 0
93      }
94      ELSE IF ( PresentGradient > 50 )
95      {
96          Motion Page = 0
97          CALL WaitMotion
98          Motion Page = 20
99          CALL WaitMotion
100         ControlData = Remocon RXD
101         WalkControl = 0
102         WalkState = 0
103     }
104 }
```

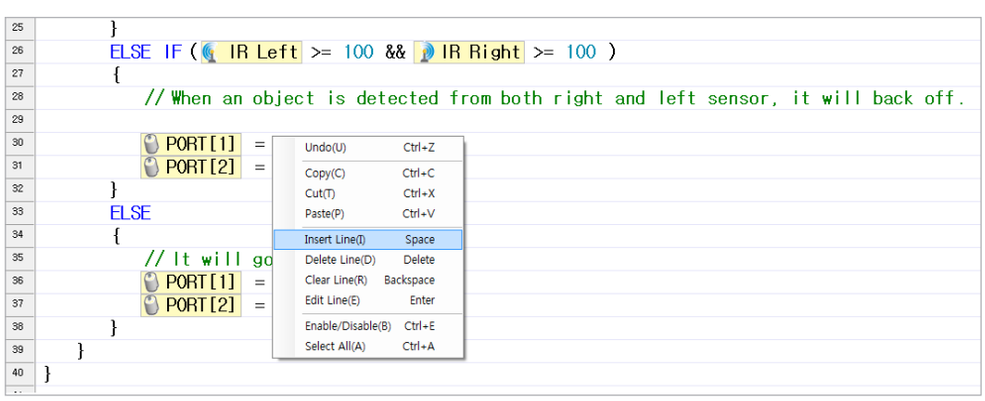
- To select all lines right-click on the code; then, click “Select All”
  You may also use the shortcut Ctrl + A keys.



Insert a New Line

Used to insert a line between existing lines. There are multiple ways to insert new lines. New lines will be inserted below the highlighted line.

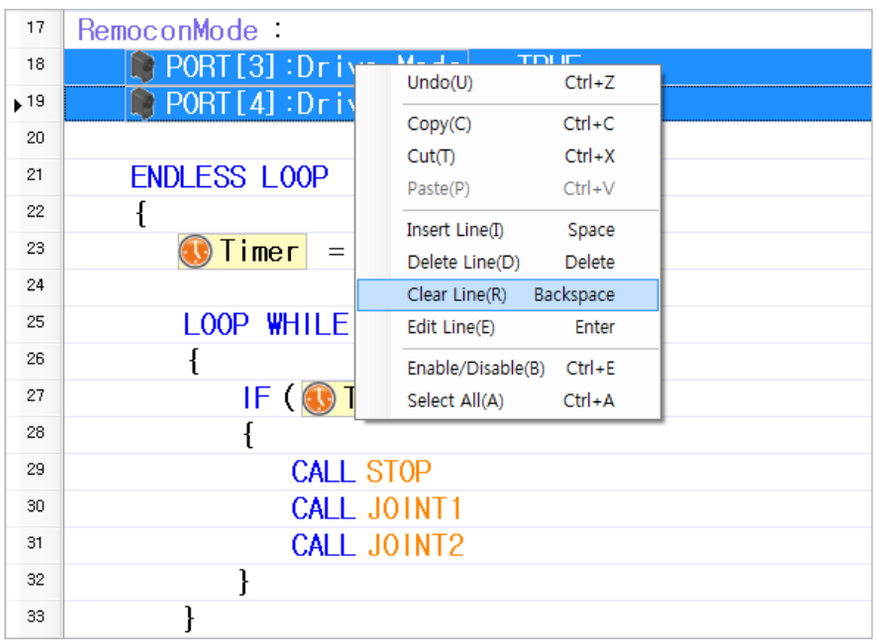
- Press the space bar.
- Right click then select "Insert Line"'.



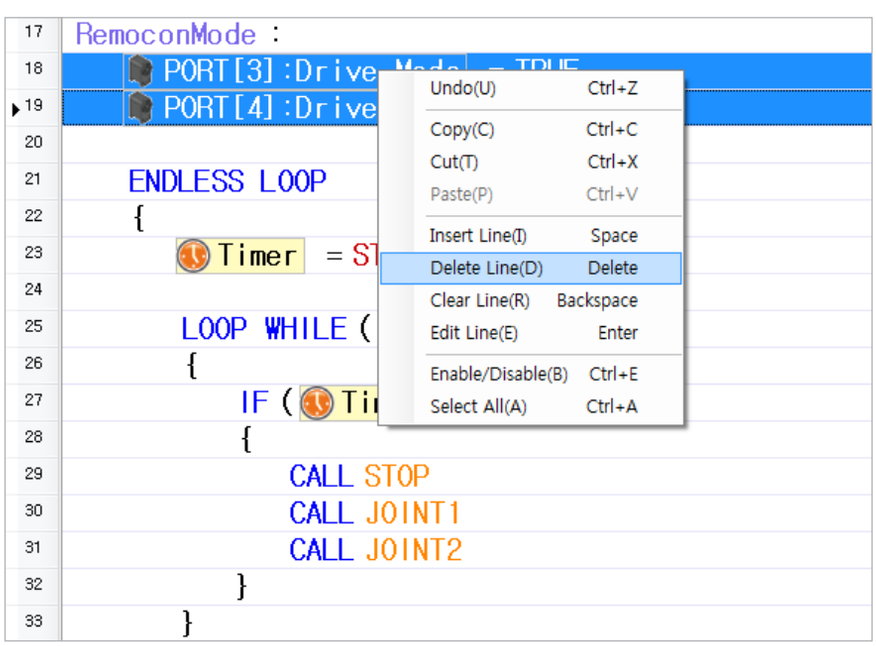
Delete Line

Used to delete one or more lines at once.

- Select the line(s) to delete, right-click, and click “Clear Line.” The lines will be cleared resulting in blank lines. You may also press the Backspace (←) key.



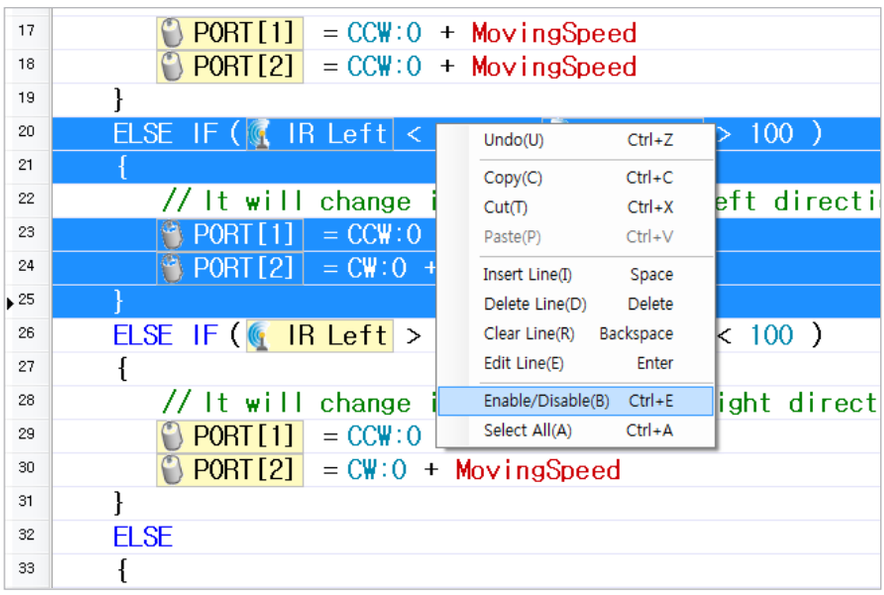
- Select the line(s) to delete, right-click, and click “Delete Line.” The lines will be deleted and the lines below the deleted lines will move up. You may also press the Delete key.



Enable/ Disable Lines

Used to enable or disable lines.

- Select the line to enable or disable, right-click, and click "Enable/Disable".
- Select the line to enable or disable and press Ctrl + E

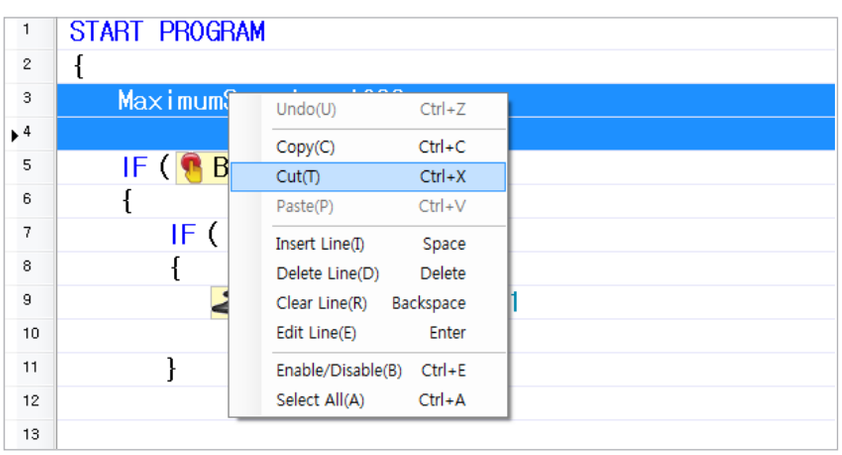


- This function will enable disabled lines.
- This function is commonly used to keep certain commands from being executed during certain situations, such as, when testing a code.

Cut

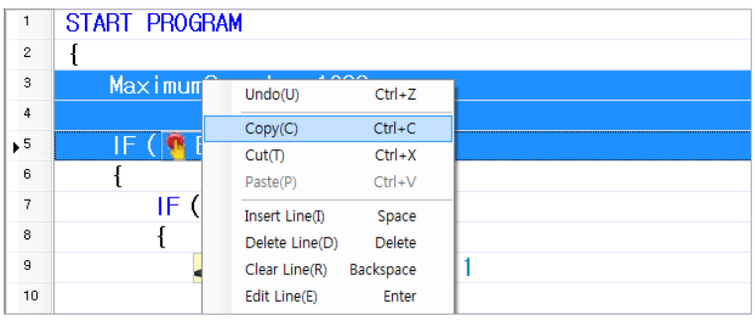
RoboPlus Task provides a function to cut copy and paste lines.

Select one or more lines, right-click, and click "Cut". The selected line will be deleted and stored in a temporary clipboard. When you perform the "Cut" function the data in the clipboard will be replaced with the new selection. The shortcut is Ctrl+X.



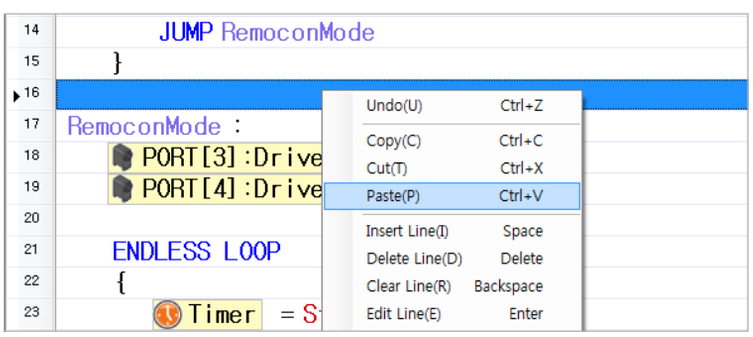
Copy

Select one or more lines, right-click, and click 'Copy'. The selected line will be copied to a temporary clipboard. When you perform the 'Copy' function the data in the clipboard will be replaced with the new selection. The shortcut is Ctrl+C.



Paste

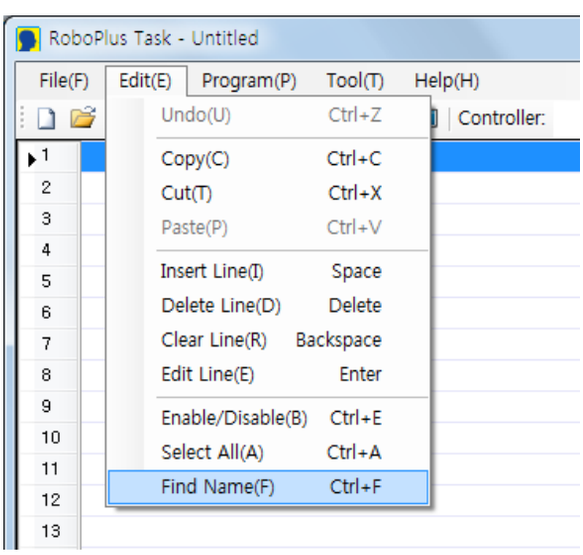
This function is used only when there is data in the clipboard. Select the line where the data will be pasted, right-click, and click "Paste." The data in the clipboard will remain even after it has been pasted, so you can paste the same data many times. If you perform the "Paste" function on a line with a code the code will be overwritten with the data from the clipboard. The shortcut is Ctrl+V.



Find Name

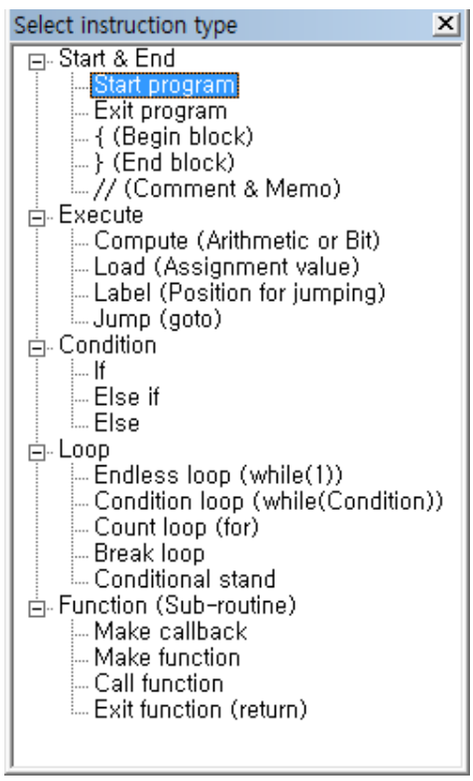
Used to search for used elements.

1. From the "Edit" menu, select the "Find Name" function.
   The shortcut is Ctrl+F.



2. Enter a keyword and click the "Find Next" button.

3-3-2 Type of Commands



Below are the types of commands used in RoboPlus Task.

Start Program

"Start Program" designates the beginning of a program. Regardless of the line number the program will always start from this point. "Start Program" is like the "main" function in the C language.

Usage

"Start Program" is executed regardless of its line number. A program can not have more than one "Start Program" command. The body of the command must be enclosed by brackets. The program will end when the closing bracket (}) is reached.

Example 1 Start a program with the "Start Program" command.

```
1  START PROGRAM
2  {
3
```

End Program

If this command is called during program execution the program exits immediately. There are 2 ways to end a program.

- When the end of "Start Program" is reached (Natural Close)

```
1  START PROGRAM
2  {
3
4
5
6  }
```

- When "End Program" is called (Forced Exit)

```
1  START PROGRAM
2  {
3
4      END PROGRAM
5
6  }
```

Example 2 In this sample code, the program will end when the touch sensor connected to Port3 is pressed.

```
IF ( PORT[3] == TRUE )
{
    END PROGRAM
}
```

Start/End of Block or Section

A block or a section (identified by "{" and "}") is a group of commands. All commands in a block have the same scope. The concept of a block is the same as in the C language.

Usage

- Each block has an opening bracket ({) and a closing bracket (}). RoboPlus Task performs automatic indentation to show whether the brackets have been paired properly. If any brackets are missing they must be added before the program can run the indentation. If they are not arranged properly you have to revise them by yourself.

```
1  START PROGRAM
2  {
3      ENDLESS LOOP
4      {
5
6      }
7  }
```

- Each block must be "owned" by a command. Blocks can not be used independently with commands. Below are the most commonly used commands that are followed by a block.

Start Program, If / Else if / Else, Endless Loop, Loop For, Loop While, Callback Function, Function

- If a command is made by one line it is possible to skip the block brackets.

```
IF ( Timer == 0 )
    Print = Timer
```

Example 3 The "Start Program" and "Endless Loop" commands must be followed by blocks, like in the example below.

```
START PROGRAM
{
    ENDLESS LOOP
    {
        Print = 10
    }
}
```

Comments or Notes

Used to insert a comment or note in the program code. Comments are helpful when interpreting or reviewing the code later. They are frequently used to mark easily forgotten parts or to emphasize important information. Comments and notes do not affect the program in any way. Like in C, comments can be made with two slashes (//). Comments blocks (/*, */) are not supported.

Usage

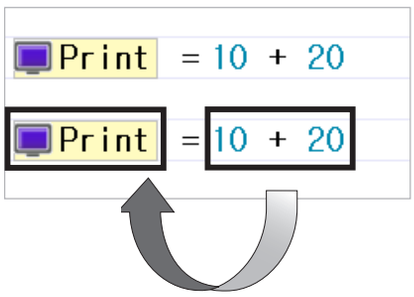
- Insert the comment where you would like to write a comment or note.
- When "//" is added double-click or press enter to write in the comment or note.
- Pressing ESC while writing will erase what has been written and return the line to its previous state.
- When done press Enter.

Example 4 This code will print "10" on the screen. The comment explains how the command line below will be executed.

```
// Print out the value, and move the cursor to next line.
Print with Line = 10
```

Calculate

Used to perform an arithmetic operation on two numbers. The meaning is as follows



Calculate can perform the following operations

- Basic Operations (supports negative numbers)

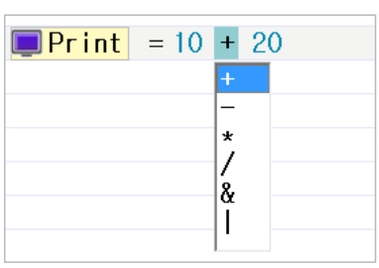
Addition (+)	Add two values.
Subtraction (-)	Subtract the second number from the first number.
Multiplication	Multiply two numbers.
Division (/)	Divide the first number by the second number. (Remainders are discarded.)

- Bit Operations (Means 2 decimal operation)

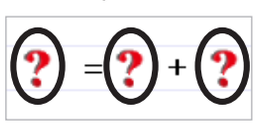
AND (&)	Perform a logical AND operation.
OR (|)	Perform a logical OR operation.

Usage

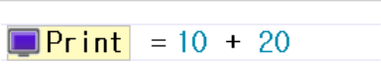
- You can choose an operator by double clicking the mouse or by pressing the enter key.



- Choose 3 appropriate parameters (result, operator 1, operator 2) necessary for the command.

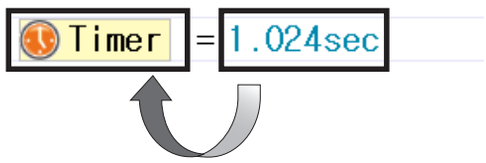


Example 5 This example shows how to add 10 and 20 and display the result on the screen.



Load

"'Load" is defined as "to place into an appropriate device." In RoboPlus Task "Load" places a value in a device.



"Load" is used to mean the following

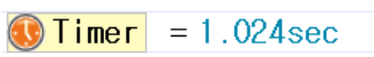
1. Execute a device's function.
2. Move a value.

Usage

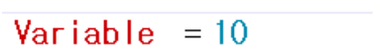
Choose 2 appropriate parameters (destination, source) necessary for the command.

Example 6 Examples for each

• To execute a device (Set the timer to 1.024 seconds)



• To set a value (Insert 10 into the variable)



Label/Jump

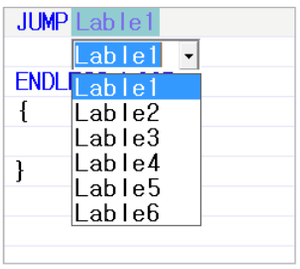
Used to branch a program. Branching is used to change the order of the program. "Jump" branches the program and "Label" designates where to branch to. This is the same function as "Label" and "Goto" in C language.

```
START PROGRAM
{
    JUMP JumpPoint

    ENDLESS LOOP
    {
        // Routine1
    }

JumpPoint :
    ENDLESS LOOP
    {
        // Routine2
    }
}
```

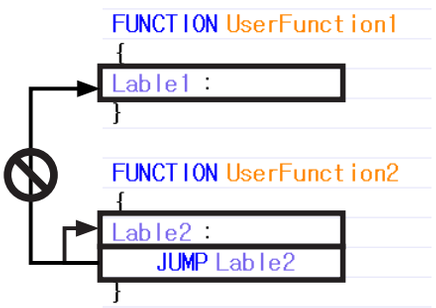
Usage

- Label names must abide by the following rules:
  - Can not have duplicate label names
  - Must exist within a program or function body
  - Can not jump to a label in another function
  - Can not have spaces and/or special characters (! @, #, $, etc.) for label names
  - Label names can not start with a number

- When inputting the label name press Esc to cancel.
- Otherwise, press enter to save
- When selecting the label to jump to press Esc to cancel.
- Otherwise, click the appropriate label or press enter while the label is highlighted to save.



- A jump can only be made to an existing label.
- The label must be in the same function block as the jump command.



Example 7 In this sample code, the program jumps to "JumpPoint" as soon as it starts and executes" 'Routine 2."

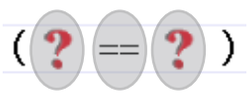
```
START PROGRAM
{
    JUMP JumpPoint

    ENDLESS LOOP
    {
        // Routine1
    }

JumpPoint :
    ENDLESS LOOP
    {
        // Routine2
    }
}
```

If/Else if

Used to branch the flow of the program depending on whether the condition is true or false.

- If : Execute if the clause is true. This is the equivalent to the "if" statement in C language.
- Else If : Execute if the clause is true and previous clause ("if" or "else if" clause) is false. This is the equivalent to the "else if" statement in C language.
- Else : Execute if none of the conditions are true. This is the equivalent to the "else" statement in C language.

- Conditional clause is composed of the following 3 parts: parameter 1, relational operator, and parameter 2 in order.



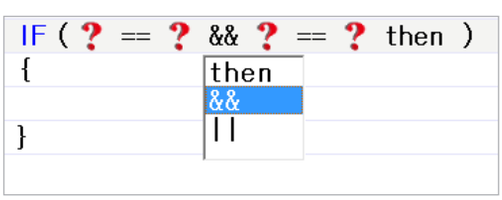
- These are 6 types of relational operators.

==	True if the two parameters are equal.
!=	True if the two parameters are not equal.
>=	True if parameter 1 is greater than or equal to parameter 2.
>	True if parameter 1 is greater than parameter 2.
<=	True if parameter 1 is less than or equal to parameter 2.
<	True if parameter 1 is less than parameter 2.

Conditional clause can be combined into a complex conditional clause using conditional operators.

A complex conditional clause is composed of the following 3 parts: conditional clause 1, conditional operator, and conditional clause 2.

- There are 3 types of conditional operators.

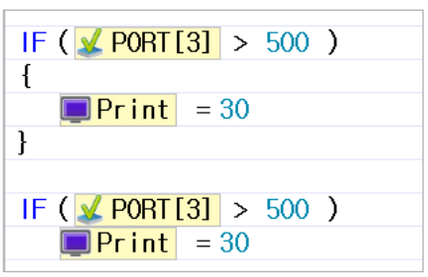


then	Does not link any clauses.
AND (&&)	True if both conditional clauses are true.
OR (||)	True if one of the conditional clauses is true.

There is no limit to how many conditional clauses can be combined into one complex conditional clause. Each conditional clause is evaluated in order and the final value will be either "true" or "false."

Usage

- An 'IF' command must always be preceded with an "Else if" or an "Else" command.
- A block, designated by brackets, needs to follow each clause. When the block consists of only one line, it does not need to be enclosed with brackets.



Example 8 The examples below show how to program the following conditions.

- When the variable is greater than or equal to 90.
- When the variable is greater than 80 and less than 90.
- When the variable is greater than 70 and less than 80.
- When the variable is greater than 60 and less than 70.
- Other cases

```
IF ( Variable  >= 90 )
{
    // When the variable is greather than or equal to 90
}
ELSE IF ( Variable  >= 80 && Variable  < 90 )
{
    // When the variable is greather than or equal to 80 and less than 90
}
ELSE IF ( Variable  >= 70 && Variable  < 80 )
{
    // When the variable is greather than or equal to 70 and less than 80
}
ELSE IF ( Variable  >= 60 && Variable  < 70 )
{
    // When the variable is greather than or equal to 60 and less than 70
}
ELSE
{
    // Other cases
}
```

Endless Loop

Used to repeat the command lines in the block without end.

Usage

- A block is always required. However, if the block consists of only one, it does not need to be enclosed with brackets.

```
ENDLESS LOOP
{
    Print  = 30
}
```

- Use the "Break Loop" command to exit the loop.

Example 9 Continuously prints "10" on the Program Output Monitor.

```
START PROGRAM
{
    ENDLESS LOOP
    {
        Print  = 10
    }
}
```

```
START PROGRAM
{
    ENDLESS LOOP
        Print  = 10
}
```

Loop While

Used to repeat the command lines in the block while the clause is true. It is the equivalent to the "while" function in C language.

Usage

- A block is always required. However, if the block consists of only one line, it does not need to be enclosed with brackets.

```
LOOP WHILE ( Timer > 0.000sec )
{
    Print  = 10
}

LOOP WHILE ( Timer > 0.000sec )
    Print  = 10
```

Use the "Break Loop" command to exit the loop.

Example 10 Continuously prints the value of on the Program Output Monitor until the variable reaches 30.

```
LOOP WHILE ( Variable <= 30 )
{
    Print  = Variable
    Variable  = Variable + 1
}
```

Loop For

Used to repeat the command lines in the block for the specified number of times. Given the initial value and a terminal value the loop will repeat while increasing the variable by 1.

```
LOOP FOR ( LoopValue  = 1  ~ 10  )
{
    Print  = 30
}
```

Number to Executions = Terminal Value - Initial Value + 1

This is the equivalent to the "for" function in C language.

Usage

- Choose 3 appropriate parameters (Variable, Start value, End value) necessary for the command

```
LOOP FOR ( ? = ? ~ ? )
{

}
```

- The initial value must be less than the terminal value. If the initial value is greater than the terminal value the loop will not be executed.
- A block is always required. However, if the block consists of only one line, it does not need to be enclosed with brackets.

```
LOOP FOR ( LoopValue  = 1  ~ 10  )
{
    Print  = 30
}

LOOP FOR ( LoopValue  = 1  ~ 10  )
    Print  = 30
```

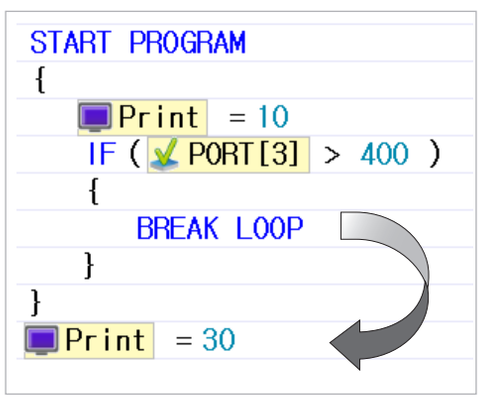
Use the "Break Loop" command to exit the loop.

Example 11 This example will repeat the loop as many as detected.

```
LOOP FOR ( Loop  = 1  ~ Sound count )
{
    // Repeatation
}
```

Break Loop

Used to exit the loop while it is being executed. It is the equivalent of the "break" function in C language.



Usage

- The command must always be used in the block being repeated.

Example 12 Continuously prints number "10" on the screen until the value of the center IR sensor becomes greater than 400 in which case exits the loop and prints "30" on the screen.

```
START PROGRAM
{
    Print = 10
    IF ( PORT[3] > 400 )
    {
        BREAK LOOP
    }
}
Print = 30
```

Wait While

This command is used to pause execution when a conditional clause is true. It is the equivalent of the "while" function in C language.

Usage

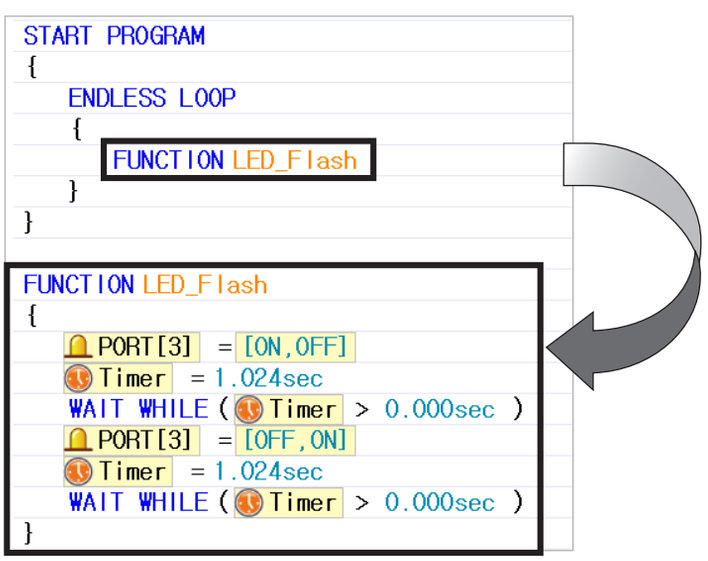
- Set the appropriate conditions without the use of blocks.

Example 13 The program will wait for the timer.

```
WAIT WHILE ( Timer > 0.000sec )
```

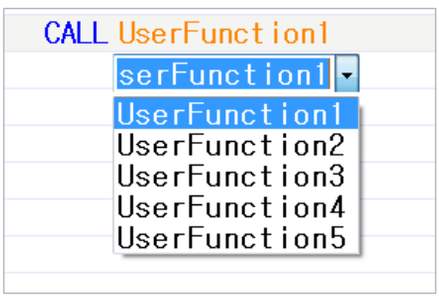
Making & Calling Function

If the same code needs to be repeated multiple times or if the code needs to be distinguished according to its role you can make the code as a function and can the function whenever necessary. This is similar to the concept of a function in C language. The only difference is that there are no return values and input parameters. When used properly you can easily figure out the flow of the program and avoid writing the same command lines over again. Functions are executed by calling them. After a called function ends, execution will be returned to the point of the calling and started from the next command line.



- The following rules apply when making a function:
  - Must not have duplicate function names.
  - Must exist outside another function or program body.
  - Must exist outside another function or program body.
  - Can not have spaces and/or special characters (! @, #, $, etc.) for label names
  - Label names can not start with a number

- While inputting function names, press Esc to cancel.
- Otherwise, press enter to save.
- While selecting the function to call, press ESC to cancel.
- Otherwise, click the appropriate function or press enter while the function is highlighted to save.



- A function cannot call on itself.

```
FUNCTION UserFunction1
{
    CALL UserFunction1
}
```

Example 14 The program will continuously call the functions to move forward, backward, right and left.

```
START PROGRAM
{
    ENDLESS LOOP
    {
        CALL Forward
        CALL Backward
        CALL TurnRight
        CALL TurnLeft
    }
}
```

Return Function

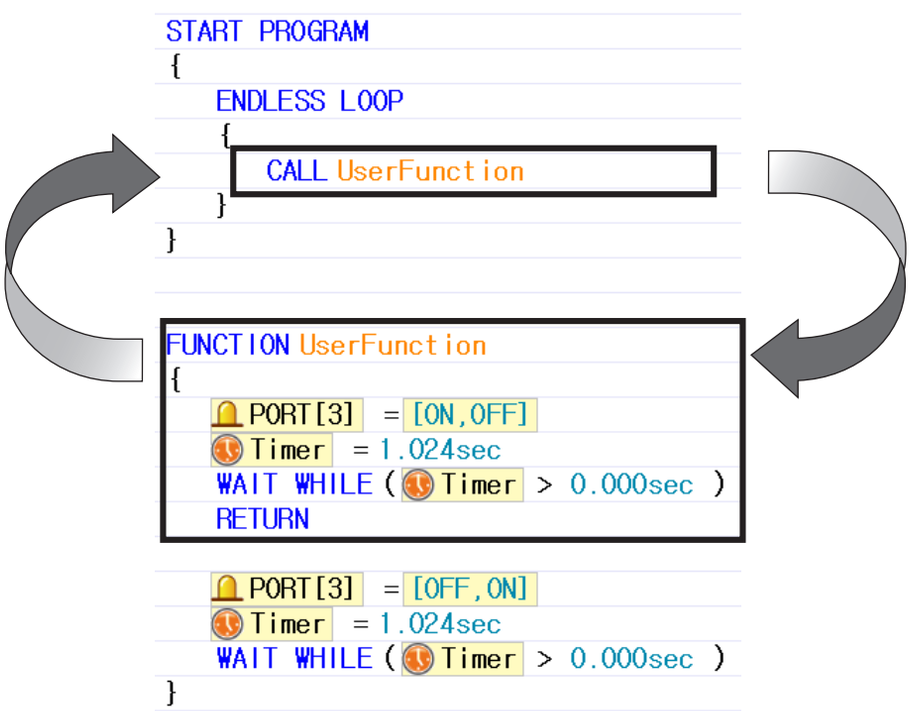
Used to end the operating function immediately even if every command line has not been executed and will return to the function called position. It is the equivalent to the "return" statement in C language.

Usage

- Can be used only within a general function or a callback function.

Example 15 In the following example, UserFunction is called repeatedly.

Because of the Return command in UserFunction the last 3 lines will never be executed.



Callback Function

Used to run independently from the main program routine and is automatically executed at fixed intervals. Therefore, a callback function can not include a code that requires much time and the use of loops, variables, and function calls are limited.

Usage

- The callback function cannot exist inside of another function or program body.
- There can be only one callback function.
- A callback function does not have a name and can not be called on.

Precautions

- Commands such as Endless Loop, Loop While, Loop For, Label, Jump, and Call Function can not be used.
- Communication between the controller external devices like AX-12A is limited to 2 times.
- Size of the command must not exceed 512 bytes

Example 16 This example shows how to save the periodically received wireless data in a variable in a callback function.

```
START PROGRAM
{
    ENDLESS LOOP
    {
        IF ( ReceivedData > 100 )
        {

        }
        ELSE
        {

        }
    }
}

CALLBACK
{
    IF ( Remocon Arrived == TRUE )
    {
        ReceivedData = Remocon RXD
        IF ( ReceivedData == 0 )
        {

        }
    }
}
```

3-4 Programming 2

3-4-1 CM-530

Below are the types of parameter used for the CM-530.

- Remocon RXD
- Remocon Arrived
- Aux LED
- Button
- Timer
- High-resolution Timer
- Remocon ID
- My ID
- Voltage
- Buzzer index
- Buzzer time
- Sound count
- Current sound count
- RC-100 Channel

Remocon TXD

Used to transfer data via wireless communication module (IR, zigbee module).

- The data must be a number/value between 0~65535 transferred or sent wirelessly (IR or Zigbee).
- When the "Remocon TXD" parameter is set the data is immediately sent wirelessly.
- It is also used to send a response to the control program on a PC connected using ZIG2Serial.

Example 17 In the example below, the program waits for data, and when the data arrives, the received data is transferred wirelessly.

```
START PROGRAM
{
    ENDLESS LOOP
    {
        WAIT WHILE ( Remocon Arrived == FALSE )
        Remocon TXD = Remocon RXD
    }
}
```

Remocon RXD

Used to read the received data received via wireless communication module (IR, zigbee module).

- The data is a number between 0 and 65535.
- The "Remocon Arrived" parameter can be used to check for new data.
- You can save up to maximum 2 wireless data by using a receiving buffer.
- When the 2 data are saved in the receiving buffer, the firstly received data will be read and then the remaining data will be read at according to the received order. If there is only 1 data value in the buffer when READ is executed the latest data will be retrieved.

Example 18 The code below shows how to control movement direction using the RC-100B.

```
START PROGRAM
{
    ENDLESS LOOP
    {
        WAIT WHILE ( Remocon Arrived == FALSE )

        IF ( Remocon RXD == U )
        {
            // This will be executed, if only the U button on the RC-100 is received.
        }
        ELSE IF ( Remocon RXD == D )
        {
            // This will be executed, if only the D button on the RC-100 is received.
        }
        ELSE IF ( Remocon RXD == L )
        {
            // This will be executed, if only the L button on the RC-100 is received.
        }
        ELSE IF ( Remocon RXD == R )
        {
            // This will be executed, if only the R button on the RC-100 is received.

        }
    }
}
```

Remocon Arrived

Used to check whether there are any new data received via wireless communication module (IR, zigbee module).

- The value is either TRUE or FALSE.
  - TRUE (No. 1) : There is a new data in the input buffer.
  - FALSE, (No. 0) : All data in the input buffer have been retrieved.
- Normally used to check whether the new data has been received to process.

Aux LED

Used to read and to set the CM-530's Aux LED status.

- The value is either TRUE or FALSE.
  - TRUE (1) : When the Aux LED parameter is set to TRUE, the LED will turn on. When the Aux LED parameter is read a value of TRUE signifies that the LED is on.
  - FALSE (0) : When the Aux LED parameter is set to FALSE, the LED will turn off. When the Aux LED parameter is read a value of FALSE signifies that the LED is off. False means the input buffer is empty. For example, either no data has been received or all data has been retrieved.

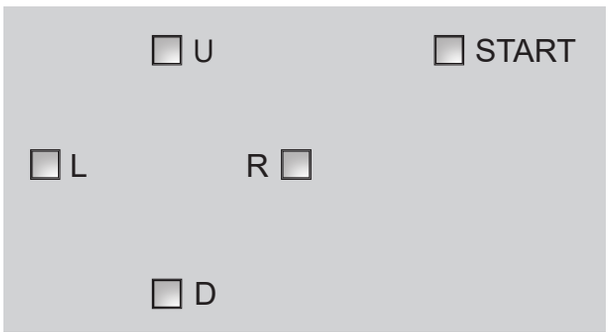
Example 19 In this example, the Aux LED is turned on and off for 1 second 3 times.

```
START PROGRAM
{
    LOOP FOR ( Count  = 1  ~ 3  )
    {
        Aux LED  = TRUE
        Timer  = 1.024sec
        WAIT WHILE ( Timer  > 0.000sec  )

        Aux LED  = FALSE
        Timer  = 1.024sec
        WAIT WHILE ( Timer  > 0.000sec  )
    }
}
```

Button

Used to read the controller's button status.

- A unique numeric value is assigned for each button and the codes value for each buttons are as follows.
- R button : 1, L button : 2, D button : 4, U button : 8, START button : 16
- When several buttons are pressed the value assigned to the pressed buttons are added and read.
- When the buttons' code values are unknown, this can be determined by using the buttons' constant values.



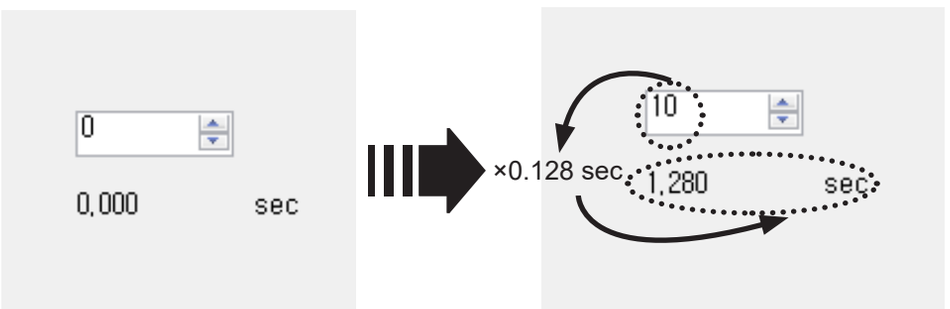
Example 20 This example shows how to perform different motions depending on which button is pressed.

```
START PROGRAM
{
    ENDLESS LOOP
    {
        IF ( Button == U )
        {
            // This will be executed, if only the U button on the controller is pressed.
        }
        ELSE IF ( Button == D )
        {
            // This will be executed, if only the D button on the controller is pressed.
        }
        ELSE IF ( Button == L )
        {
            // This will be executed, if only the L button on the controller is pressed.
        }
        ELSE IF ( Button == R )
        {
            // This will be executed, if only the R button on the controller is pressed.
        }
    }
}
```

Timer

Used to read the timer's current value or to set the timer, which begins to count down automatically. The timer is embedded in the controller.

- You can use "Timer value" constant to set the timer's value.



When a decimal number is entered it will automatically convert to the corresponding timer value.

- The actual timer value is between 0 and 255. Each timer value is 0.128 seconds.
- If you set a value greater than 0 the timer will start to count down every 0.128 seconds.

Example 21 The code below will print the value from the Center IR sensor every second.

```
START PROGRAM
{
    ENDLESS LOOP
    {
        Print with Line = PORT[3]
        Timer = 1.024sec
        WAIT WHILE ( Timer > 0.000sec )
    }
}
```

Remocon ID

Used to set or read the currently set remote control ID. Please note that the controller will not receive any data if this parameter does not match the ID of the transferring remote. *The function of the parameter is infrared. It cannot be used in Bluetooth module.

- The ID should be a number between 0 and 65535.
- When the opponent's ID is set to 65535 (0xFFFF, in hexadecimal) it will send data to all zigbee modules, regardless of their ID (Broadcasting Mode).
- For seamless zigbee communication, the opponent's wireless ID must be set to the correct value.
- Using the broadcasting mode improperly may cause unforeseen problems.

Example 22 This example sets the opponent's wireless ID to "123", reads the value, and prints to the screen.

```
1  START PROGRAM
2  {
3      Remocon ID = 123
4
5      Print with Line = Remocon ID
6  }
```

My ID

Used to read the ID of the Zigbee module installed in the robot. *The function of the parameter is infrared. It cannot be used in Bluetooth module.

If a Zigbee module is installed its ID is read (a number between 0 and 65534). If not, 65535 (0xFFFF in hexadecimal) is returned.

Example 23 This example prints the ID of zigbee module installed in the CM-530 on the monitor.

```
1  START PROGRAM
2  {
3      Print with Line = My ID
4  }
```

Buzzer Index

Used to set the musical note or melody to be played or to retrieve the note or melody currently being played using the buzzers in the controller.

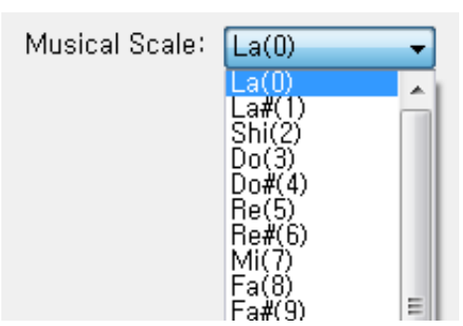
- The "Buzzer Time" parameter must always be used with the "Buzzer Index" parameter. "Buzzer Time" must be set before "Buzzer Index" is set.
- Depending on what the "Buzzer Time" is set to, the "Buzzer Index" can be set to play a musical note or a melody.

When "Buzzer Time" is set to 255 : Melody Mode

Melody Mode chooses from 16 different melodies (0~15)

When "Buzzer Time" is between 0 and 254 : Musical Note Mode

Musical Note Mode chooses from 27 notes. The selected notes will play for the length set as “Buzzer Time".



Buzzer Time (Controllers' Buzzers)

Used to set how long the note or melody will be played or to retrieve how much longer it will be played.

- The "Buzzer Time" parameter is always used with the "Buzzer Index" parameter. "Buzzer Time" must be set before "Buzzer Index" is set. This order is important.
- "Buzzer Time" can be set to a value between 0 and 255.
- "Buzzer Time" can be set to a value between 0 and 255.
- Each value represents 0.1 second. For example, when the "Buzzer Time" is set to 1, the note will play for 0.1 second. The maximum length a note will play is 5 seconds. Therefore, when values between 50 and 254 are entered, the note will play for 5 seconds.
- "Buzzer Time" can not be set while a note or melody is being played.

Example 24 This example is to play melody 3. Same as the example in "Buzzer Index".

```
START PROGRAM
{
    Buzzer time  = PlayMelody
    Buzzer index  = Melody3
    WAIT WHILE ( Buzzer time  > 0.0sec )
}
```

Example 25 Plays Do, Mi and Sol for 0.3 seconds each. Same as the example in "Buzzer Index".

```
START PROGRAM
{
    Buzzer time  = 0.3sec
    Buzzer index  = Do(3)
    WAIT WHILE ( Buzzer time  > 0.0sec )

    Buzzer time  = 0.3sec
    Buzzer index  = Mi(7)
    WAIT WHILE ( Buzzer time  > 0.0sec )

    Buzzer time  = 0.3sec
    Buzzer index  = Sol(10)
    WAIT WHILE ( Buzzer time  > 0.0sec )
}
```

Sound Count (Controller's Mic)

A controller equipped with a microphone has the function to count sounds when the sound is louder than a certain threshold. For example, it is possible to count claps. This parameter is used to retrieve the number of detected sounds.

- "Sound Count" uses the numbers between 0~255. As a result, the maximum number of sounds counted is 255.
- When no sound is detected, the number of detected sounds will be inputted into the "Sound Count" parameter.
- Number of detected sound is not automatically initialized - this needs to be manually set to 0 before you start.
- The geared motor connected to the controller may make loud noises while moving. The noise will be picked up by the microphone. Use the sound detection function only when the ROBOTIS PREMIUM has stopped moving completely.

Example 26 Detects sounds and repeats a specific motion for as many times as it is detected.

```
START PROGRAM
{
    ENDLESS LOOP
    {
        // The sound detection count will be initialized.
        Sound count = 0
        WAIT WHILE ( Sound count == 0 )

        LOOP FOR ( Count = 1 ~ Sound count )
        {

        }
    }
}
```

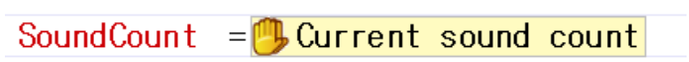
Current Sound Count (Controller's Mic)

A controller equipped with a microphone has the function to count sounds when the sound is louder than a certain threshold. For example, it is possible to count claps. This parameter is used to retrieve the number of detected sounds in real-time.

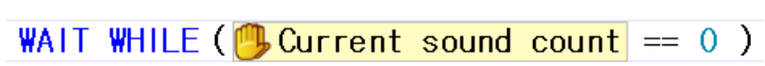
- "Current Sound Count" uses the numbers between 0~255. As a result, the maximum number of sounds counted is 255.
- Every time it detects a noise, the parameter value increases in real time.
- If a new sound is not detected for 0.8 seconds the value of the "Current Sound Count" parameter is passed over to the "Sound Count" parameter and the "Current Sound Count" parameter is reset to 0.

Example 27 Examples for each

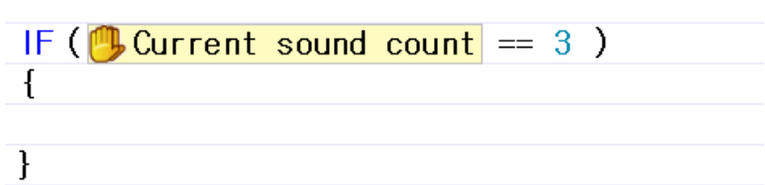
• This code saves the current sound count in the "SoundCount" variable.



• This code pauses the program when no sounds are detected.



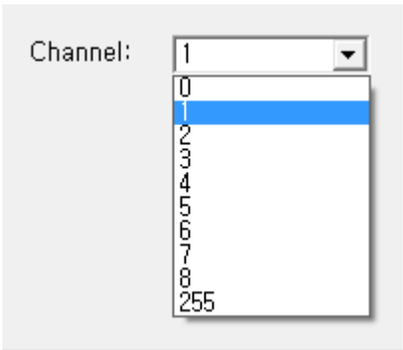
• This code executes a block of code when 3 sounds are detected.



RC-100B Channel

Used to set up the infrared communication channel or to check the current channel between the CM-530's IR receiver and the RC-100B.

- The RC-100B channel can be set using constant numbers.



- "RC-100B Channel" uses numbers between 0 and 8.
- Channel 0 is a special channel that allows communication with other channels.
- For smooth infrared wireless communication please refer to the section on how to set the channel for the RC-100B, make sure that both the RC-100B and the controller's IR receiver are set to the same channel.

Example 28 Sets the RC-100B channel according to how many times the START button was pressed.

```
1  START PROGRAM
2  {
3
4      IF ( Button <= 8 )
5      {
6          RC-100 Channel = Button
7      }
8
9      ELSE
10     {
11         RC-100 Channel = CH8
12     }
13 }
14
15
16
17
```

3-4-2 AX-12A

Below are the parameters used for AX-12A.

Torque Enable

Used to turn on/off the motor's torque and also used to determine whether the motor's torque is currently on or off.

- The value is either TRUE or FALSE.

- TRUE (1) : When set to TRUE the motor's torque turns on. When the parameter is being read a value of TRUE signifies that the motor's torque is on.
- FALSE (0) : When set to FALSE the motor's torque turns off. When the parameter is being read a value of FALSE signifies that the motor's torque is OFF.

Example 29 When the R button of the controller is pressed, the actuator with ID 1 will turn on. When the L button is pressed, it will turn off.

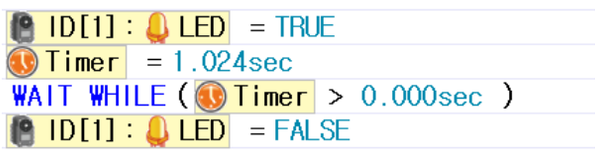
```
IF ( Button == R )
{
    ID[1] : Torque Enable = TRUE
}
ELSE IF ( Button == L )
{
    ID[1] : Torque Enable = FALSE
}
```

LED

Reads and sets status of the AX-12A

- The value is either TRUE or FALSE.
  - TRUE (1) : When set to TRUE the LED turns on. When the parameter is being read a value of TRUE signifies that the LED is on.
  - FALSE (0) : When set to FALSE the LED turns off. When the parameter is being read a value of FALSE signifies that the LED is off.

Example 30 Turns the LED on for 1 second and turns it off.

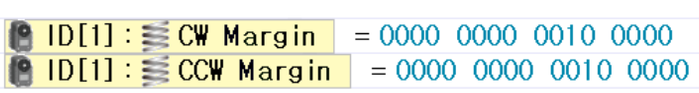


Compliance Margin

Sets or reads margin value of the AX-12A

- Margin values are numbers between 0 and 254.
- The margin designates the area around the goal position that receives no torque.
- The recommended value is 1. Unless otherwise specified use the recommended value.

Example 31 Set both margins as 1.



Compliance Slope

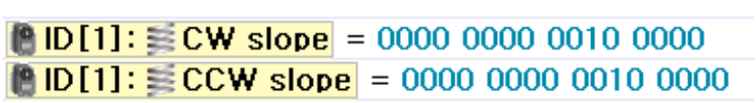
Sets or reads slope value of the AX-12A

- The slope value will be created at both CW/CCW directions, and the output level will be set near the goal position.
- When you set a lower value it barely reduces the initial power as reaches to the goal position. When you set a higher value it reduces the strength as it reaches the goal position.
- At a lower value it resists with the maximum strength not to stray from the goal position.
- Even if you set a higher value it will resist with more power if it is strayed too much from goal position.
- Compliance Slope will change to level 7 data representative values according to the input data. In other words if you input 25, in real operation, 16 the representative value will be used.

Level	Data Value	Representative Data Value
1	0 (0x00) ~ 3(0x03)	2 (0x02)
2	4(0x04) ~ 7(0x07)	4 (0x04)
3	8(0x08)~15(0x0F)	8 (0x08)
4	16(0x10)~31(0x1F)	16 (0x10)
5	32(0x20)~63(0x3F)	32 (0x20)
6	64(0x40)~127(0x7F)	64 (0x40)
7	128(0x80)~254(0xFE)	128 (0x80)

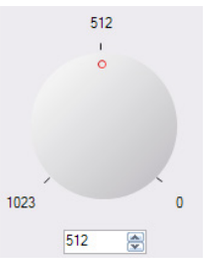
Set appropriate values for "Compliance slope," "Torque limit," and "Compliance margin" for smoother movements.

Example 32 Sets both slope values to 32. Binary numbers are used to set the parameter.



Goal Position

Sets or reads Goal position value of the AX-12A



- Position constants can be used. (0~1023)
- You can input position values directly or use jog dial to set the desired goal position.

Example 33 When the R button of the CM-530 is pressed, the goal position of the AX-12A with ID 1 will be set to 1.

```
IF ( Button == R )
{
    ID[1] : Goal position = 0
}
ELSE IF ( Button == L )
{
    ID[1] : Goal position = 1023
}
```

Moving Speed

Sets or reads speed value of the AX-12A

- Motor control constants can be used.
- In joint mode the direction value is meaningless ; only the power value will be used.
- In joint mode, set the value to 0 to output at maximum power.
- In endless rotation mode, the direction and power values must be set together
- Joint mode or endless rotation mode can be set using at RoboPlus Manager.

Example 33 When the R button of the controller is pressed, the speed of the actuator with ID1 will be set to 0. When the L button is pressed, the speed will be set to 500.

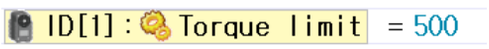
```
IF ( Button == R )
{
    ID[1] : Moving speed = CW:0
}
ELSE IF ( Button == L )
{
    ID[1] : Moving speed = CW:500
}
```

Torque Limit

Sets or reads power status of the AX-12A

- Torque limit values are numbers between 0 and 1023.

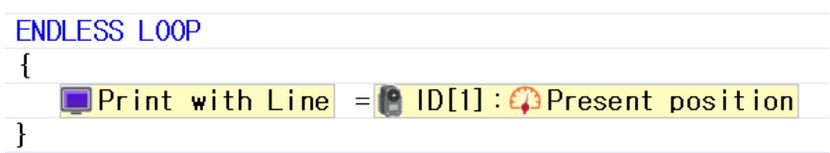
Example 35 Set the maximum torque of the actuator with ID 1 to 500.



Present Position

Sets or reads current position of the AX-12A

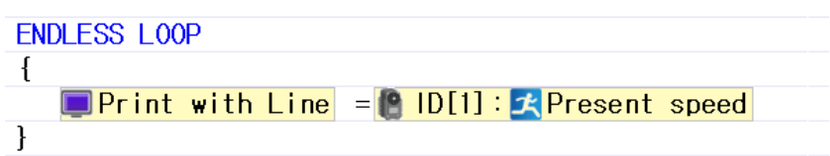
Example 36 Prints the current position of the AX-12A with ID 1 on the screen.



Present Speed

Sets or reads current speed of the AX-12A

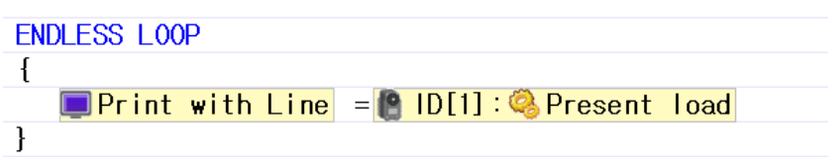
Example 37 Prints the current speed of the AX-12A with ID 1 on the screen.



Present Load

Sets or reads current load of the AX-12A

Example 38 Prints the current load of the AX-12A with ID 1 on the screen.



Voltage

Reads current voltage of the AX-12A

- The actual voltage is 1/10 of the read value. For example, if the returned value is 115 the actual voltage is 11.5V.

Example 39 Prints the current voltage of the AX-12A with ID 1 on the screen.

```
ENDLESS LOOP
{
    Print with Line  = ID[1] : Voltage
}
```

Temperature

Reads current temperature of the AX-12A

Example 40 Prints the current temperature of the AX-12A with ID 1 on the screen.

```
ENDLESS LOOP
{
    Print with Line  = ID[1] : Temperature
}
```

Moving or Not

Determines whether the AX-12A is moving.

- If it is moving 1 is returned. If it is not 0 is returned.

Example 41 Sets the goal position of the actuator with ID 1 as 0, and waits for it to stop moving.

```
ENDLESS LOOP
{
    ID[1] : Goal position  = 0
    WAIT WHILE ( ID[1] : Moving  == TRUE )
}
```

Direct Access

Accesses the address of devices (i.e. Dynamixel))then can read from and write to these devices.

- Data can be read from or written to a specific in the form of bytes or words.

Example 42 Examples for each.

• Write 0 at word address 25 of the AX-12A with ID 105.

• Print the value stored in the word address 25 of the AX-12A with ID 105.

3-4-3 Peripheral Devices

IR sensor

Reads value of the IR sensor module

- IR sensor values are numbers between 0 and 1023.
- For objects with the same or similar color, the closer it is, the higher the value (closer to 1023), and the farther away it is the lower the value (closer to 0).
- For objects with the same distance the lighter colored object will give a higher value; the darker object will give a lower value.

Example 43 This example executes a block of code if the value of the IR sensor connected to Port 3 is less than 500.

```
IF ( PORT[3] < 500 )
{

}
```

Distance Measurement Sensor

Reads value of the DMS

- DMS sensor values are numbers between 0 and 1023.
- For objects with the same or similar color the closer the object the higher the value; the father the object the lesser the value.
- Unlike IR sensors, DMS sensors are hardly affected by colors.

Example 44 This example executes a block of code if the value of the DMS sensor connected to Port 3 is less than 500.

```
IF ( PORT[3] < 500 )
{

}
```

User's Devices

Sets or reads values of user device.

- Although the same address is used to read and to write values, the actual pin ports are different. Check the pin port information of the User's devices.
- Once it reads the value from the user's device it prints the value of the voltage level of the input port. The value is in between 0 and 1023.
- When the user's device is set to 1 it delivers 5V to the output port.
- Setting the user's device to 1 does not guarantee that the obtained value from the user's device will be 1.
- Constant values for ports can be used to print the value of the user's device.
- Use the User's device when balancing/restoring Gyro Sensor.

Example 45 After setting the user's device at Port 3 to 1 (high), if the value read from Port 3 is less than 500, the output port is set to 0 (low).

```
PORT[3] = HIGH
IF ( PORT[3] < 500 )
{
    PORT[3] = LOW
}
```

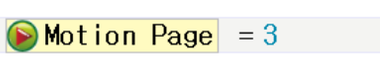
Motion Parameter is one of the most frequently used parameters for the output.

Motion Page

Execute motions.

- When the motion page number is entered the corresponding motion is executed.
- "Motion Page" can be read to see which motion is currently being executed.
- Certain page numbers can be used to stop the current motion.
- When the stop command is executed "Number of Page Repeats" will be ignored.
- To confirm that a motion has stopped completely check the motion status.
- When "Motion Page" is set to 0 the controller will execute to the exit page and stop.
- When "Motion Page" is set to -1 the controller will execute to the current page and stop
- If a page with no motions is set an error message will be returned.
- "Motion Page" uses numbers between 1 and 255.

Example 47 Executes motion page #3.

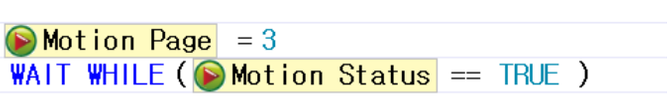


Motion Status

Used to check the status of the motion.

- If a motion is being performed 1 is returned; Otherwise, 0 is returned.
- True/False can be used.
  - True: Motion is being performed.
  - False: Motion is not being performed.

Example 48 Executes motion page #3 and waits until it is completed.



Joint Offset

Joint offset is applied when motion is performed. This parameter can be applied to each joint separately.

- 255 ~ 255 : The offset will be applied to the selected joint's location value.
  i.e.) If the location value of the joint with ID #3 is set to 300 → 400 → 500 in the motion data and the joint offset is -100; the actual location value of the joint will be adjusted to 200 → 300 → 400.
- 1024 : If the joint offset is set to 1024 the selected joint will not be affected by the motion data during operation.
  i.e) This function is used to control the location values directly, instead of the motion. A primary example is the gripper which should not move when a motion is being performed.

Example 49 Examples for each

Set an offset value to a specific joint : While motion page #3 is being performed, set the offset of the joint with ID 4 to -100, and wait for the motion to finish.

```
ID[4]:Joint offset = 100
Motion Page = 3
WAIT WHILE ( Motion Status == TRUE )
```

Keep a joint from being affected by motion data. Set up joint ID #4 to be unaffected while a motion is being executed.

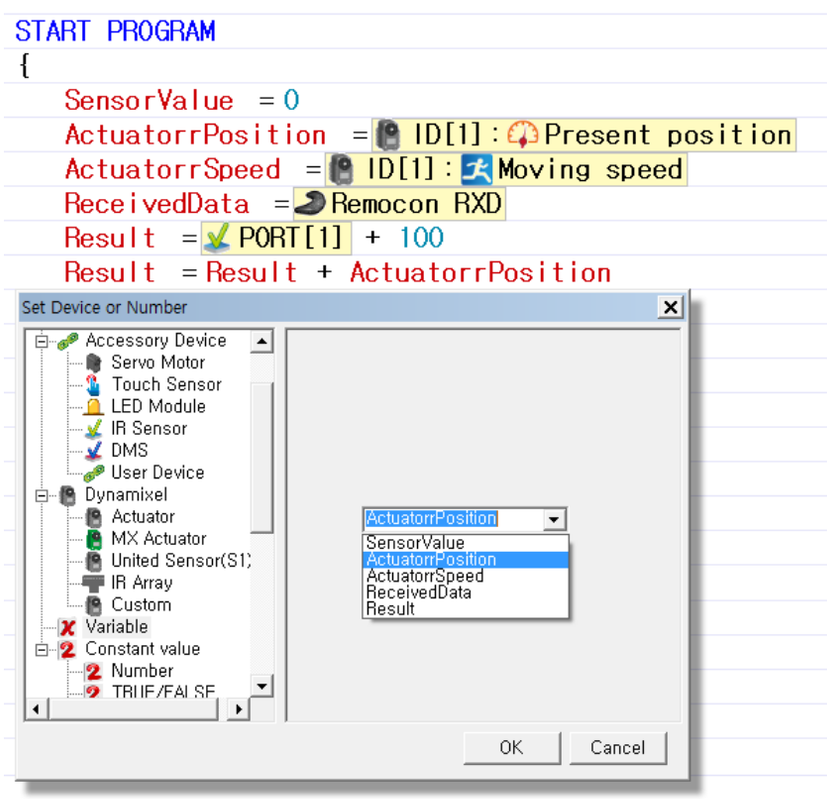
```
ID[4]:Joint offset = 1024
```

Variables

This is a storage place inside a program capable of saving, editing and reading data.

- If a variable with the same name already exists it is not created. Instead, the existing variable is used.
- Spaces are not permitted in variable names.
- Variable names cannot start with a number.
- Cannot have spaces and/or special characters (! @, #, $, etc).
- Variables are useful:
  - To remember a number
  - To change a value
  - To change multiple values simultaneously

Example 50 Variables may be used for many purposes. Existing variables are listed in the "Set Device or Number" window.



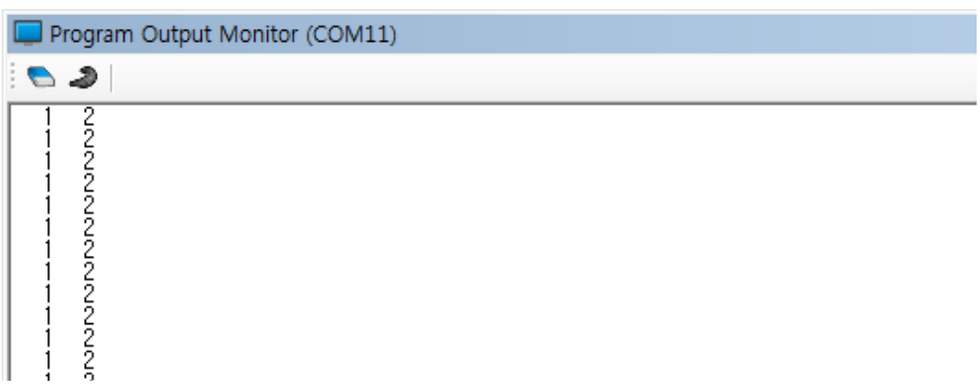
3 - 5 Application Phase

Screen Output
Buttons and LED
Attacking Duck
Smart Car

3 - 5 - 1 Screen Output

[Learning Objective]

Print 1 and 2 on the screen

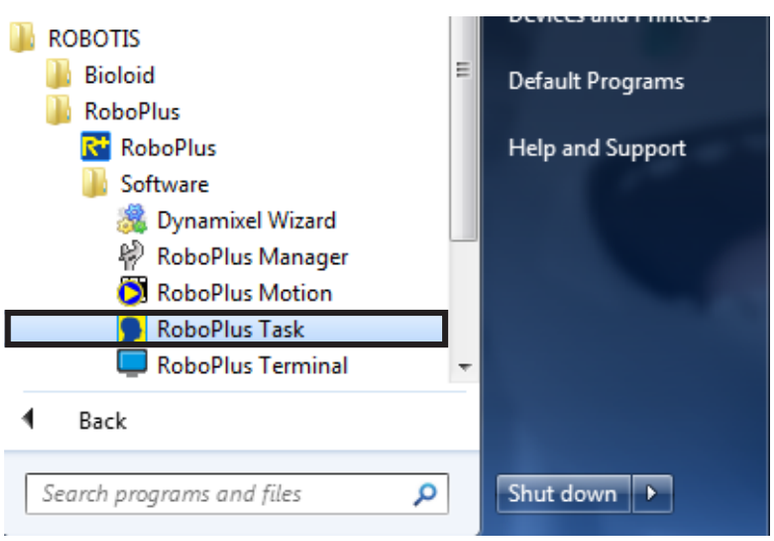


[STEP 1]

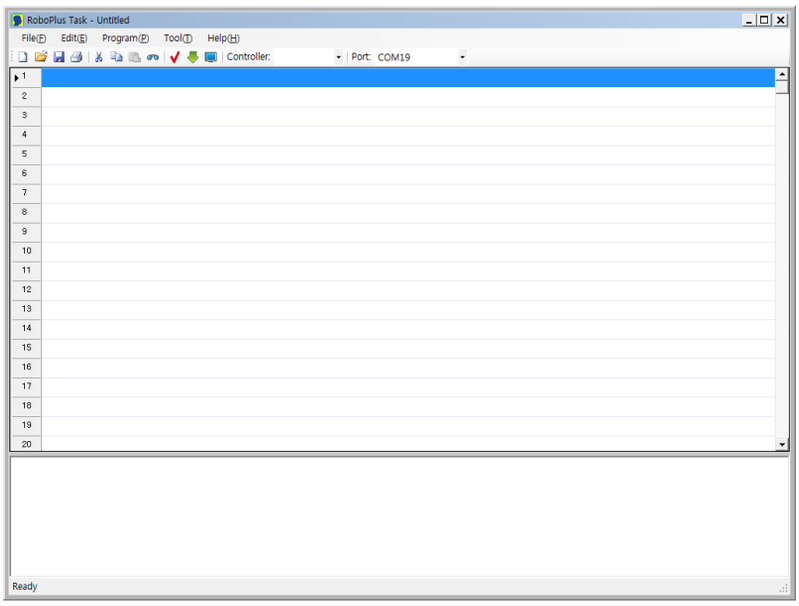
Write a task code

1. Run RoboPlus Task Program

Like in the image below, "Start>>All programs>>ROBOTIS>>RoboPlus>> Software>> RoboPlus Task" to run RoboPlus Task.

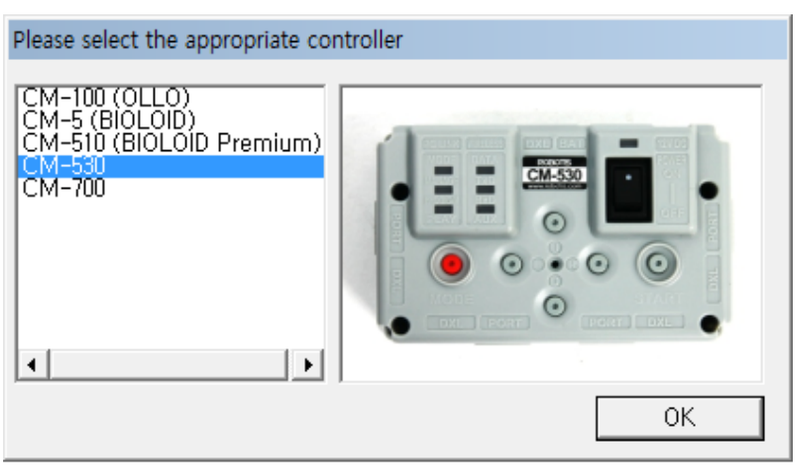


RoboPlus Task Initial screen



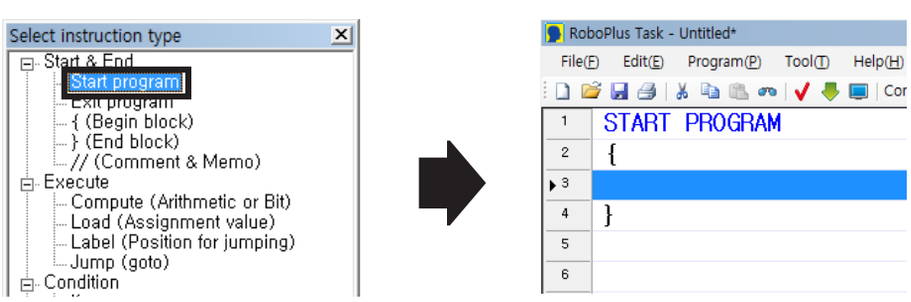
2. Select the controller

Press enter or double click on the blank line. From the ‘Select Controller’ communication window Select CM-530 and click OK.



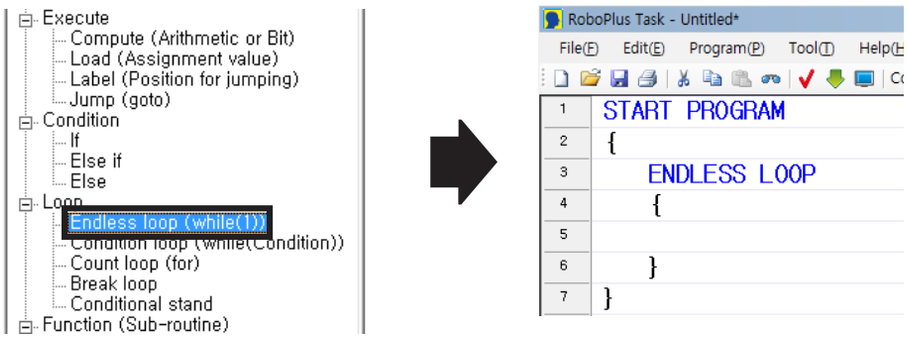
3. Generating 'Start Program'

From Select Instruction Type window, select "Start Program" and Start Program will be automatically generated in RoboPlus Task.



4. Input 'Endless Loop' Command

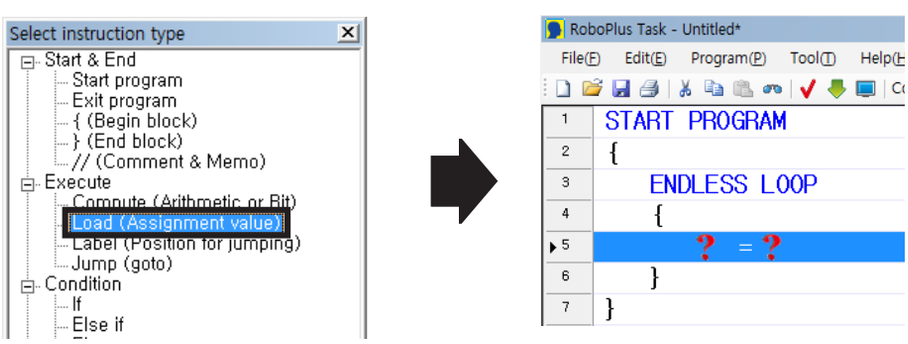
To print numbers repeatedly on the screen use the Endless Loop command (Create a command line). Double click or press ‘Enter’ on any blank lines in between {and} of 'Start Program.'



5. Input Load Command

Use the 'Load' command to input the 'print' command and to print numbers on the screen.

Double click the blank line in between {and} of 'endless loop'. Select Execute>>Load (assignment value) from the select instruction type window.

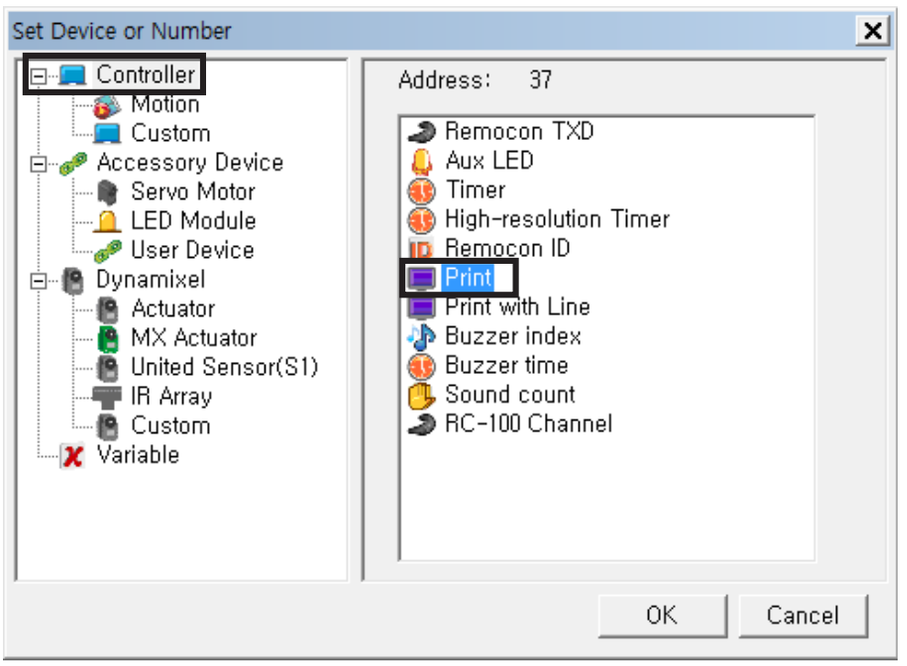


6. Load '1' into 'Print'

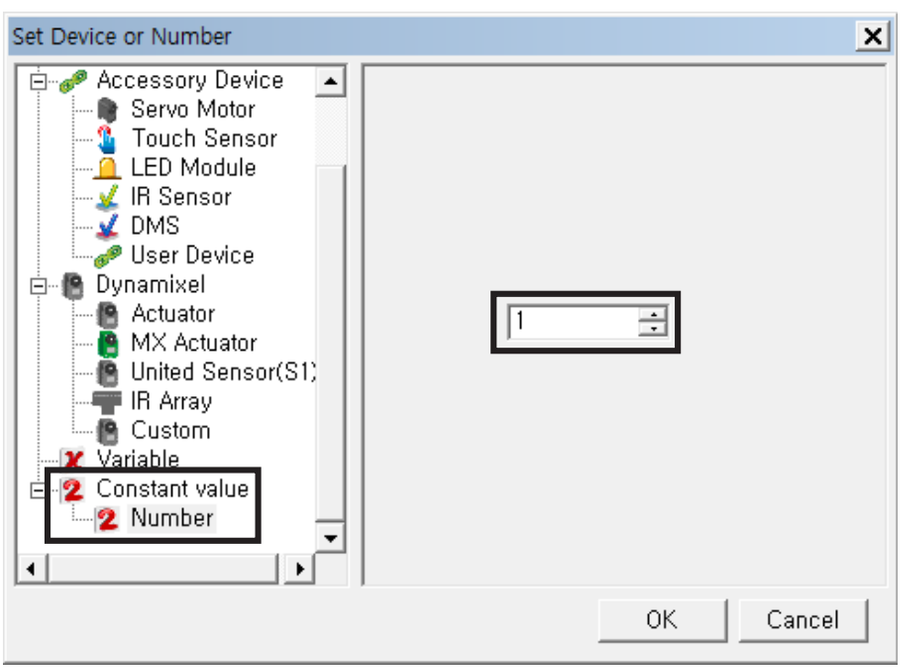
Select the left parameter from Load command.

Double click the left parameter or click and press enter to see the parameter selection window.

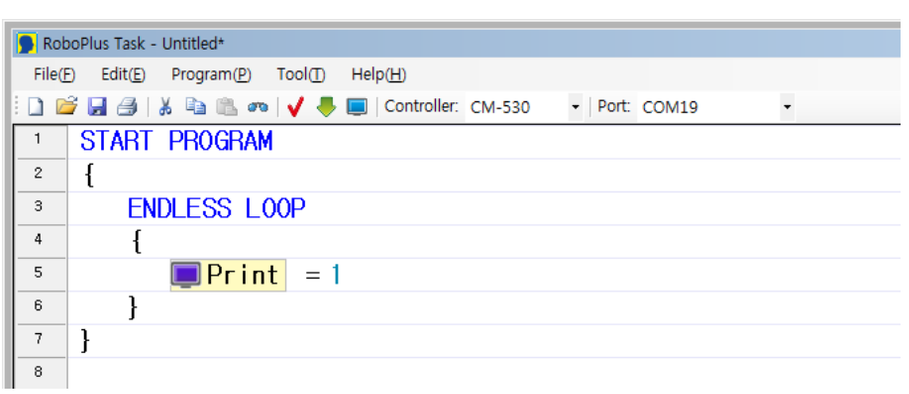
Select Controller>>Print and Click OK.



For the right parameter Select Constant value>>Number>>1

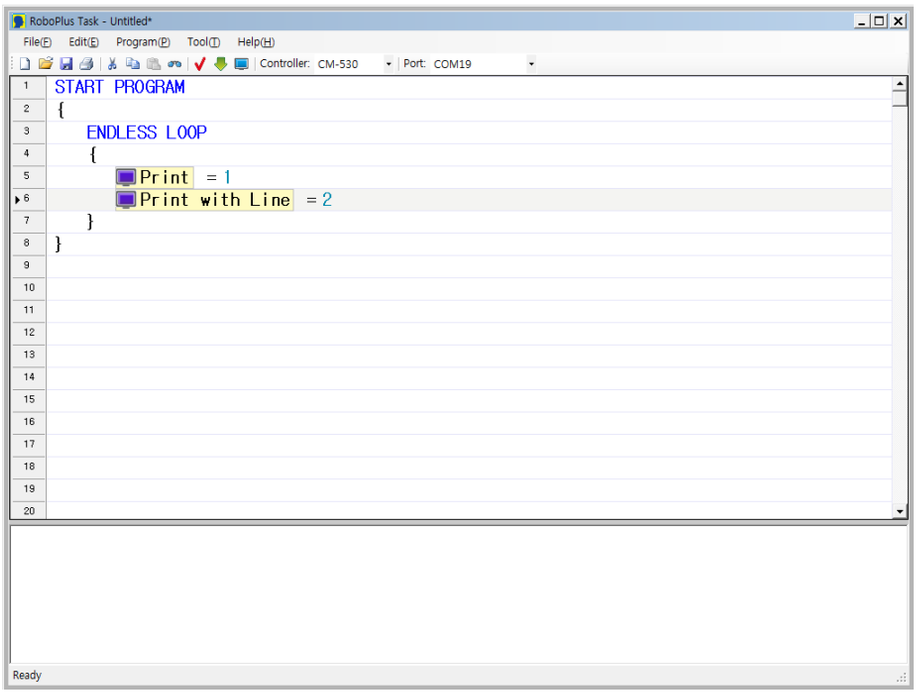


Once all parameters are set it will look like the following screen.



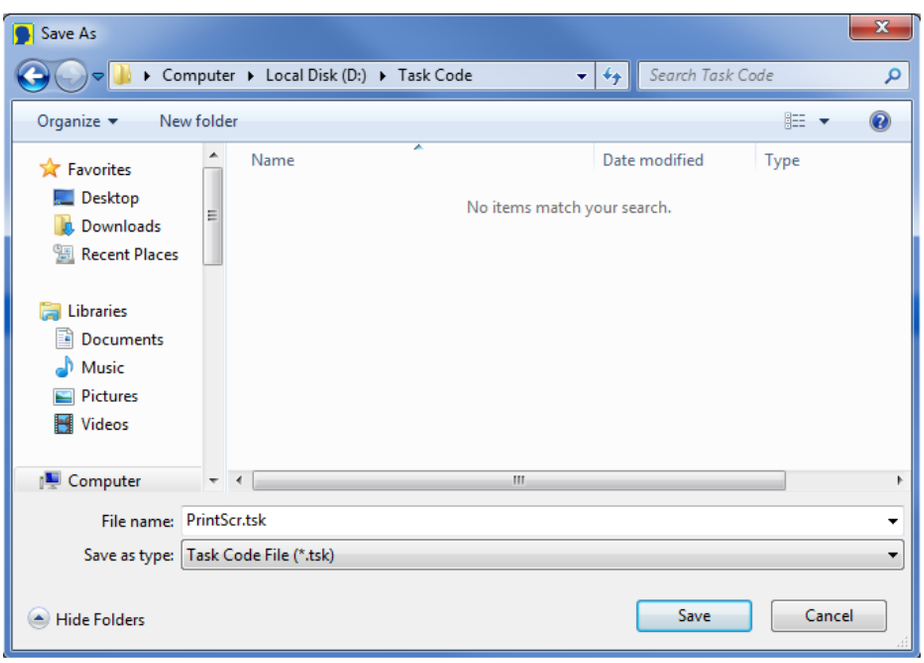
7. Load '2' into 'Print with line'

Select '}' under 'print' command and press the space bar to insert a blank line. Repeat steps 5~6 to add a 'Load' command. Select Controller>> Print with Line as the left parameter and set '2' as the right parameter. The final task code is as follows.



8. Save the task code

You can use the shortcut Ctrl + S or click on [ ] icon.



[STEP 2]

Download the task code.

Download the task code written in [STEP 1] to your controller.

[STEP 3]

Execute the task code

1. Open the Program Output Monitor

To view output during execution open <Program Output Monitor> window before executing the program. Please refer to 'Print Program Output'.

2. Execute the program

When you turn on the controller the mode LED will blink as it does in standby mode. Press the 'Mode' button to select play mode and press the start button to execute the downloaded task code. Check if it prints '1' and '2' on the Program Output Monitor.

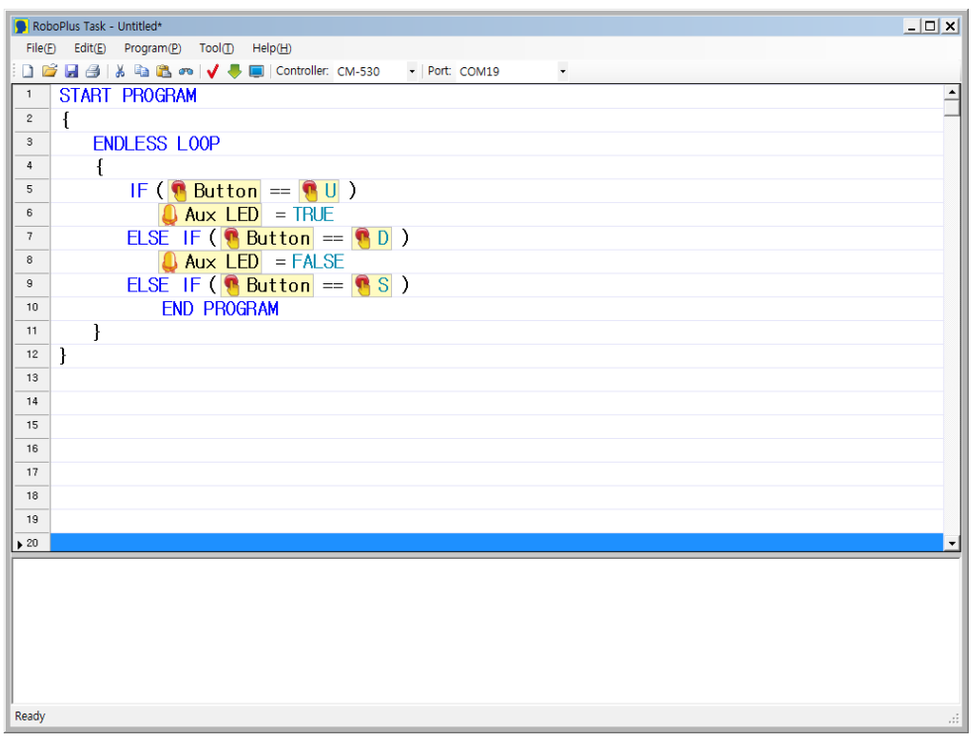
3-5-2 Buttons and LED

[Learning Objective]

Press 'U' to turn on the AUX LED and press 'D' to turn off the AUX LED.

[STEP 1]

Write task code.



[STEP 2]

Download the task code.

Download the task code written in [STEP 1] to the CM-530

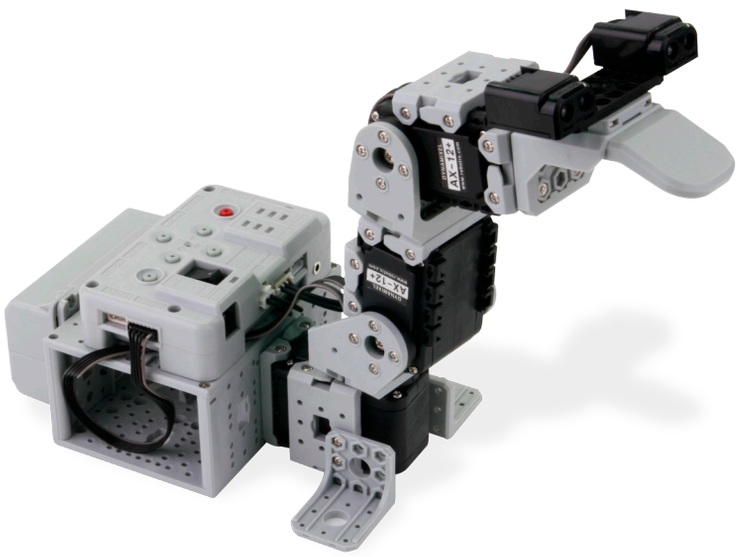
[STEP 3]

Execute the task code.

Execute the task code and see if the AUX LED is turning on/off when the corresponding button is pressed. Press 'Start' to end the program.

3-5-3 Attacking Duck

Let's build a robot duck that attacks approaching objects. Please refer to the assembly manual to complete the hardware.



Behavior Summary

Below are the behavior patterns of the attacking duck :

1. If no object is detected ID [1] moves sideways continuously until an object is detected; IDs [2] and [3] stay still prepared to attack.
2. If an object is detected by either the right or left sensor it rotates the beak towards the object by moving ID [1] joint.
3. If an object is detected by both sensors it stretches ID [2] & [3] to attack the object.
4. After the attack it resets to the initial position.

Task Code

```
23    // Assembly check
24    ID[1] : Moving speed = 50
25    CALL InitialPosition
26    ENDLESS LOOP
27    {
28
29        LOOP WHILE ( ID[1] : Present position < 752 )
30        {
31            ID[1] : Moving speed = 50
32            ID[1] : Goal position = 762
33            CALL Detect
34        }
35        LOOP WHILE ( ID[1] : Present position > 272 )
36        {
37            ID[1] : Moving speed = 50
38            ID[1] : Goal position = 252
39            CALL Detect
40        }
41    }
```

1. Input the default value of motor's moving speed and call on the initial behavior.
2. If the current motor position value of ID [1] is less than 752 it continuously moves at speed 50 to 762 and continues to search.
3. If the current motor position value of ID [1] is greater than 272 it continuously moves at speed 50 to 252 and continues to search.
4. Repeat 2, 3

```
133 FUNCTION Detect
134 {
135     IF ( PORT[2] >= 50 )
136     {
137         LOOP WHILE ( PORT[2] >= 50 )
138         {
139             ID[1] : Moving speed = 100
140             ID[1] : Goal position = ID[1] : Present position + 300
141             IF ( PORT[2] >= 50 && PORT[6] >= 50 )
142             {
143                 ID[1] : Goal position = ID[1] : Present position
144                 CALL BeakAttack
145             }
146         }
147     }
148     IF ( PORT[6] >= 50 )
149     {
150         LOOP WHILE ( PORT[6] >= 50 )
151         {
152             ID[1] : Moving speed = 100
153             ID[1] : Goal position = ID[1] : Present position - 300
154             IF ( PORT[2] >= 50 && PORT[6] >= 50 )
155             {
156                 ID[1] : Goal position = ID[1] : Present position
157                 CALL BeakAttack
158             }
159         }
160     }
161     ELSE
162     {
163
164     }
165 }
```

5. Program the initial motion.

   For the initial position input the initial alarm sound, appropriate target position values, and moving speeds for IDs [2] and [3] to bring the head up.

6. Program "Beak Attack"

   Input the appropriate number for the goal position value of ID [2] and ID [3] joints to create the attacking motion and initial preparation (head up). Program a standby function to pause the program process until the each motion is completed.

7. Program 'Detect'

   Create a function that follows the direction where an object is detected depending on the IR sensor value and attacks when both sensors detect an object.

```
168
169 FUNCTION InitialPosition
170 {
171     Buzzer time  = 255
172     Buzzer index  = 0
173     ID[2] : Moving speed  = 50
174     ID[3] : Moving speed  = 50
175     ID[2] : Goal position  = 512
176     ID[3] : Goal position  = 812
177 }
178
179 FUNCTION BeakAttack
180 {
181     // attack
182     ID[2] : Moving speed  = 300
183     ID[3] : Moving speed  = 300
184     ID[2] : Goal position  = 812
185     ID[3] : Goal position  = 512
186     CALL Movement
187     // turning
188     ID[2] : Moving speed  = 300
189     ID[3] : Moving speed  = 300
190     ID[2] : Goal position  = 512
191     ID[3] : Goal position  = 812
192     CALL Movement
193 }
194
195
```

```
197
198 FUNCTION Movement
199 {
200     // Actuators come to a sfull stop
201     WAIT WHILE ( ID[2] : Moving == TRUE || ID[3] : Moving == TRUE )
202 }
203
```

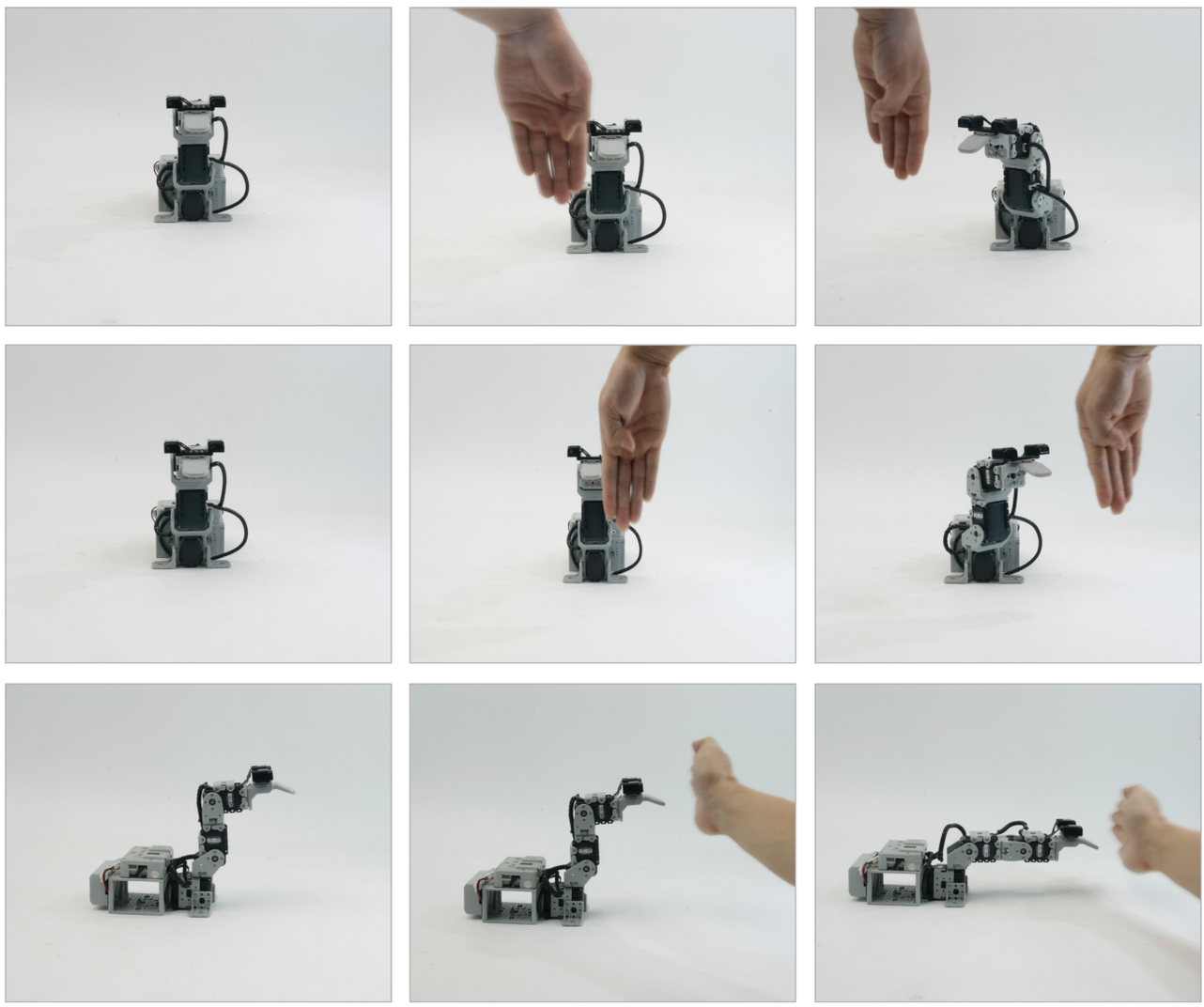
Double click to edit

Check Assembly

Basically, all task codes for examples include 'Motor Check (ID check),' ' Mode settings for motors (wheel mode/joint mode),' 'Check Assembly (assembly position, basic posture, sensors, etc)'

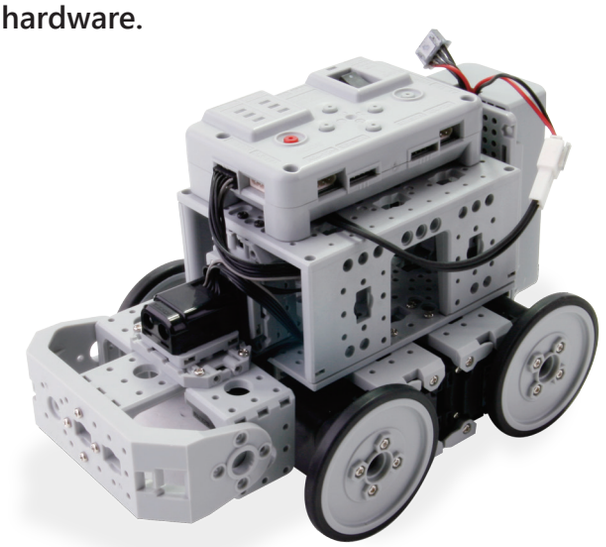
Check Motions

Play the downloaded task code. Place an object near the duck. Check if the robot follows the object it detects and attacks when detected by both sensors. Keep your face away from the robot during the operation.



3-5-4 Smart Car

Let's make a smart car that avoids obstacles on its own. Please refer to the assembly manual to complete the hardware.



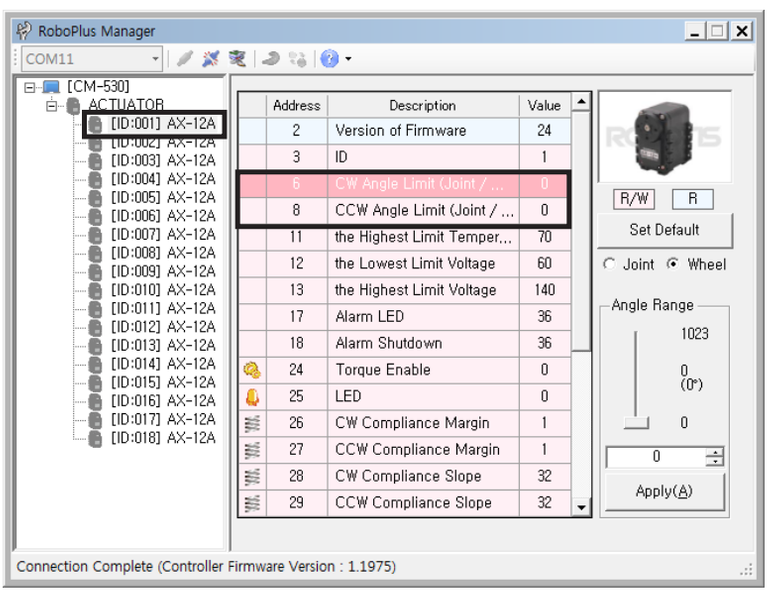
Behavior Summary

The behavior pattern of smart car is as follows.

1. Move forward when in operation and standby for mode selection.
2. It can be operated in either 'Control Mode' or 'Smart Mode.'
3. When in ‘Control Mode’ (U+S) it plays a different melody as it moves to the direction where the button is pressed.
4. When in ‘Smart Mode’ (D+S) It automatically detects objects and cliffs in front of it and avoids as it moves forward.

Mode Setting for AX-12A

To use the AX-12A as wheels, change to the wheel mode (endless rotation mode). You can change it using the RoboPlus Manager and set both CW and CCW position limit value to 0.



Mode Setting

Basically, all task codes for 26 examples of ROBOTIS PREMIUM are programmed to change to the appropriate motor mode in accordance with the examples (except the humanoid).

Task Code

```
// Assembly check
CALL ActuatorInitialization
// Actuators' initial speed
ForwardSpeed  = 500
ReverseSpeed  = 500
RotatingSpeed  = 500
// Initial Poisition : Forward
CALL Forward
Timer  = 1.408sec
WAIT WHILE ( Timer > 0.000sec )
CALL Stop
ENDLESS LOOP
{
    IF ( Button == U+S )
    {
        JUMP ControlMode
    }
    IF ( Button == D+S )
    {
        JUMP SmartMode
    }
}
```

```
50
51  ControlMode :
52      ENDLESS LOOP
53      {
54          IF ( Button == L )
55          {
56              CALL Advance
57          }
58          ELSE IF ( Button == R )
59          {
60              CALL Retreat
61          }
62          ELSE IF ( Button == D )
63          {
64              CALL ForwardLeft
65          }
66          ELSE IF ( Button == U )
67          {
68              CALL ForwardRight
69          }
70      }
71
```

```
74
75  SmartMode :
76      ENDLESS LOOP
77      {
78          // Avoids cliffs or voids
79          IF ( PORT[2] < 100 )
80              CALL AvoidCliff
81
82          // Avoids objects
83          IF ( PORT[1] >= 15 )
84              CALL AvoidObject
85          ELSE
86          {
87              CALL Forward
88          }
89      }
90
```

1. Call 'ActuatorInitialization' function to initialize the actuator; enter the default value of moving speed and move forward as an initial behavior.

2. Allow to select two different modes depending on the pressed button on the controller. (U+S : Control Mode, D+S : Smart Mode)

3. Control Mode : Robot moves depending on the pressed button on the controller. (L : Forward, R : Backward, U : Turn left, D : Turn right)

4. Smart Mode : When robot detects an object or a cliff in front of it as it moves forward it detours to avoid.

```
234 FUNCTION TimerStandby
235 {
236     WAIT WHILE ( Timer > 0.000sec )
237 }
238
239 FUNCTION Stop
240 {
241     ID[1] : Moving speed = 0
242     ID[2] : Moving speed = 0
243     ID[3] : Moving speed = 0
244     ID[4] : Moving speed = 0
245 }
246
247 FUNCTION Forward
248 {
249     ID[1] : Moving speed = CW:0 + ForwardSpeed
250     ID[2] : Moving speed = CCW:0 + ForwardSpeed
251     ID[3] : Moving speed = CW:0 + ForwardSpeed
252     ID[4] : Moving speed = CCW:0 + ForwardSpeed
253 }
254
255 FUNCTION Reverse
256 {
257     ID[1] : Moving speed = CCW:0 + ReverseSpeed
258     ID[2] : Moving speed = CW:0 + ReverseSpeed
259     ID[3] : Moving speed = CCW:0 + ReverseSpeed
260     ID[4] : Moving speed = CW:0 + ReverseSpeed
261 }
```

```
265 FUNCTION TurnRight
266 {
267     ID[1] : Moving speed = 0
268     ID[2] : Moving speed = CCW:0 + RotatingSpeed
269     ID[3] : Moving speed = 0
270     ID[4] : Moving speed = CCW:0 + RotatingSpeed
271 }
272
273 FUNCTION TurnLeft
274 {
275     ID[1] : Moving speed = CW:0 + RotatingSpeed
276     ID[2] : Moving speed = 0
277     ID[3] : Moving speed = CW:0 + RotatingSpeed
278     ID[4] : Moving speed = 0
279 }
280 FUNCTION FallbackRight
281 {
282     ID[1] : Moving speed = CCW:0 + RotatingSpeed
283     ID[2] : Moving speed = CCW:0 + RotatingSpeed
284     ID[3] : Moving speed = CCW:0 + RotatingSpeed
285     ID[4] : Moving speed = CCW:0 + RotatingSpeed
286 }
287
288 FUNCTION ActuatorInitialization
289 {
290     ID[1] : ADDR[8(w)] = 0000 0000 0000 0000
291     ID[2] : ADDR[8(w)] = 0000 0000 0000 0000
292     ID[3] : ADDR[8(w)] = 0000 0000 0000 0000
293     ID[4] : ADDR[8(w)] = 0000 0000 0000 0000
294 }
```

5. Create 'Stop, Forward, Backward, Turn Left, Turn Right, Fallback Right' functions. For the moving speed of each motors add CW:0 and CCW:0 to the initial value of forward speed, backward speed, turning speed variables.

6. Create 'WaitTimerCompletion' and 'ActuatorInitialization' functions.

```
204 FUNCTION AvoidCliff
205 {
206     CALL Reverse
207
208     Timer = 12
209     CALL TimerStandby
210
211     CALL FallbackRight
212
213     Timer = 16
214     CALL TimerStandby
215
216     CALL Forward
217 }
218
219 FUNCTION AvoidObject
220 {
221     CALL Reverse
222
223     Timer = 6
224     CALL TimerStandby
225
226     CALL TurnRight
227
228     Timer = 12
229     CALL TimerStandby
230
231     CALL Forward
232 }
233
```

7. Program to execute 'Forward, Backward, Turn Left, Turn Right' functions which are used in the control mode with different melodies for 3 seconds.

8. Create 'Avoid_Cliff' and 'Avoid_Object' functions for the smart mode.

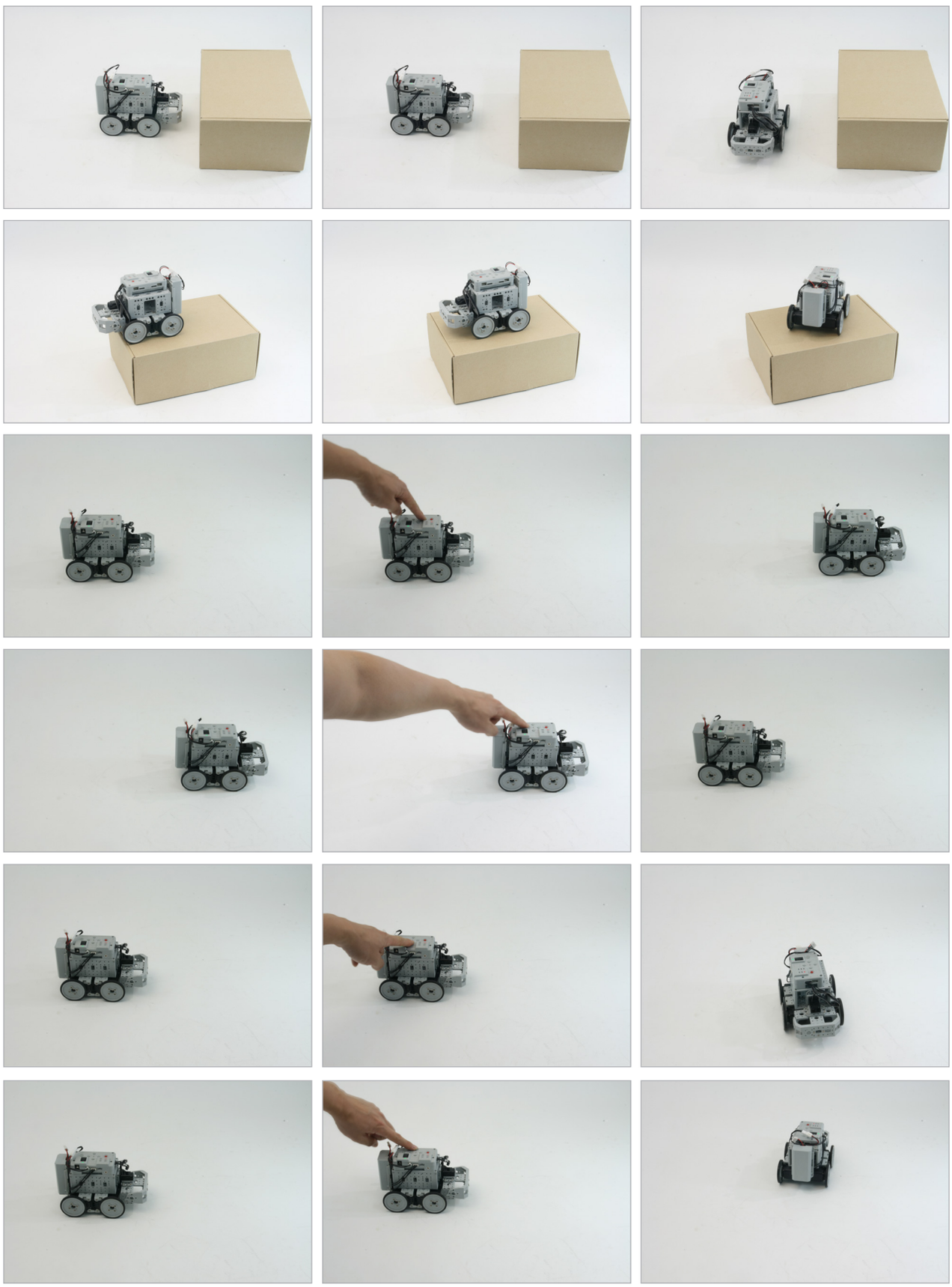
9. For 'Avoid_Cliff' function, call both 'Backward' and 'Fallback Right' functions for long. For 'Avoid_Object' function, call both 'Backward and Fall Backright' functions for short.

```
304 FUNCTION Advance
305 {
306     Buzzer time = Play Melody
307     Buzzer index = Melody3
308     CALL Forward
309     Timer = 3.072sec
310     WAIT WHILE ( Timer > 0.000sec )
311     CALL Stop
312 }
313 FUNCTION Retreat
314 {
315     Buzzer time = Play Melody
316     Buzzer index = Melody4
317     CALL Reverse
318     Double click to edit 072sec
319     WAIT WHILE ( Timer > 0.000sec )
320     CALL Stop
321 }
```

```
326 FUNCTION ForwardLeft
327 {
328     Buzzer time = Play Melody
329     Buzzer index = Melody1
330     CALL TurnLeft
331     Timer = 3.072sec
332     WAIT WHILE ( Timer > 0.000sec )
333     CALL Stop
334 }
335 FUNCTION ForwardRight
336 {
337     Buzzer time = Play Melody
338     Buzzer index = Melody1
339     CALL TurnRight
340     Timer = 3.072sec
341     WAIT WHILE ( Timer > 0.000sec )
342     CALL Stop
343 }
```

Check Motion

Play the downloaded task code. Check if it avoids objects and cliffs in the smart mode. In the control mode, check if it follows the direction of pressed buttons as it plays the melody.



3-6 Other information

3-6-1 Rule Check

Rule check

Rule Check is a process to verify that the task code has been properly written according to the rules. RoboPlus Task has a function that checks for grammatical errors and prints the causes and locations of the errors. Below is an example of a grammatical error and its location.

```
=== Line Check(Untitled) ===
Line:17 Please select a device or number.
=== Total Error: 1 ===
```

Double-click on the error message to move to the location of the error.

Below is a list of grammatical errors.

- Select a number or device.
- Block beginning doesn't exist.
- A block must be designated.
- Only 1 "Start Program" is permitted
- "Start Program" can not be defined in a block.
- A function can not be defined in a block.
- A command to execute the designated block does not exist.
- This line should be included in a block.
- The start and end of the block are not paired.
- "Start Program" does not exist.
- A function with the same name already exists.
- "Restoration" can only be used in a function.
- A label with same name already exists.
- A function can not call on itself.
- "If/If else" has been used improperly.
- There is no loop to end.
- A block must consist of at least one command.
- A jump to another block is not permitted.
- There is no function to call on.
- Only 1 callback function may exist.

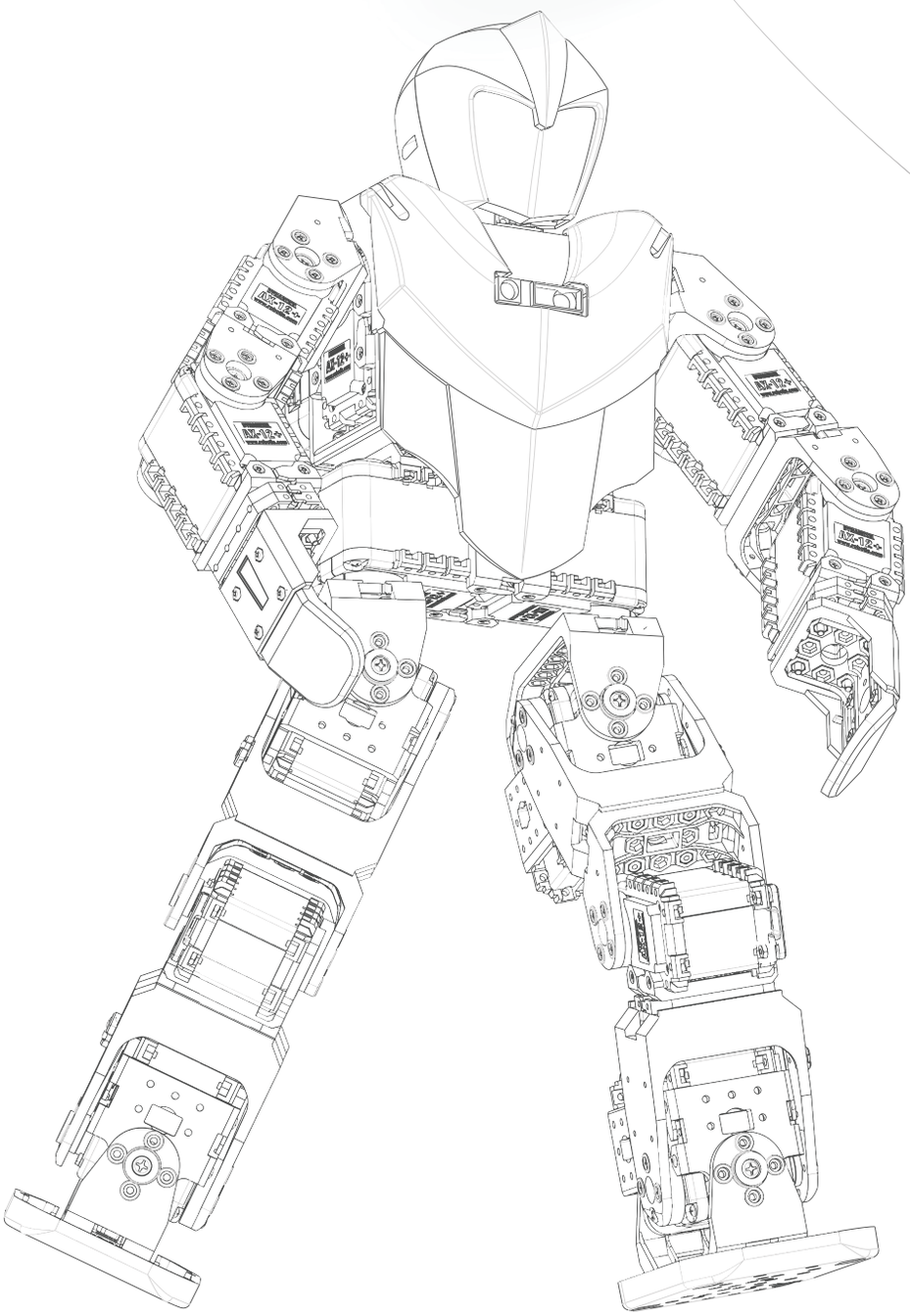
- A callback function can not be defined in a block.
- This command can not be used in a callback function.
- This device can not be used in a callback function.
- This command can not be used with the selected controller.
- There is a device which can not be used with the selected controller.

3-6-2 Error Messages

Error Messages

The following table provides a list of errors you may see while writing codes and printing the output on the screen with RoboPlus Task.

Error Code	Description	Output Example
8100	Attempting to communicate with an unconnected Dynamixel ID. If properly connected, check the cables.	{[ERROR:8100:0005:03]} 8100 : Error Code 0005 : ID of Dynamixel which was attempted for communication 03 : Internal Information
0009	Too many consecutive function called on. (More than 6 times)	{[ERROR:0009:0033:06]} 0009 : Error Code 0033 : Internal Information 06 : Internal Information
8001	Attempting to read address of the controller designated as "write only"	{[ERROR:8001:000C:02]} 8001 : Error Code 000C : Internal Information 02 : Internal Information
8002	Attempting to write address of the controller designated as "read only".	{[ERROR:8002:0009:02]} 8002 : Error Code 0009 : Internal Information 02 : Internal Information
NONE	Motion page to execute does not exist.	[Invalid Page Read:00FE] 00FE : Motion page number (hexadecimal)

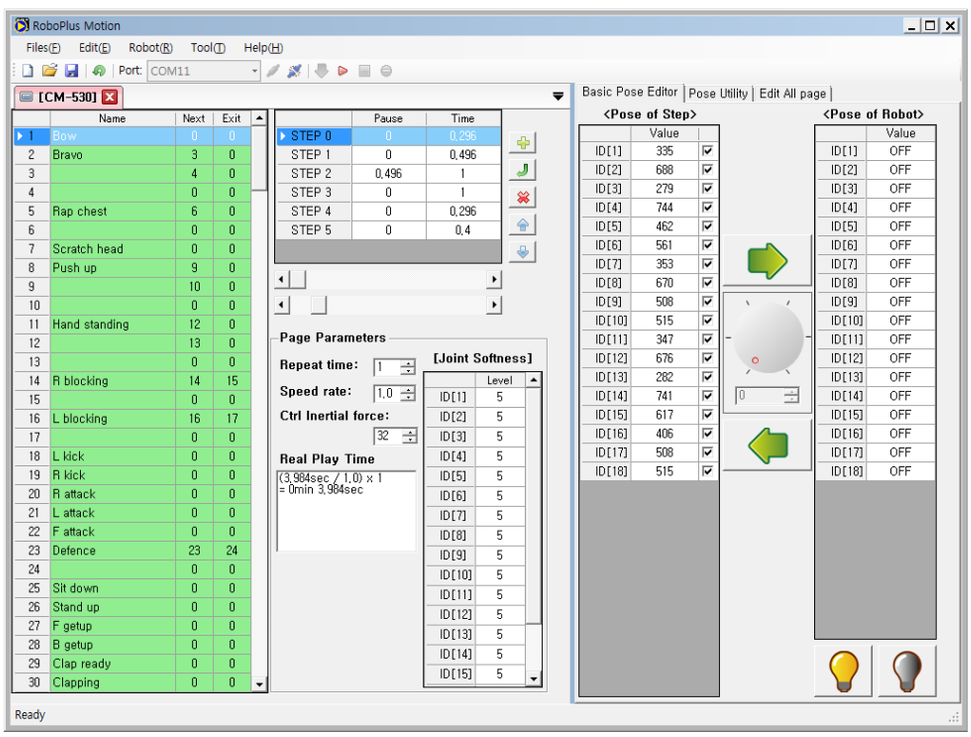


4. RoboPlus Motion

4 - 1 What is RoboPlus Motion?

Motion

A motion is a set of AX-12A's position and speed data necessary for robot movements. In order for the robot to move a motion file is required.
A suitable motion file must be downloaded for the assembled robot in order to move. A motion file is identified by the icon below and its file extension is .mtn.



Motion and Task Code

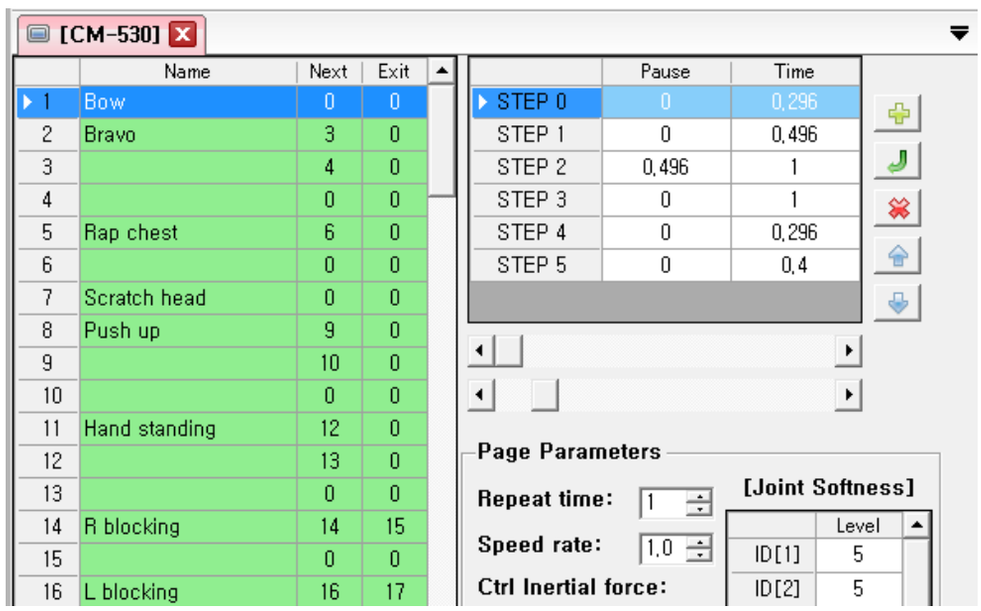
A task code file is a program, and a motion file is data. For better understanding, let us think about MP3 players and MP3 files. If there were no MP3 players, you will not be able to listen to music because MP3 files could not be played. The result is the same when there is an MP3 player but no MP3 file. If you want to make your robot move, you need a task code file. If the task code downloaded into your robot uses motions, you must download the motion file as well. If no motions are used in the task code, you do not need the motion file.

4-2 Getting Started

4-2-1 Robot Motion and File Motion

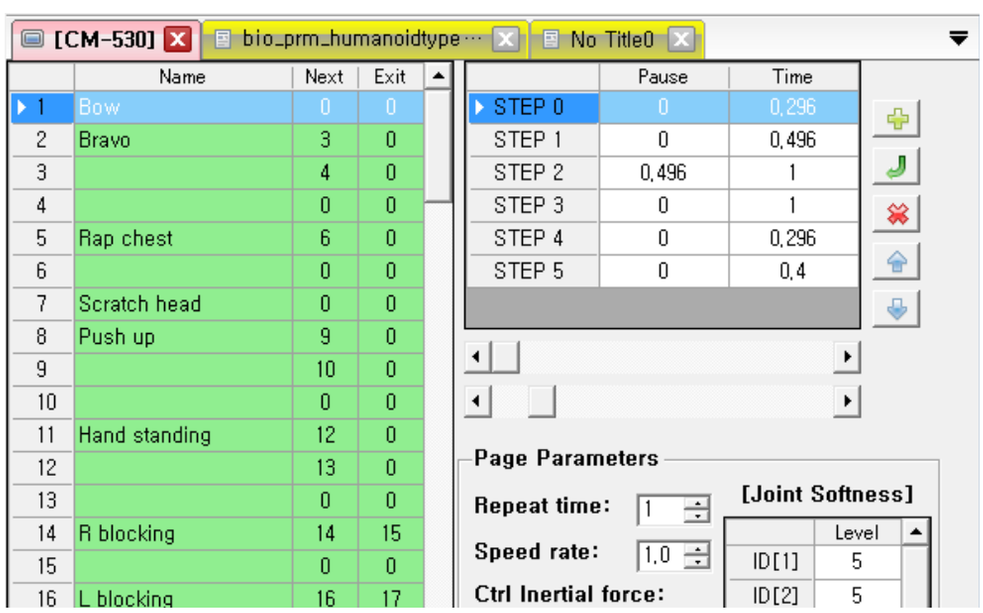
Robot Motion

Robot Motion refers to the motion data in the controller. These data can be seen and edited on the "Robot Motion" window. This window is displayed only when the robot is connected.



File Motion

"File Motion" refers to the motion data in the form of files in the PC. These data can be seen and edited on the "File Motion" window. Multiple "File Motion" windows can be displayed at once.



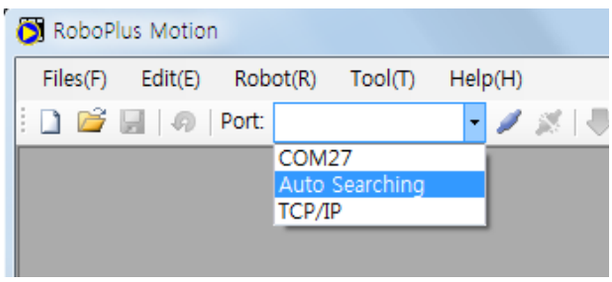
4-2-2 Connect Robot

[STEP 1]

Connect your robot to the PC. For the CM-530 to communicate with the PC you need to use a USB cable (from mini USB connector of the CM-530 to PC's USB slot).

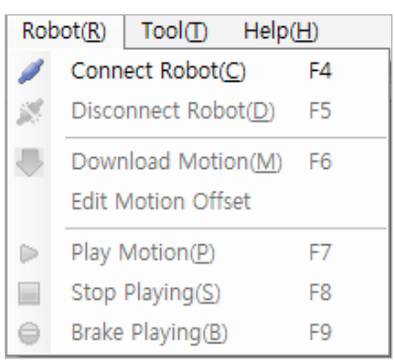
[STEP 2]

Select the communication port to use which your robot is connected to. If you don't know the port number, use the "Auto Searching" function.



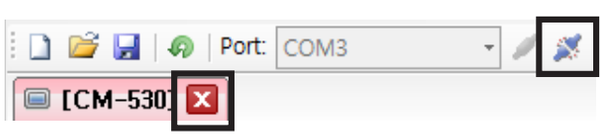
[STEP 3]

Choose the "Connect Robot" menu.



[STEP 4]

Disconnect the robot. To disconnect from the robot choose the "exit" menu or simply close Robot Motion window.

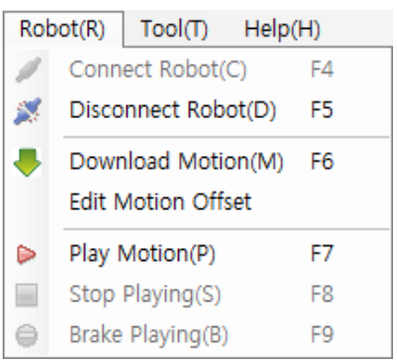


4-2-3 Motion Download

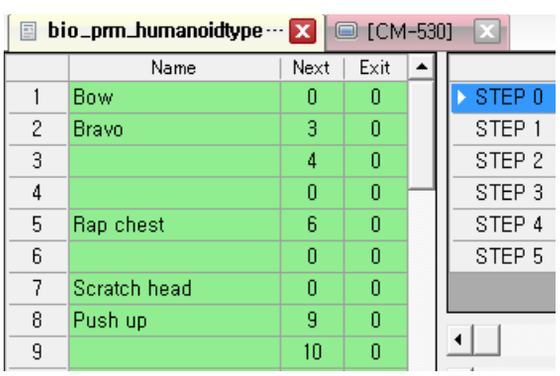
[STEP 1] Open the file motion to download.

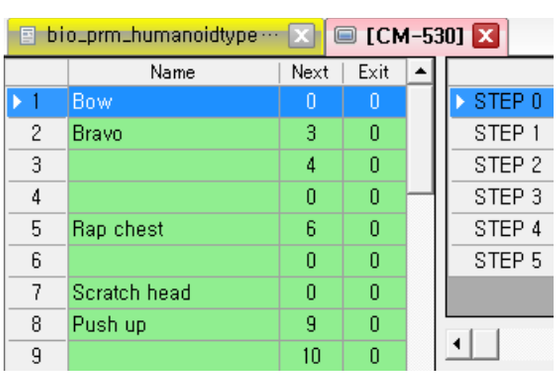
[STEP 2] Connect to the robot

[STEP 3] Click "Download Motion" menu and wait for the download to complete.



[STEP 4] Verify that the contents of the file motion have been copied to the robot motion as seen below.

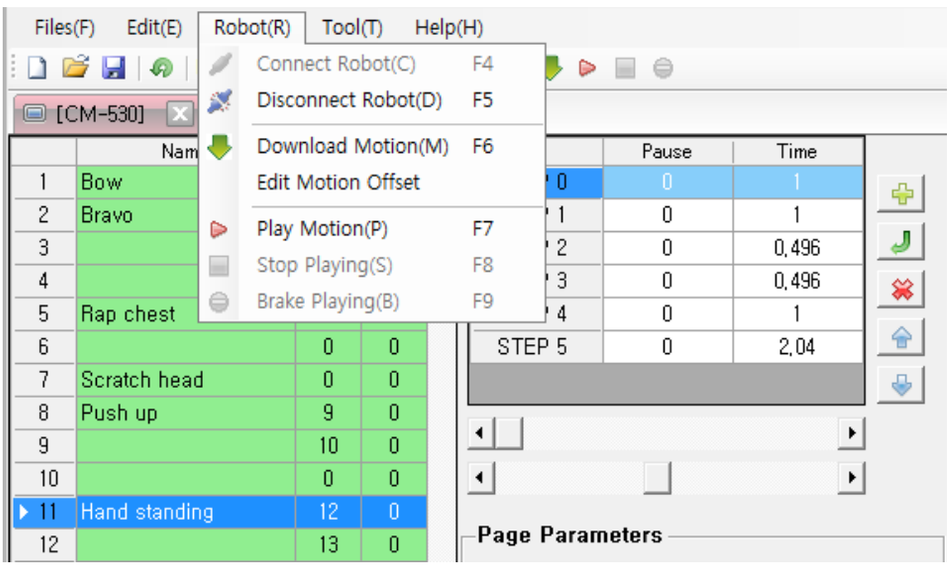




4-2-4 Play/Stop Motion

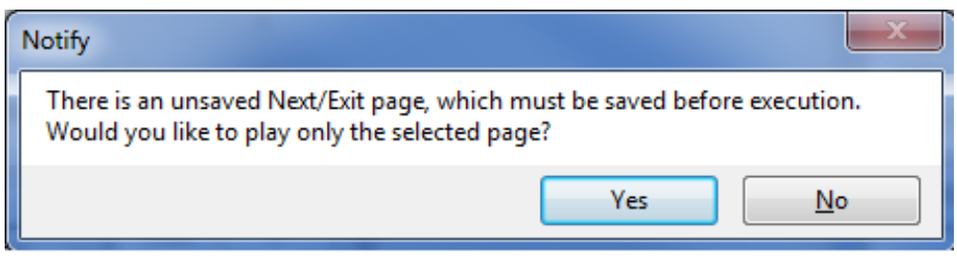
Play Motion

You can play the created motions. Search the page to play and click 'Play Motion.'

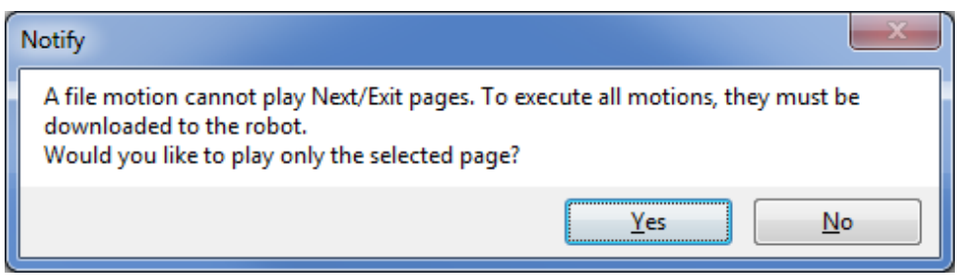


Errors may occur when trying to play motions

This error occurs when working on a "Robot Motion" window. In this case, the page linked as 'Next' or 'Exit' has been modified; but, the controller does not have enough memory to temporarily save it. This can be solved by saving the page before execution. If you proceed without saving it only the current page will be played.



This error occurs when working on a "File Motion" window. In this case, the data in the PC is not the controller and the controller does not have enough memory to temporarily save the ‘Next’ or ‘Exit’ page. You can execute only the selected page. To play linked pages you must download the motion to the robot.



Motion Stop

Stops the motion that is being carried out. "Stop Motion" does not stop execution right away. Instead, the "Exit" page is executed before stopping.

Emergency Stop

Stops the motion that is being carried out. Unlike "Stop Motion," "Emergency Stop" halts execution immediately.

4-3 Motion Editor

ID Setting for AX-12A

The motion player in the CM-530 can control a total of 26 AX-12As (from ID 0 to 25). Therefore, to create a motion with the RoboPlus Motion the ID of each AX-12A must be between 0 and 25.

Control Priority

Below is the control priority for the CM-530 :

1. RoboPlus Motion (ID of Dynamixel is between 0 and 25.)
2. RoboPlus Task

Setting method for Motion Editor and Task Code.

AX-12A Auto Shutdown Function

The AX-12A has an Auto Shutdown function. When the Auto Shutdown function is triggered the following will appear :

- The Dynamixel's LED will blink.
- The motor will stop moving, resulting in no torque.

This function prevents the AX-12A from getting damaged. When creating a motion the joint may not move. This is because the Auto Shutdown function has been triggered by the causes listed below.

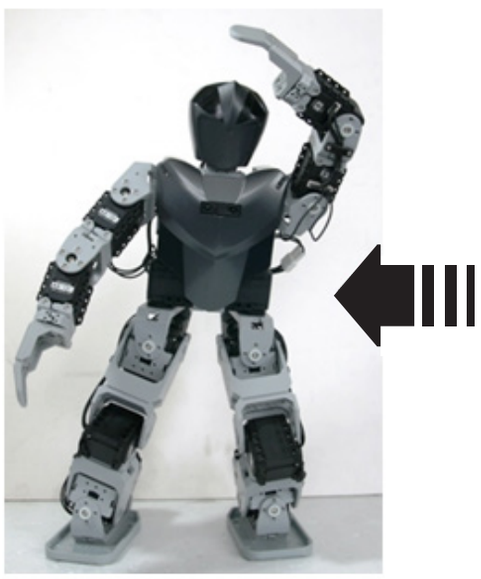
- The motor has overheated due to an increase in internal temperature.
- The motor has been under too much load for an extensive period of time.

To solve this problem the following steps must be taken.

- Resolve what triggered the Auto Shutdown function.
  - If the motor has overheated allow it time to cool down.
  - If the motor is under too much load remove some of the load.
- Turn off the AX-12A and turn it back on.

4-3-1 Pose Editing

A pose is the robot's position at a point in time. It is a collection of motor position values required for the posture.



<Pose of Step>

	Value
ID[1]	200
ID[2]	815
ID[3]	248
ID[4]	741
ID[5]	496
ID[6]	575
ID[7]	502
ID[8]	507
ID[9]	501
ID[10]	507
ID[11]	281
ID[12]	739
ID[13]	30
ID[14]	992
ID[15]	767
ID[16]	256
ID[17]	502
ID[18]	509

200

<Pose of Robot>

	Value
ID[1]	OFF
ID[2]	OFF
ID[3]	OFF
ID[4]	OFF
ID[5]	OFF
ID[6]	OFF
ID[7]	OFF
ID[8]	OFF
ID[9]	OFF
ID[10]	OFF
ID[11]	OFF
ID[12]	OFF
ID[13]	OFF
ID[14]	OFF
ID[15]	OFF
ID[16]	OFF
ID[17]	OFF
ID[18]	OFF

Pose of Step

"Pose of Step" refers to the data values of the pose.

<Pose of Step>

	Value
ID[1]	233
ID[2]	786
ID[3]	299
ID[4]	725
ID[5]	462

Pose of Robot

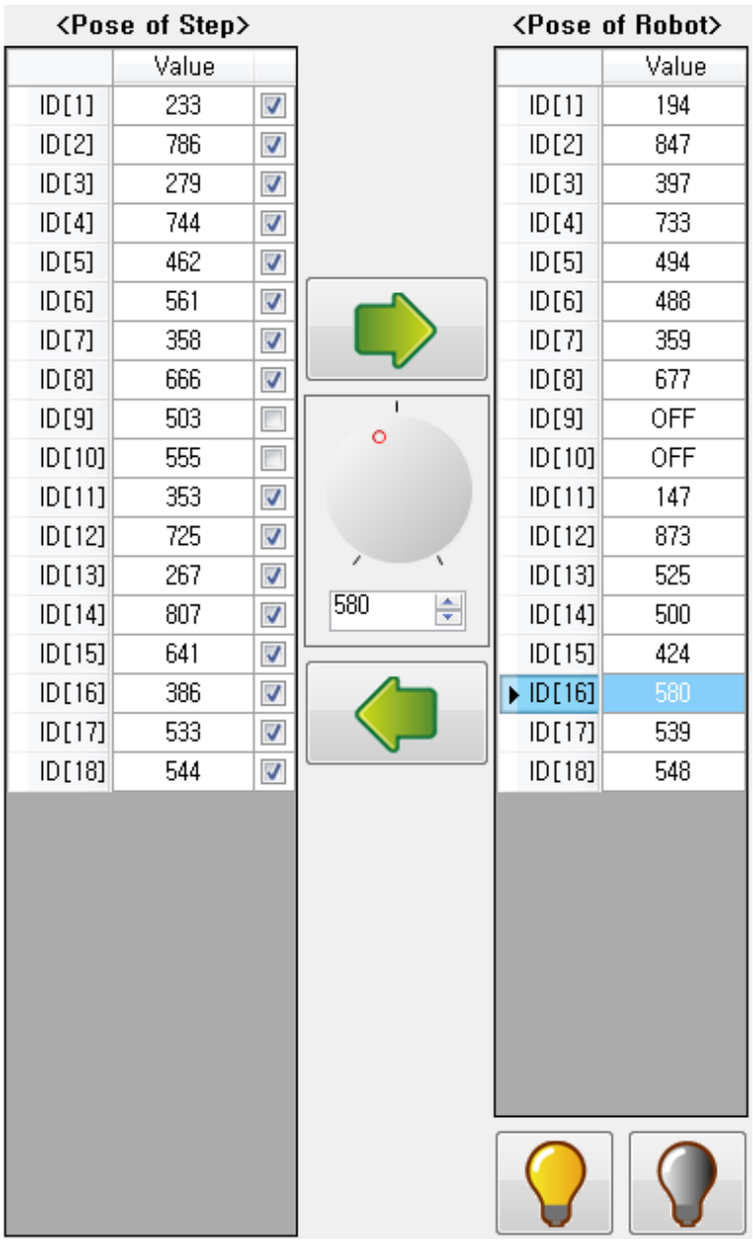
"Pose of Robot" refers to the position values of the connected robot's joints. When the "Pose of Robot" is modified the robot will move accordingly.

<Pose of Robot>

	Value
ID[1]	194
ID[2]	847
ID[3]	397
ID[4]	733
ID[5]	494

Basic Pose Editing

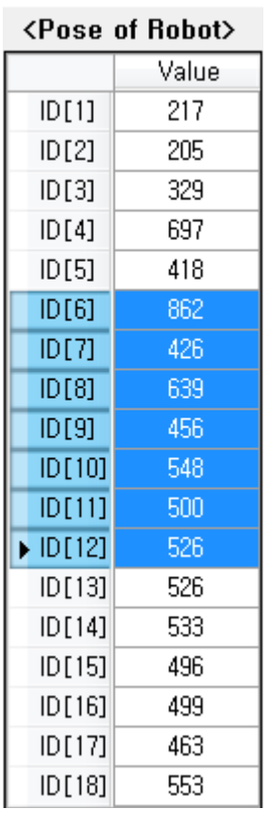
The Basic Pose Editor is a simple editor that may be used for any type of robot. To change ID number used in "Pose of Step," use the ID Editing Function.



Select Actuator

Click on a row to select an actuator. The following methods may be used to select multiple actuators.

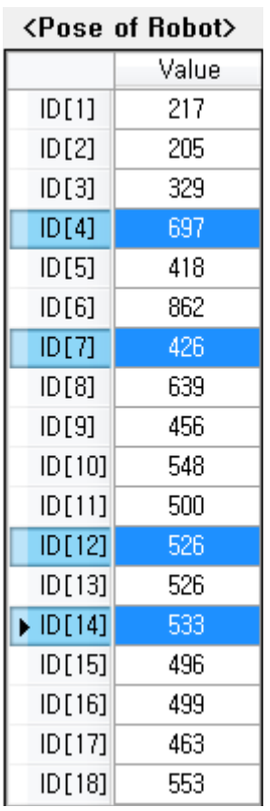
To select actuators in consecutive order



<Pose of Robot>	
	Value
ID[1]	217
ID[2]	205
ID[3]	329
ID[4]	697
ID[5]	418
ID[6]	862
ID[7]	426
ID[8]	639
ID[9]	456
ID[10]	548
ID[11]	500
ID[12]	526
ID[13]	526
ID[14]	533
ID[15]	496
ID[16]	499
ID[17]	463
ID[18]	553

- Drag with the mouse

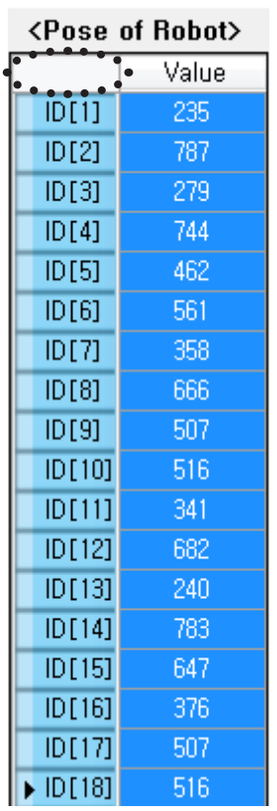
To select actuators separately



<Pose of Robot>	
	Value
ID[1]	217
ID[2]	205
ID[3]	329
ID[4]	697
ID[5]	418
ID[6]	862
ID[7]	426
ID[8]	639
ID[9]	456
ID[10]	548
ID[11]	500
ID[12]	526
ID[13]	526
ID[14]	533
ID[15]	496
ID[16]	499
ID[17]	463
ID[18]	553

- Press Ctrl and select ID#

To select all actuators



<Pose of Robot>	
	Value
ID[1]	235
ID[2]	787
ID[3]	279
ID[4]	744
ID[5]	462
ID[6]	561
ID[7]	358
ID[8]	666
ID[9]	507
ID[10]	516
ID[11]	341
ID[12]	682
ID[13]	240
ID[14]	783
ID[15]	647
ID[16]	376
ID[17]	507
ID[18]	516

- Press button to select

Torque on/off

The "Torque On/Off" function enables you to make a pose manually by turning the robot's joints on or off. This function is only available in "Pose of Robot." If the torque is on its position value will be shown. Otherwise, the value will be displayed as 'OFF.'

<Pose of Robot>

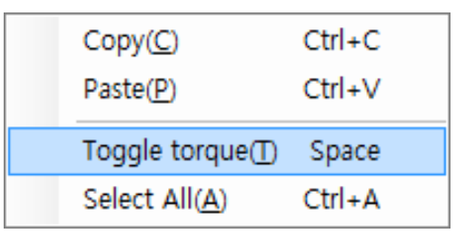
	Value
ID[1]	216
ID[2]	204
ID[3]	329
ID[4]	OFF
ID[5]	418
ID[6]	862
ID[7]	426
ID[8]	OFF
ID[9]	456
ID[10]	548
ID[11]	499
ID[12]	OFF
ID[13]	526
ID[14]	OFF
ID[15]	OFF
ID[16]	499
ID[17]	463
ID[18]	553

The torque may be turned on/off using the following methods.

Press the "On" button to turn on the selected actuator.

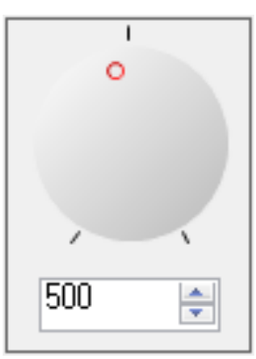
Press the "Off" button to turn off the selected actuator.

Click the "Torque Toggle" menu to turn it on/off.



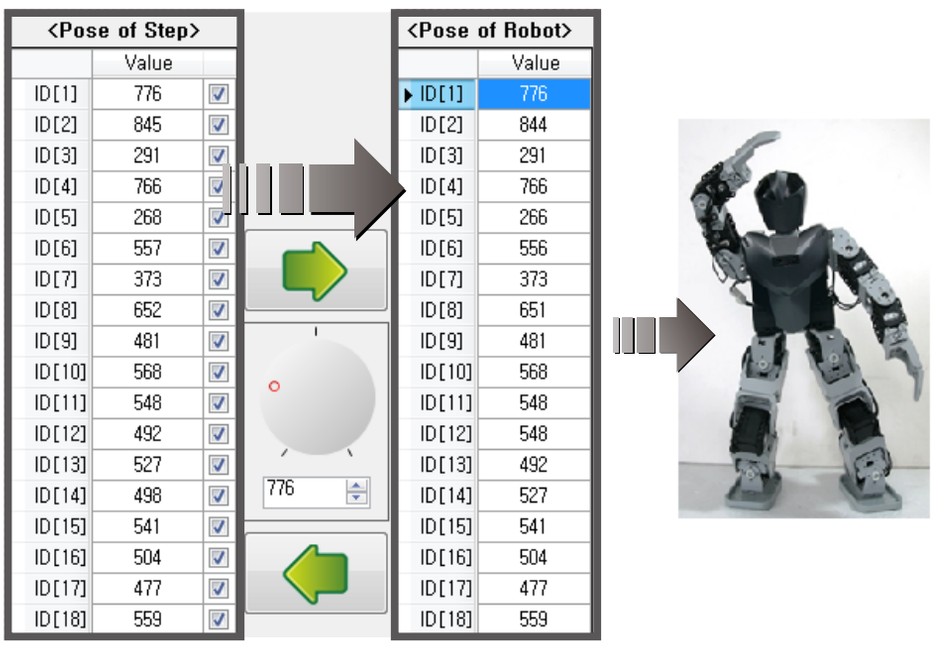
Change Actuator Value

The position value of the selected actuator may be changed after choosing the joint in "Pose."



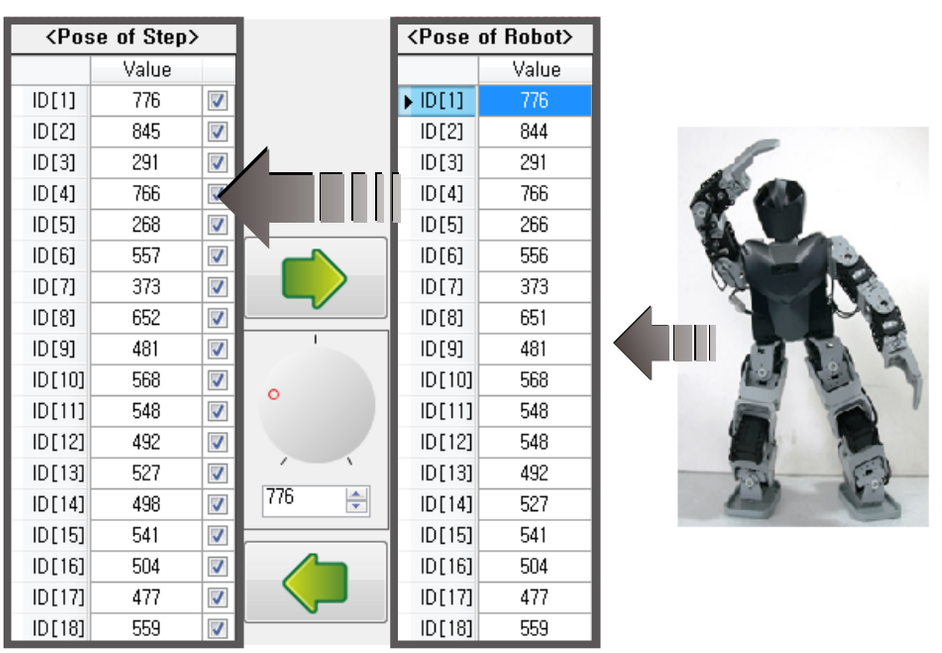
Play Pose

To execute a pose move the "Pose of Step" to "Pose of Robot."



Capture Pose

To "Capture"(store) a pose move "Pose of Robot" to "Pose of Step"

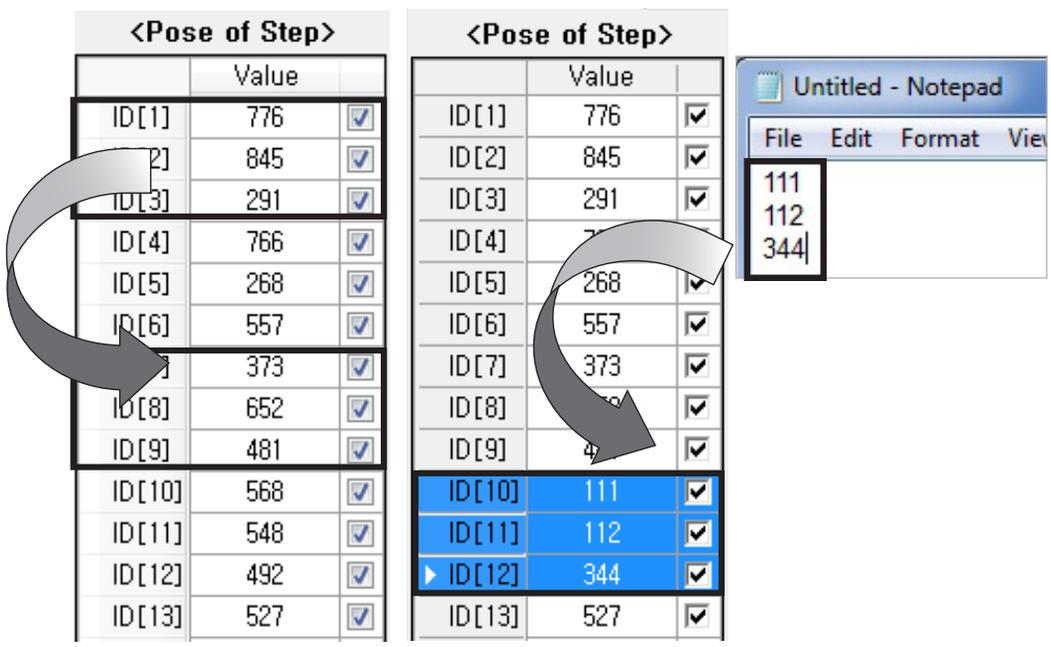


Copy/Paste Pose

These functions enable to change the actuator values easily. Not only can poses be copied and pasted within a program, but texts from other files, such as Microsoft Excel, may be copied.

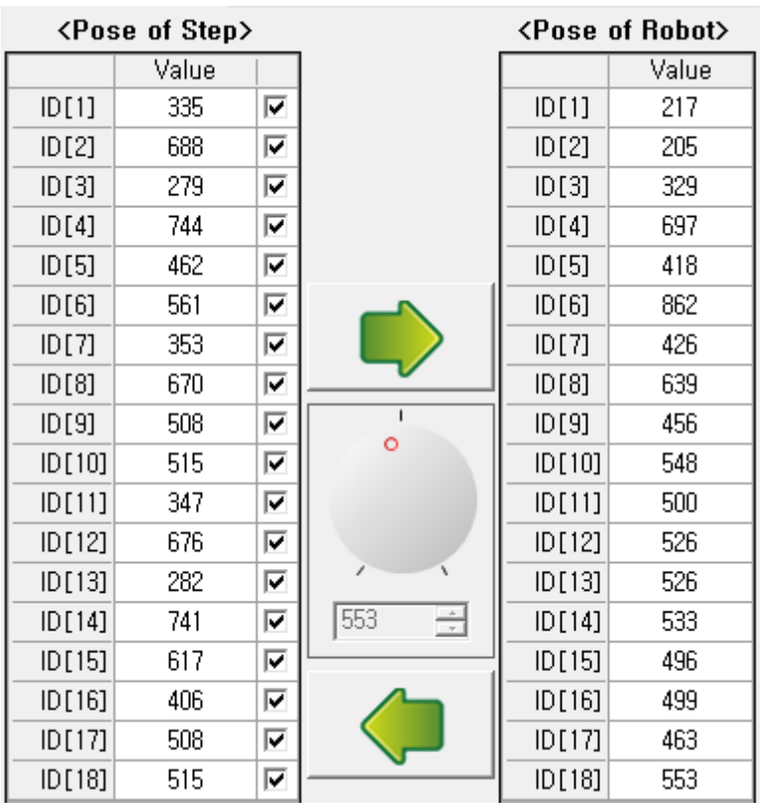
When copying text from another program values are delimited by spaces, new lines, and tabs.

- Copy Pose: Click on "Copy" or press Ctrl+C.
- Paste Pose: Click on "Paste" or press Ctrl+V.

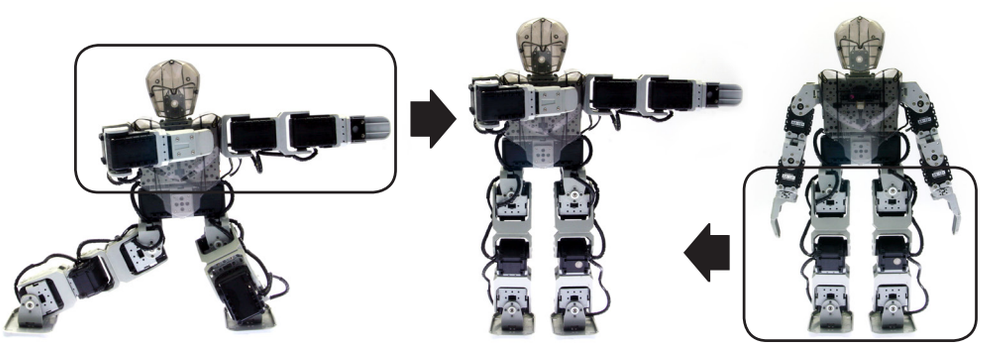


Mask Pose

Masking a pose refers to the process of making a new pose by combining 2 poses by setting whether the value is used while executing or capturing a pose. For example, Pose C may be created by adding the upper body of Pose A with the lower body of Pose B.



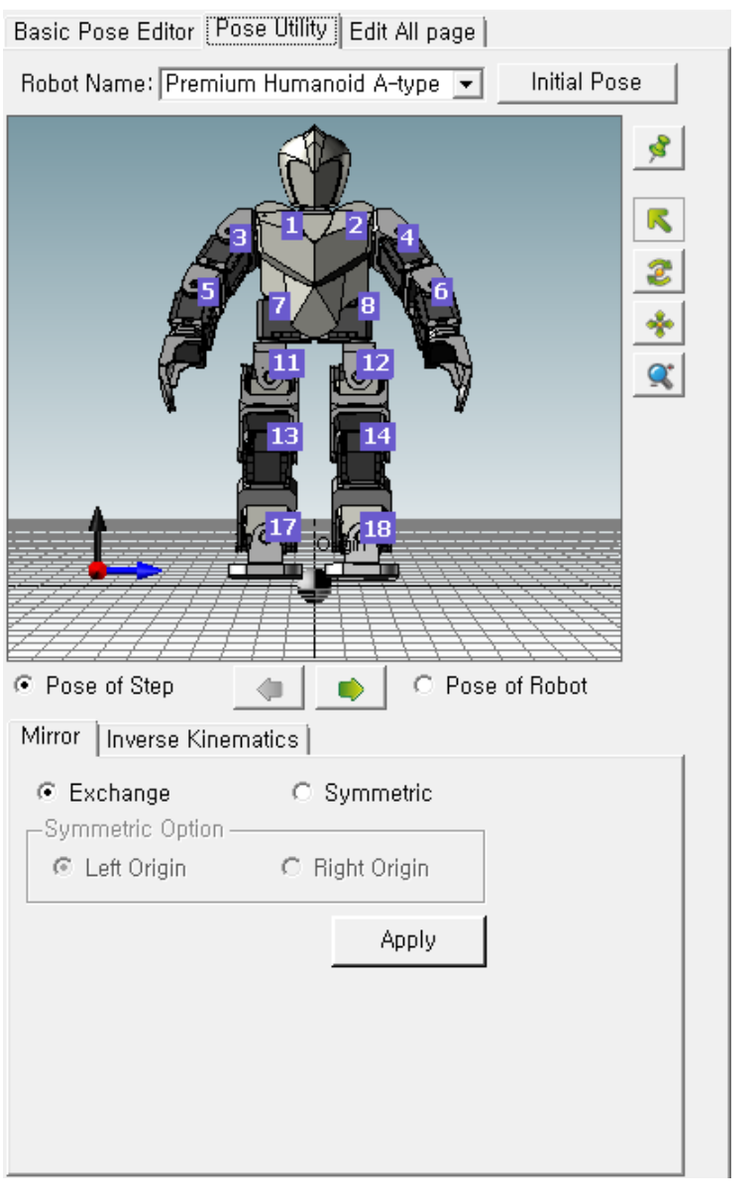
<Pose of Step>			<Pose of Robot>	
	Value			Value
ID[1]	335	☑	ID[1]	217
ID[2]	688	☑	ID[2]	205
ID[3]	279	☑	ID[3]	329
ID[4]	744	☑	ID[4]	697
ID[5]	462	☑	ID[5]	418
ID[6]	561	☑	ID[6]	862
ID[7]	353	☑	ID[7]	426
ID[8]	670	☑	ID[8]	639
ID[9]	508	☑	ID[9]	456
ID[10]	515	☑	ID[10]	548
ID[11]	347	☑	ID[11]	500
ID[12]	676	☑	ID[12]	526
ID[13]	282	☑	ID[13]	526
ID[14]	741	☑	ID[14]	533
ID[15]	617	☑	ID[15]	496
ID[16]	406	☑	ID[16]	499
ID[17]	508	☑	ID[17]	463
ID[18]	515	☑	ID[18]	553



Pose Utility

The pose utility is a tool to easily create a pose based on previously supplied information.

- 3D robot control : Created by moving the 3D robot's joints.
- Mirror: A symmetrical pose can be created or reversed.
- Inverse Kinematics: Accurate positions of each joint can be calculated.

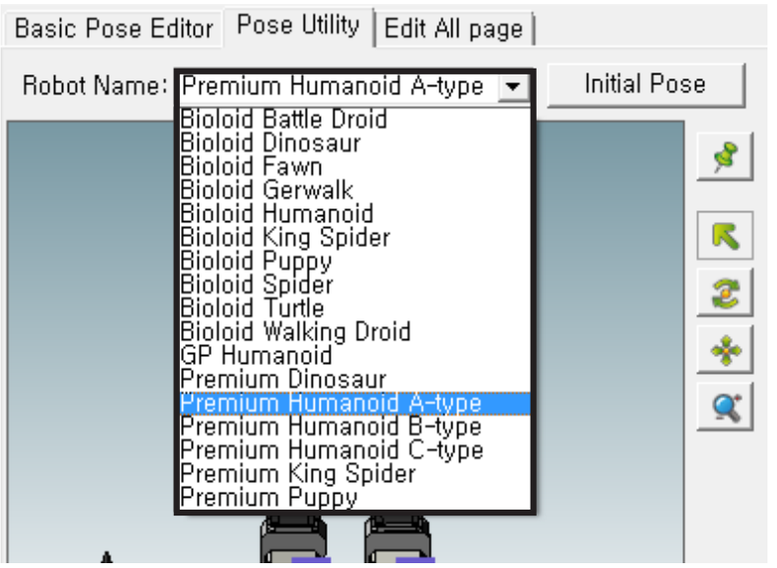


Pose Utility

Robot information is required to create a pose using the pose utility. Therefore, an unlisted robot can not be used. Some robots may not support the functions listed above. Performance for the pose utility may vary based on your graphic card.

Select Robot

Before using the pose utility you must first select the robot. Click the robot's name on the list below to select an applicable robot.



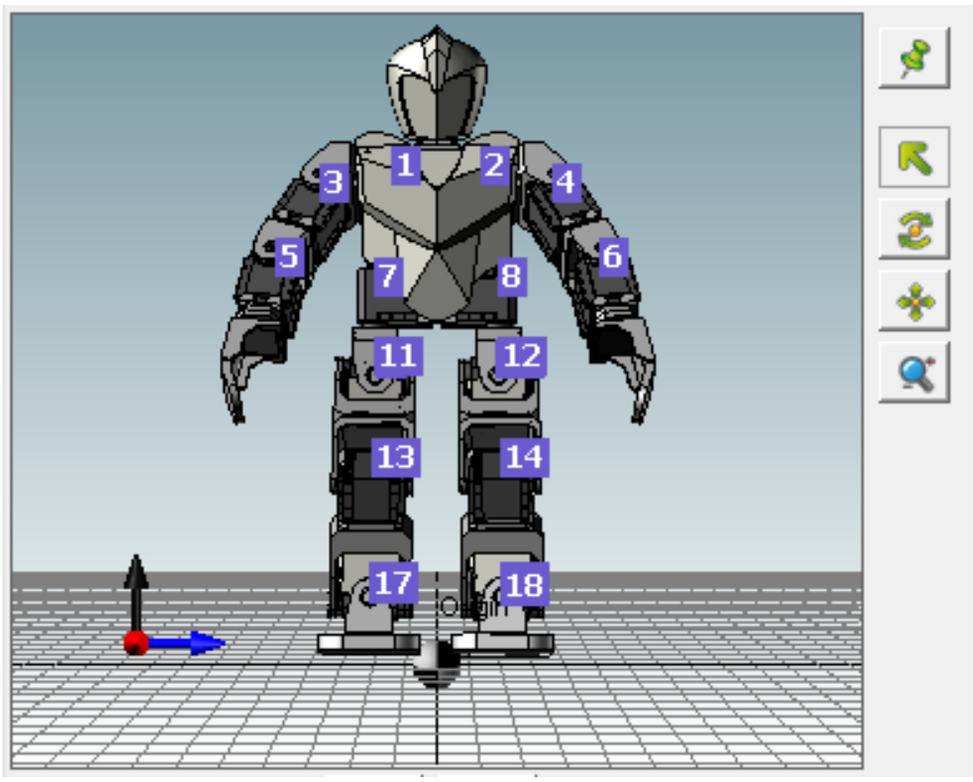
[STEP 1]

Select the name of robot to create a pose for.

Robot Name
…
Premium Humanoid A-type
Premium Humanoid A-type
Premium Humanoid C-type
…

[STEP 2]

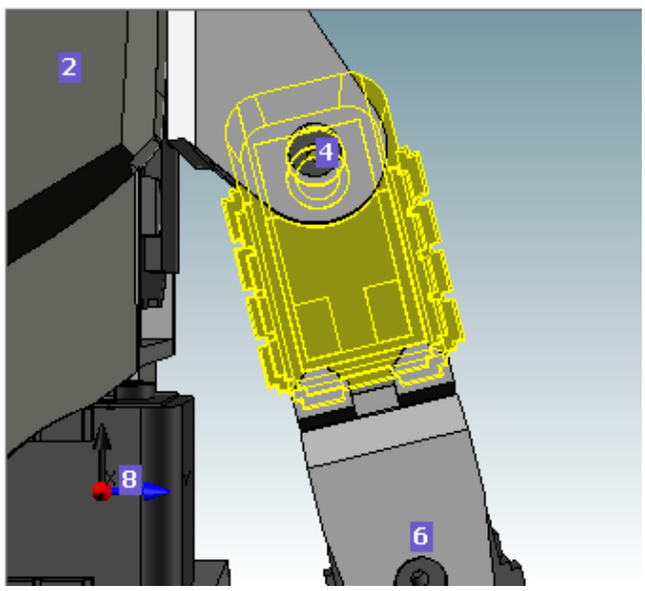
When the "Initial Pose" button is pressed the robot will assume its initial position.



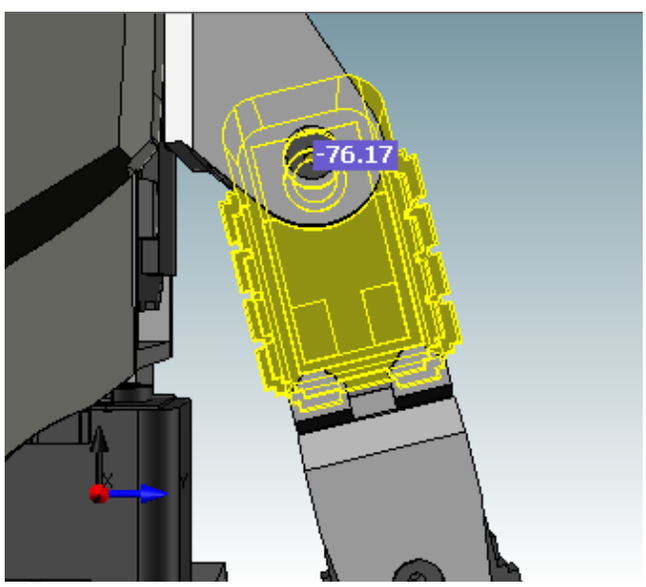
- Zoom Fit : View angle is reset to the initial status.
- Select Objects : Joints can be selected using the mouse cursor.
- Rotate the View : View can be rotated using the mouse. The same thing as above occurs when you press the wheel button of mouse and move.
- Move the View : View can be moved horizontally using the mouse. The same thing as above occurs when you press the wheel button of mouse and move, while pressing the "Ctrl" key.
- Increase/Decrease the View : View can be increased or decreased using the mouse. The same thing as above occurs when you spin the mouse wheel.

Control Joints

The numbers above the robot on the screen are the IDs of the AX-12A. Place your mouse on the ID to change the color of selected AX-12A.



If you click the relevant joint the joint value appears. The joint value is appears as an angle not the motor value.



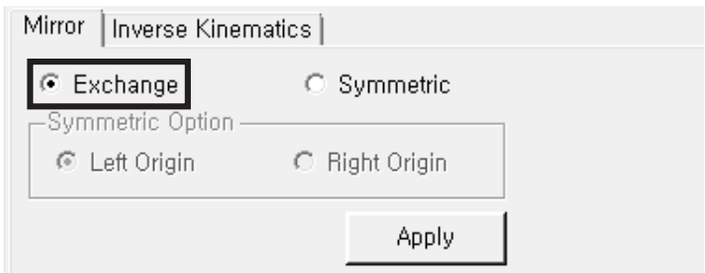
If you move the mouse to left and right while pressing the left button of the mouse the value increases or decreases. In case of 1024-based control, the unit of the value is approximately 0.29(300 / 1024), and in case of 4096-base, it is approximately 0.06(250.92 / 4096).

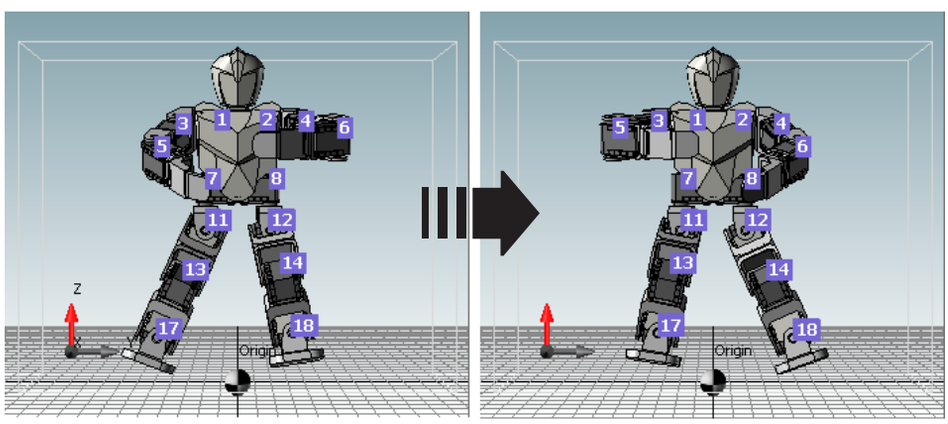
Mirror

The mirror function provides two functions: "Exchange" and "Symmetric." Press "Apply" after choosing the function to apply it to your robot.

Exchange

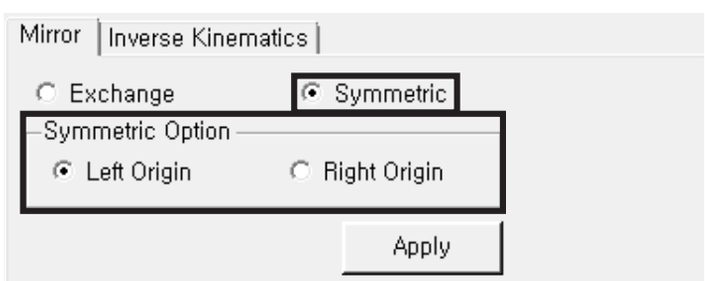
The robot's left side and right side are reversed to create a mirror image of the previous pose.

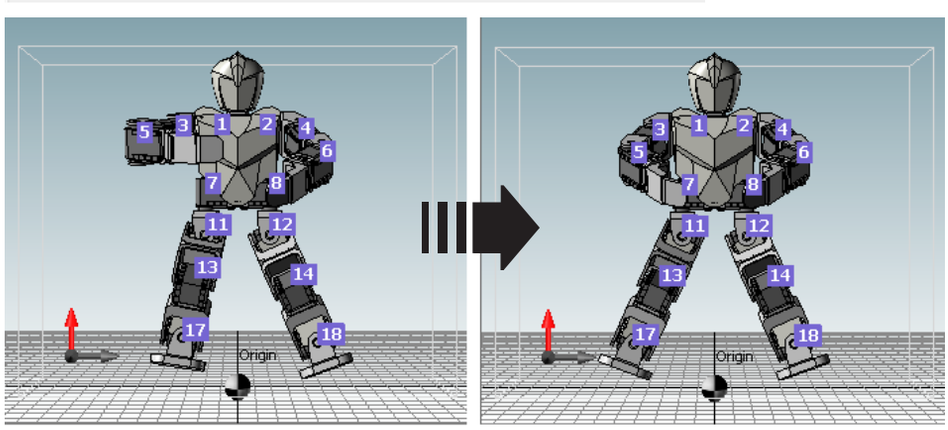




Symmetry

A symmetric pose based on the selected side is created.



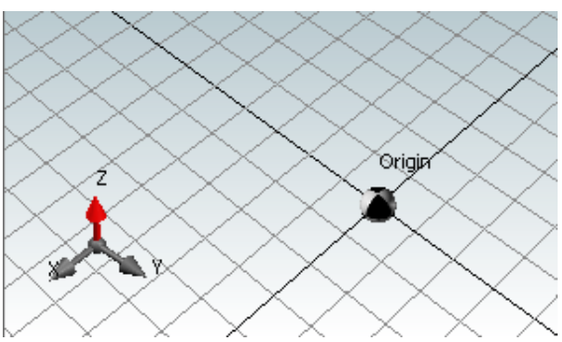


Inverse Kinematics

The mirror function is a widely used and useful tool that provides the two functions below. Select the function then press the "Apply" button to apply to robot.

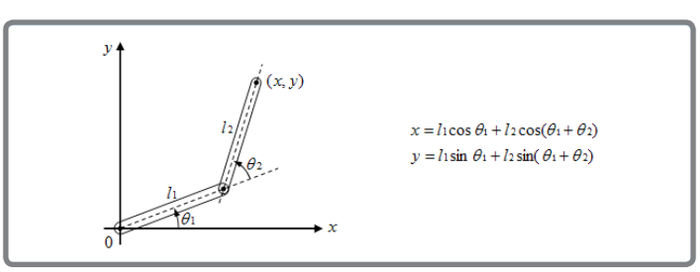
Position and Coordinate System

Understanding the kinematics of the robot's movements start with figuring out where each robot part is located. We must first assign a coordinate point as the origin, and then mark the displacement of each part on the coordinate system. Coordinates axis and origin on the view are shown as below. The unit of the grid is 20mm. Here, Origin means that the coordinates of X, Y, Z is (0, 0, 0).

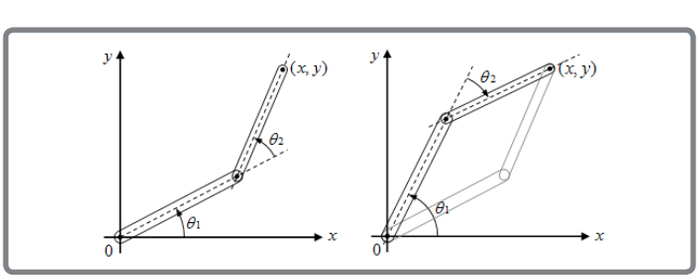


Kinematics and Inverse Kinematics

Kinematics is used to determine the location or movement of the end point from the angle or movement of the joint. In other words, kinematics allows us to determine where the end points once the joint values have been decided. For example, suppose there is a manipulator with two joints in the same plane as shown below. Using the angles of the joints, the coordinate (x,y) of the end point can be determined through kinematics. Kinematics results in only one solution.



On the other hand, inverse kinematics may be used to determine the angle or movement of the joint from the location or movement of the end point. For example, suppose again that there is a manipulator with two joints in the same plane coordinates. If the end point (x,y) has been determined there are 2 possible values for each joint as seen below. When using inverse kinematics, the coordinate (x,y) of the end point may be located at an unreachable distance from the origin or no solution may be obtained due to limitations on joint angles. If more joints are used there may be infinitely many solutions.



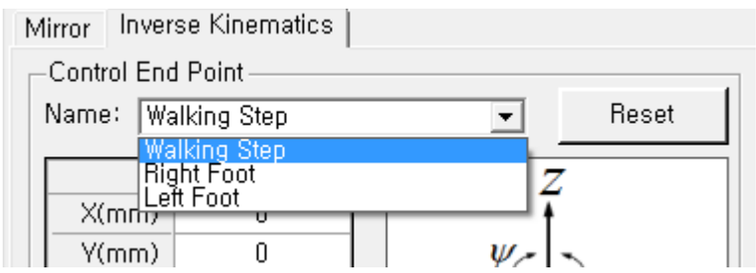
End Point Control

When the user selects how much and in which direction to move the end point, the "Inverse Kinematics" function in the pose utility will calculate the values of each joint and move the end point automatically. This function needs a module to execute "Inverse Kinematics" calculation. Currently, the robots that support "Inverse Kinematics" calculation are as follows :

- ROBOTIS PREMIUM Humanoid
- ROBOTIS PREMIUM Humanoid A-type
- ROBOTIS PREMIUM Humanoid B-type
- ROBOTIS PREMIUM Humanoid C-type

Select the end point

This explanation is based on the ROBOTIS PREMIUM Humanoid Type A.



- Walking Step : Located at the middle of both feet and used to move both feet.
- Right Foot : Located at the center of the right foot and used to move only the right foot.
- Left Foot : Located at the center of the left foot used to move only the left foot.

Initialize the end point

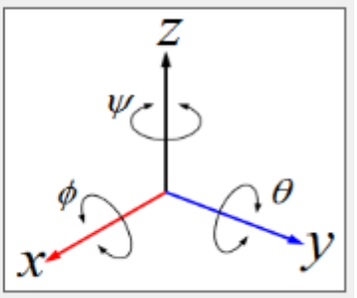
The location of the end point is initialized (All parameters are 0, initial position).

Move the end point

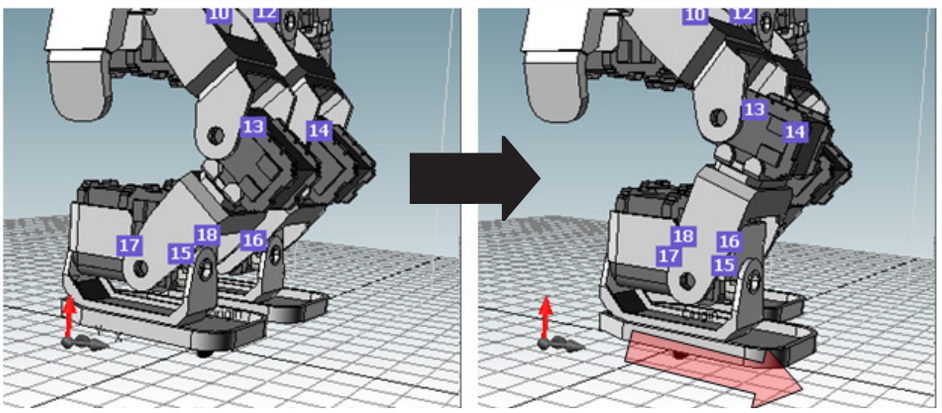
The end point in 3D space can be controlled by 6 parameters. Depending on the structure of robots all the 6 parameters may not be appeared. To change the values, select relevant parameters, and then use the following methods.

- Press [and] to increase or decrease the value by 1.
- Press [and] while pressing "Shift" to increase or decrease the value by 10.
- Double click on mouse or press enter and the control to change the value appears.

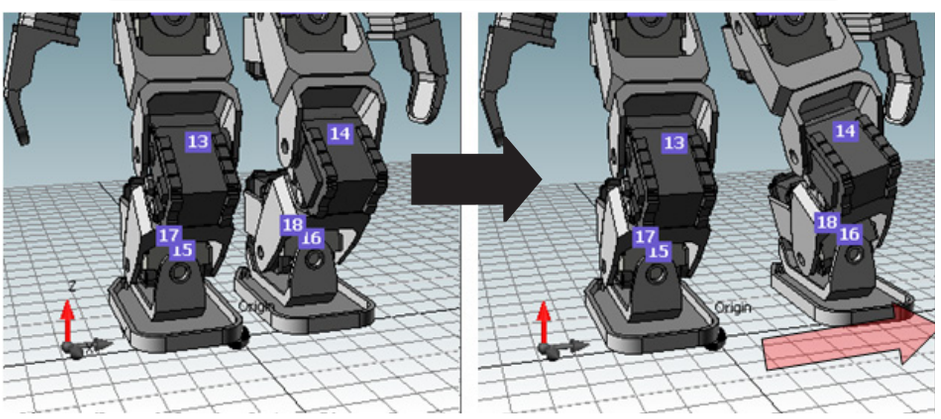
	Value
X(mm)	0
Y(mm)	0
Z(mm)	0
ϕ(°)	0
ϴ(°)	0
ψ(°)	0



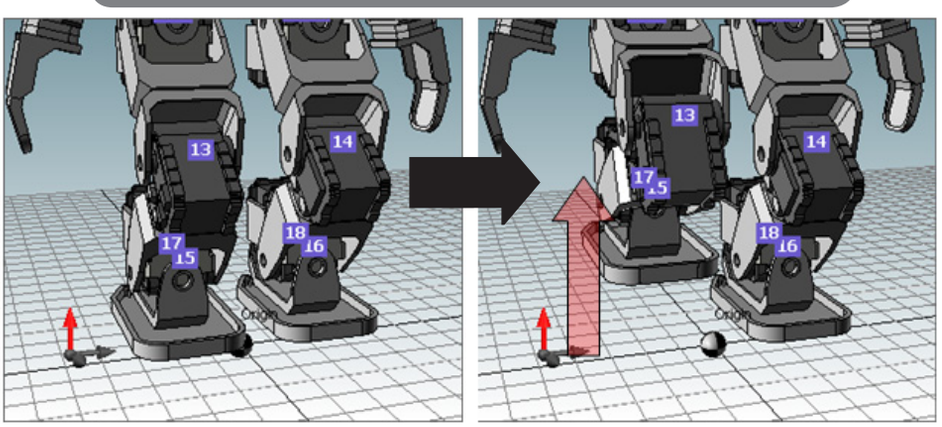
X(mm): it is moved to the X-axis direction by the unit of mm.



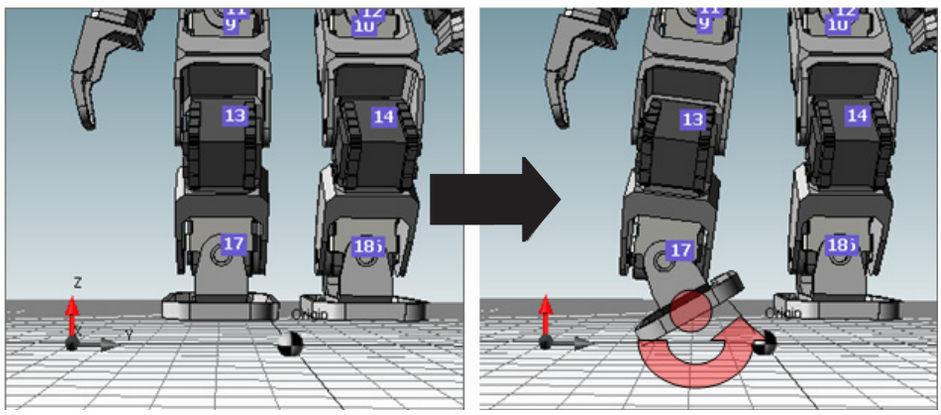
Y(mm): it is moved to the Y-axis direction by the unit of mm.



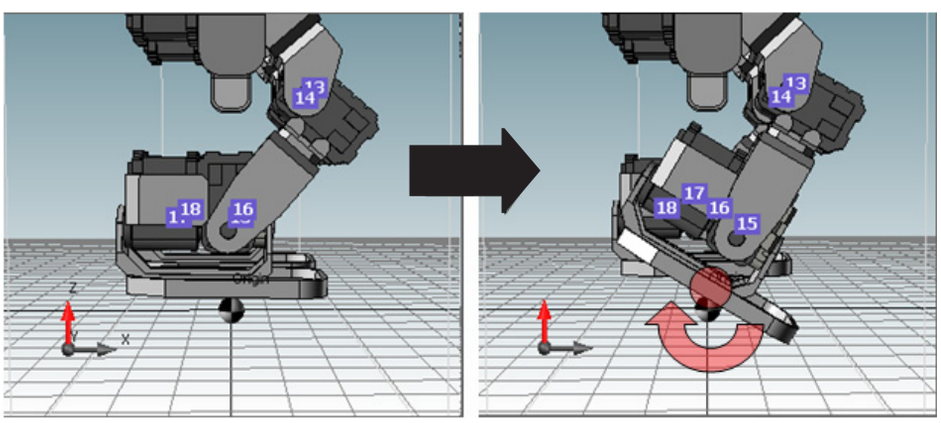
Z(mm): it is moved to the Z-axis direction by the unit of mm.



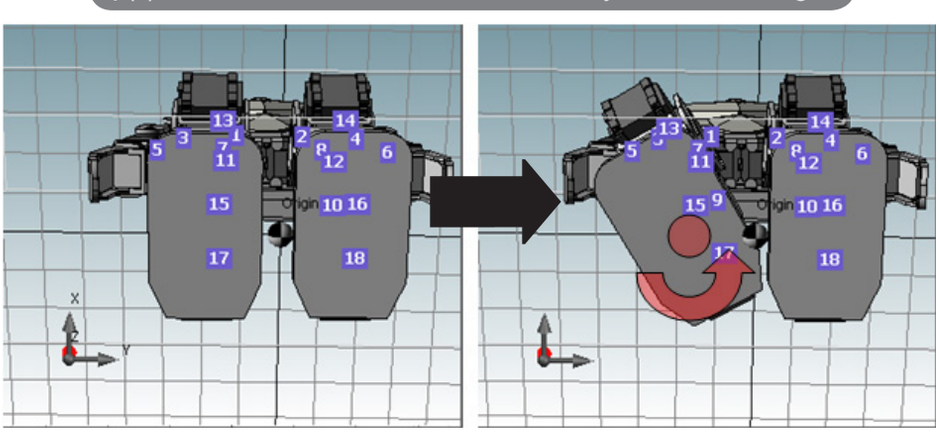
φ(°): it is rotated based on the X-axis by the unit of angle.



θ(°): it is rotated based on the Y-axis by the unit of angle.



ψ(°): it is rotated based on the Z-axis by the unit of angle.

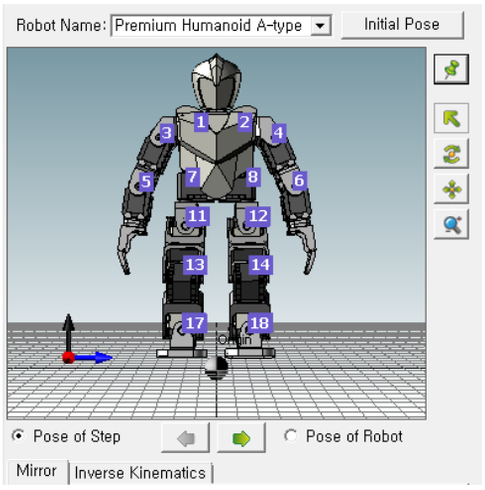


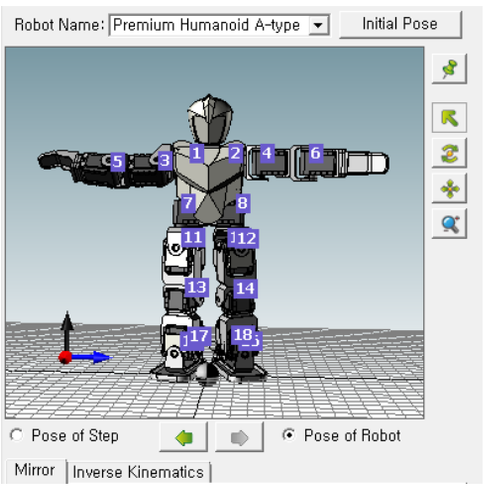
Precaution

Since each parameter has its minimum and maximum values, only the range can change. Sometimes, mathematical results cannot be obtained using inverse kinematics calculation so the situation is called "no solutions" or "Infinite solutions." Due to such fact the parameter values do not change despite their range location. In that case, the solution can be obtained if other parameter values are replaced. For instance, when the legs are straightened until the end (z=0), X or Y parameter is not changed.

Apply the Result

When Pose of Step is selected pose values change. If Pose of Robot is selected pose values change.





Pose of Step refers to the steps on the currently selected motion file. Pose of Robot refer to the robot currently connected. That is, the pose changed in pose utility is immediately reflected to the robot, while Pose of Robot has been selected.

Pose Execution/ Capture

It is the same function as Pose Execution/Capture of the Basic Pose Editor.

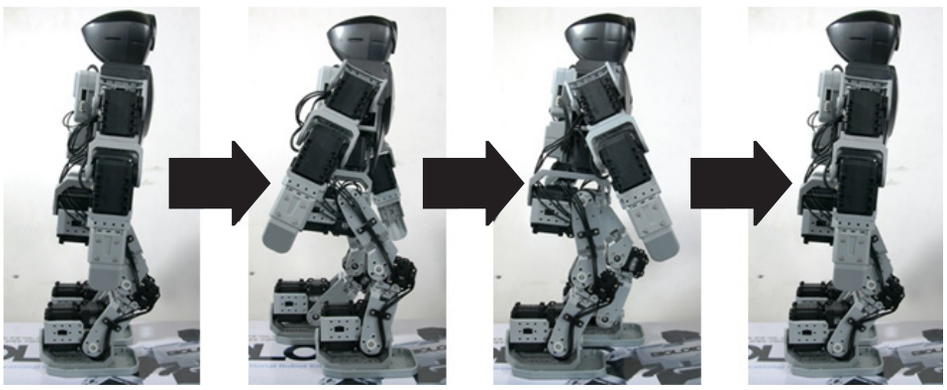
- Pose Execution : Pose of Step is reflected to Pose of Robot
- Pose Capture : Pose of Robot is reflected to currently selected Pose of Step.

Precaution

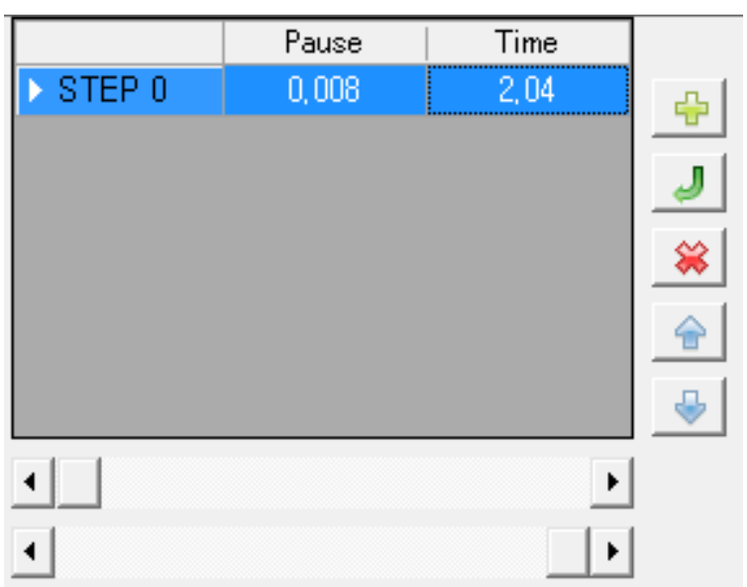
Pose of Step is activated only when there are steps on the current selected page. Pose of Robot is activated only when the robot is connected.

4-3-2 Step Editing

A "Motion Step" refers to the key frames that are required to play consecutive motions.



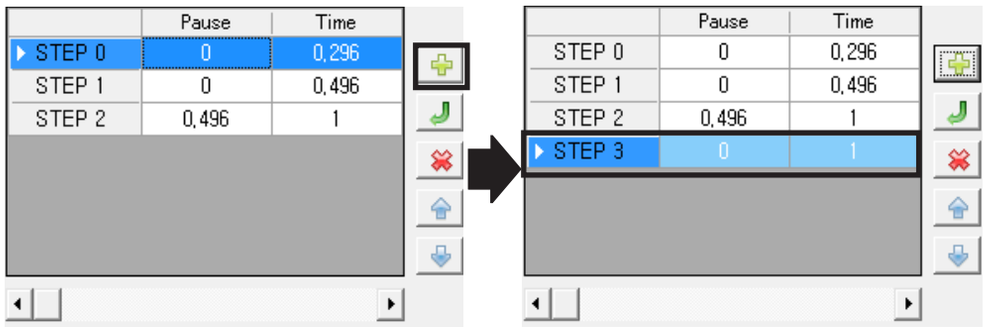
The speed of a motion is determined by the time of each step. The step editor enables steps to be edited easily.



Each page consists of a maximum of 7 steps. To create a motion with more than 7 steps you will need to connect pages.

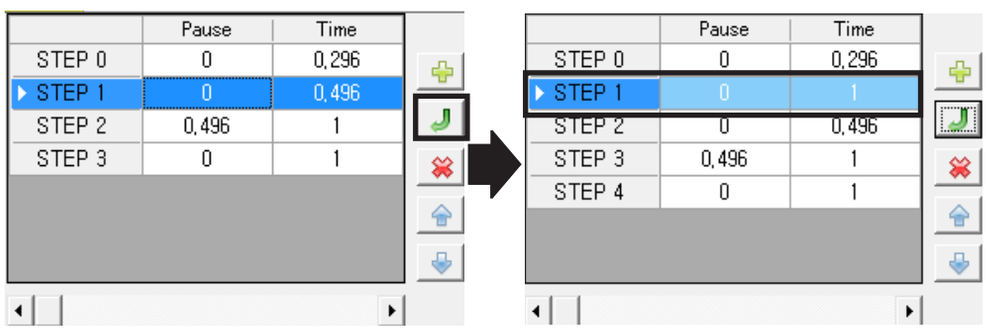
Add Step

A new step is added at the bottom of the step list.



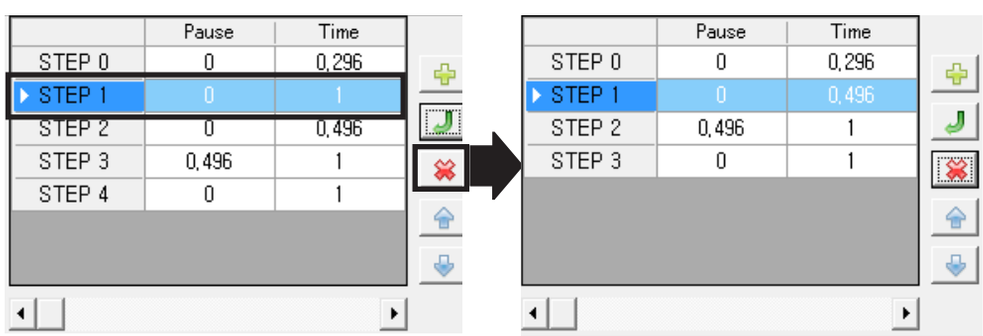
Insert Step

A new step is inserted above the selected step.



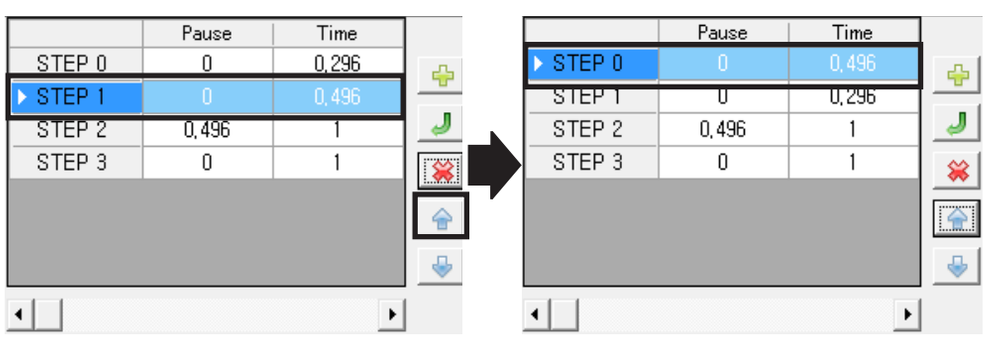
Delete Step

The selected step is deleted from the list.



Change Step Order

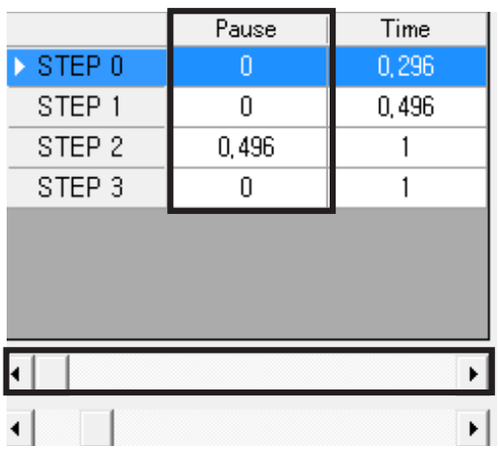
The selected step may be moved up or down.



Pause Step

"Pause" refers to the time between the end of the current step and the start of the next step.

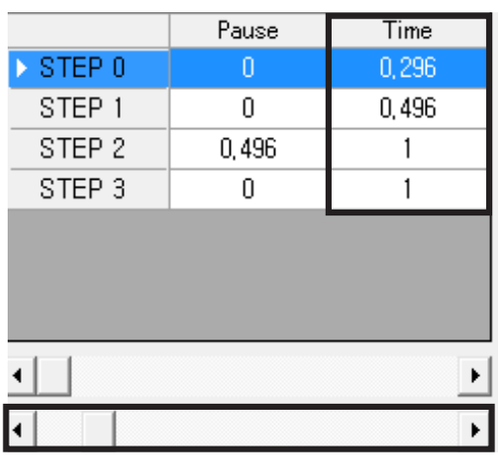
- The unit here is in seconds and can be change in 0.008 increments.
- The value is between 0 and 2.04 seconds.
- The value can be changed using the upper scroll bar.



Step Play Time

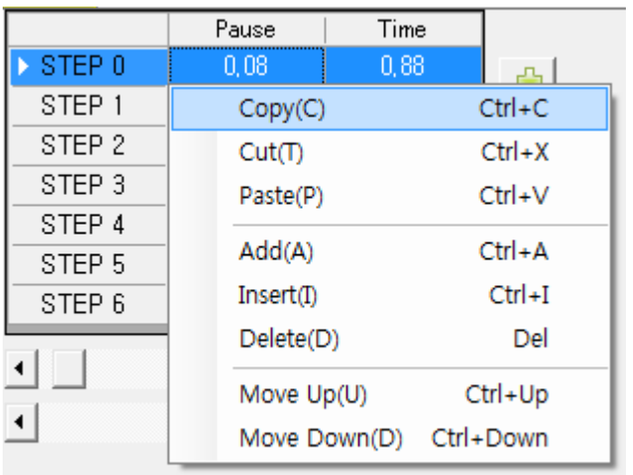
"Time" is the time between the start time from the current step to the end of the current step. The unit here is in seconds and can be change in 0.008 increments.

- The value is between 0.072 and 2.04 seconds.
- The value can be changed using the lower scroll bar.



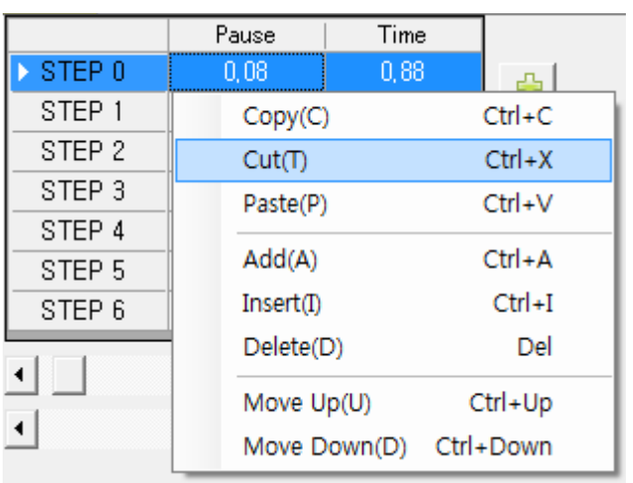
Copy Step

The selected step is copied.



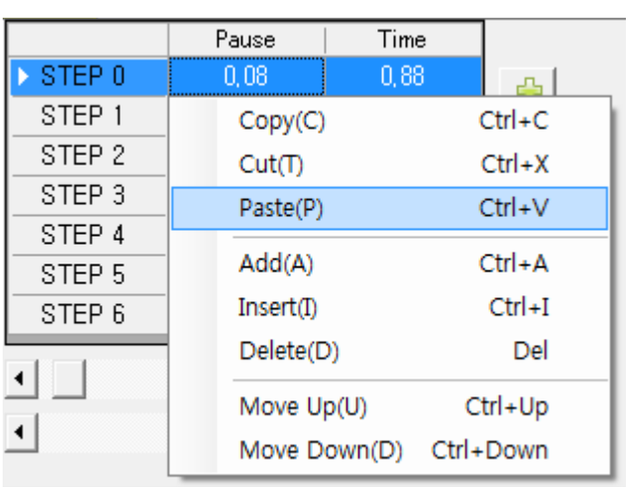
Cut Step

The selected step is cut.



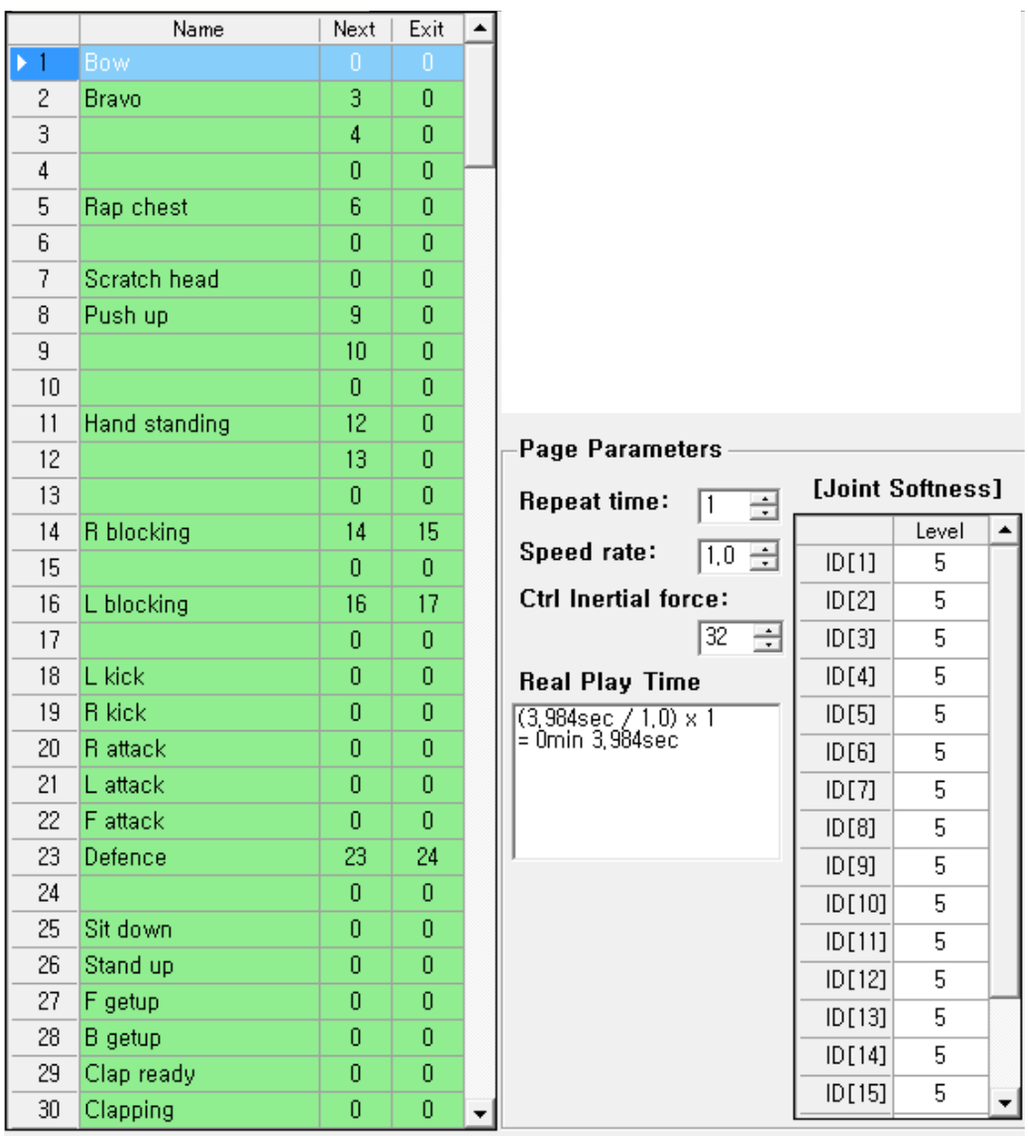
Paste Step

The copied or cut step is pasted.



4-3-3 Page Editing

"Motion page" is the unit used to distinguish between saved motions. Imported motions are read in terms of pages. There are 255 pages in a motion data.



	Name	Next	Exit
1	Bow	0	0
2	Bravo	3	0
3		4	0
4		0	0
5	Rap chest	6	0
6		0	0
7	Scratch head	0	0
8	Push up	9	0
9		10	0
10		0	0
11	Hand standing	12	0
12		13	0
13		0	0
14	R blocking	14	15
15		0	0
16	L blocking	16	17
17		0	0
18	L kick	0	0
19	R kick	0	0
20	R attack	0	0
21	L attack	0	0
22	F attack	0	0
23	Defence	23	24
24		0	0
25	Sit down	0	0
26	Stand up	0	0
27	F getup	0	0
28	B getup	0	0
29	Clap ready	0	0
30	Clapping	0	0

Select Page

Click on a row to select a page.

The following methods may be used to select multiple pages.

To choose pages in consecutive order

	Name	Next	Exit
1	Bow	0	0
2	Bravo	3	0
3		4	0
4		0	0
5	Rap chest	6	0
6		0	0
7	Scratch head	0	0
8	Push up	9	0

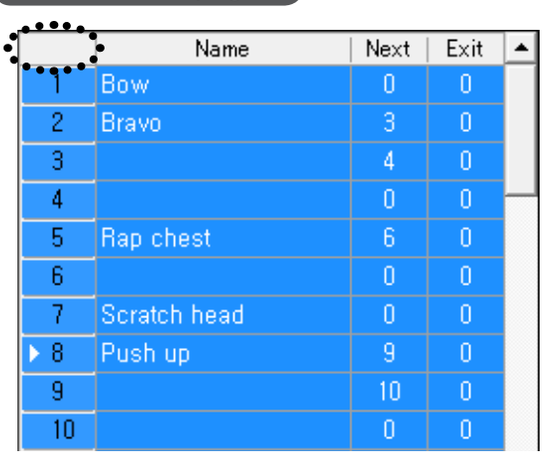
- Drag with the mouse
- Choose pages while holding down the "Shift" key.

To choose pages separately

	Name	Next	Exit
1	Bow	0	0
2	Bravo	3	0
3		4	0
4		0	0
5	Rap chest	6	0
6		0	0
7	Scratch head	0	0
8	Push up	9	0
9		10	0
10		0	0

Choose pages while holding down the "Ctrl" key.

To choose all pages



	Name	Next	Exit
1	Bow	0	0
2	Bravo	3	0
3		4	0
4		0	0
5	Rap chest	6	0
6		0	0
7	Scratch head	0	0
8	Push up	9	0
9		10	0
10		0	0

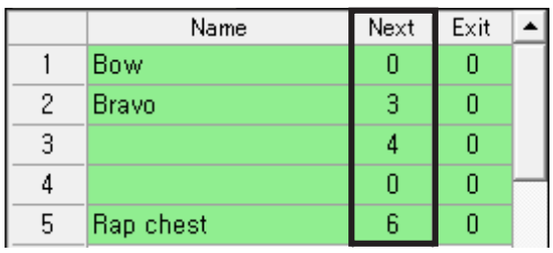
Press the button in the upper left corner.

Connect Page

Pages can be connected with each other when necessary.

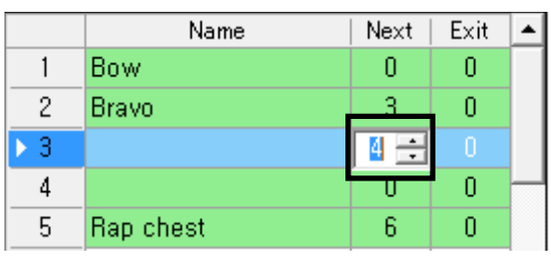
Next Page

A single page can have a maximum of 7 steps. Therefore, some motions may not fit in one page. To use multiple pages for one motion designate the page to link to.



	Name	Next	Exit
1	Bow	0	0
2	Bravo	3	0
3		4	0
4		0	0
5	Rap chest	6	0

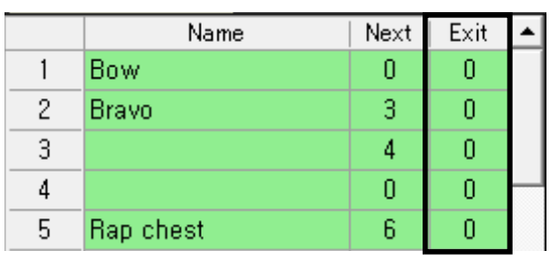
Enter the number of the next page in the "Next" column.



	Name	Next	Exit
1	Bow	0	0
2	Bravo	3	0
3		4	0
4		0	0
5	Rap chest	6	0

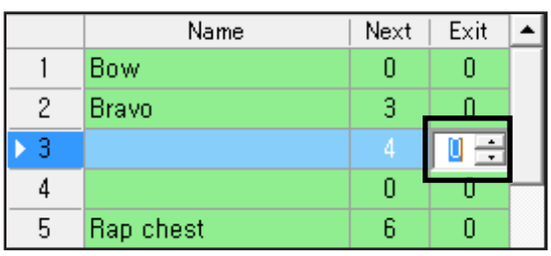
Exit Page

When commands to stop a motion are made, the robot will usually be in a very unstable state due to the motion being executed. To stop a motion in a stable state, designate an exit page.



	Name	Next	Exit
1	Bow	0	0
2	Bravo	3	0
3		4	0
4		0	0
5	Rap chest	6	0

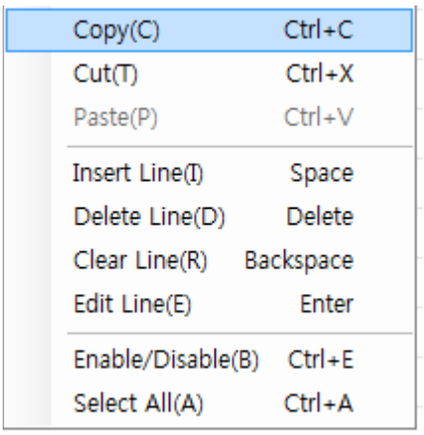
Enter the number of the exit page in the "Exit" column.



	Name	Next	Exit
1	Bow	0	0
2	Bravo	3	0
3		4	0
4		0	0
5	Rap chest	6	0

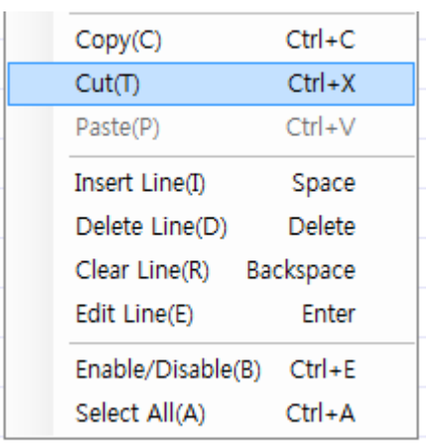
Copy Page

Copy the selected page.



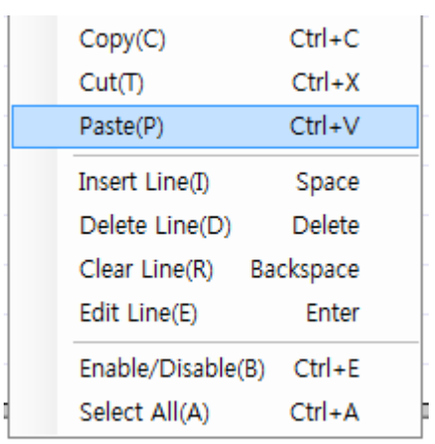
Cut Page

Cut the selected page.



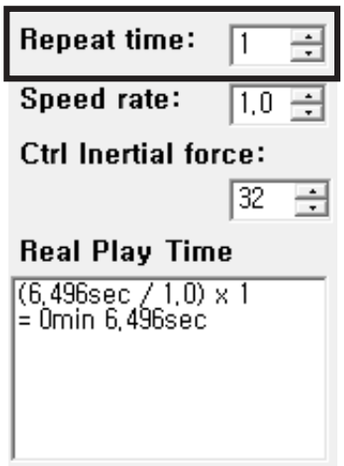
Paste page

Paste copied or cut page. The contents of selected page is overwritten.



Set Page Repeat/Time

Repeat Time : This is the number of times the current page is repeated during motion execution

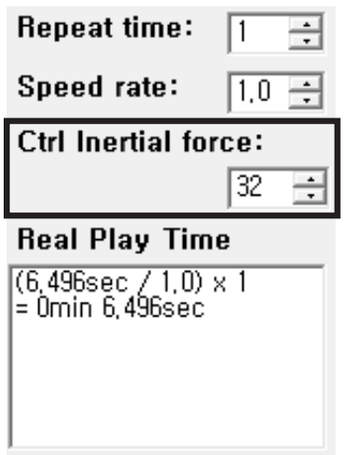


Speed Rate : This is the playback speed of the page during motion execution. Unlike the "Step Time," this applies to the entire page.

- If the speed rate is 1.0 the page will be executed at normal speed.
- If the speed rate is lower than 1.0 the execution speed will decrease.
- If the speed rate is higher than 1.0 the execution speed will increase.

Inertial Force Control

Inertial Force is a force generated between steps as a result of the Law of Inertia. In general, inertial forces are created by acceleration, which is the change in speed. That is, as acceleration increases, inertial force also increases; and, as acceleration decreases, inertial force also decreases. To reduce acceleration increase or decrease the speed gradually; and to increase acceleration, change the speed drastically. "Ctrl Inertial Force" is used to control this acceleration. Increase this value to increase or decrease the speed gradually reducing the acceleration.



The value is between 0 and 127. (Default is 32.)

- The closer the value is to 0 the greater the inertial force.
- The closer the value is to 127 the lower the inertial force.

Inertial Force Flexibility

Joint softness is used to set the compliance of the AX-12A. Below are the pros and cons of different joint softness values :

When the joint softness is big

- Pro: Movement is smooth. Used for fluid movements, such as dancing.
- Con : May not be suitable for legs that need much support.

When the joint softness is small

- Pro: Movement is stable. Used for movements that require support such as walking.
- Con: Movement may look too rigid when performing fluid motions.

There are 7 joint softness levels. (level 5 is default value)

- Level 1: Almost none (Not recommended)
- Level 2: Very Low
- Level 3: Low
- Level 4: Somewhat Low
- Level 5: Average (Default)
- Level 6: High
- Level 7: Very High

[Joint Softness]

	Level
ID[1]	5
ID[2]	5
ID[3]	5
ID[4]	5
ID[5]	5
ID[6]	5
ID[7]	5
ID[8]	5
ID[9]	5
ID[10]	5
ID[11]	5
ID[12]	5
ID[13]	5
ID[14]	5
ID[15]	5

4 - 4 Advanced Learning

4 - 4 - 1 Humanoid Robot's Gait Principle

A two wheeled robot can easily be programmed to move forward/backward and to turn by controlling the direction and speed of each wheel. However, when a robot has joints, such as a biped, quadruped, or hexapod, it needs a behavioral pattern for its joint movements. A task to create and to adjust this behavioral pattern is called 'Walking Motion Generation.' A robot's walking motion is to move appropriately in the given situation meaning it needs a lot of motions. There are several examples of walking motion for other robots as well as quadruped/hexapod robots, but what interests people the most is the humanoid (biped). Thus there are many humanoid competitions. Let's learn about the humanoid's gait using with the simple practice and the motion example provided.

For a humanoid robot to walk you need to use either the Pose Utility, the customized user defined walking motion, or the pre-programmed walking motion. The basic walking motion example is created by a program called a Motion Generator. Here, you will need a task code to call on this motion.

For a detailed omni-directional walking motion there are more things to consider.

Walking motion can be created in many different ways depending on the user's intention, but it follows the instruction below in general.

Motion Generator

Motion generator automatically generates the motion when you set up the variables such as how to swap left/right legs while walking, how much it would lift up the leg, the duration of walking per cycle, etc. When you use this program it creates the walk-in-place motion first and then decides on the little details for the each foot. It is the same idea with what we are to discuss soon except on how to control the variables and its efficiency.

1 Hardware Preparations

Check if there are any broken frames or any loose parts in between frames and/or actuators. Changes in parts will alter kinematics adversely affecting robot gait generation.

2 Initial Position

If the initial position is tilted to one side it may change its direction to the tilted side or make a turn while walking. Use the default value for the basic motion and adjust the balance with the motion offset editor. It is better to do this on the ankle joint parts only.

3 Generating "Walking-in-place" motion

This is to decide the general way of walking and speed and to generate continuous walk-in-place motion by setting the left/right foot swapping, the foot's lift height, speed, etc. We will explain about this in detail through practice.

4 Generation Basic Motion

This is to give the additional change to the walk-in-place motion and to generate forward, backward, pivot left, pivot right, left turn, right turn, etc.
While in walk-in-place motion change the step value of the hip joints, front, back, left, right, ankle front, back, left, right to generate turning and walking.

5 Generating additional walking motion

Generate an additional walking motion such as move forward/backward when necessary, turn left/right as it moves forward/backward, etc
If you adjust the values of left/right joints and/or turn-related joints in addition to "walk-forward" motion you can create the motion patterns such as turning as moving forward, or move forward diagonally, etc.
If you adjust the values of turn-related joint in addition to "walk-sideways" motion you can create motion patterns such as turning as walking sideways, etc.

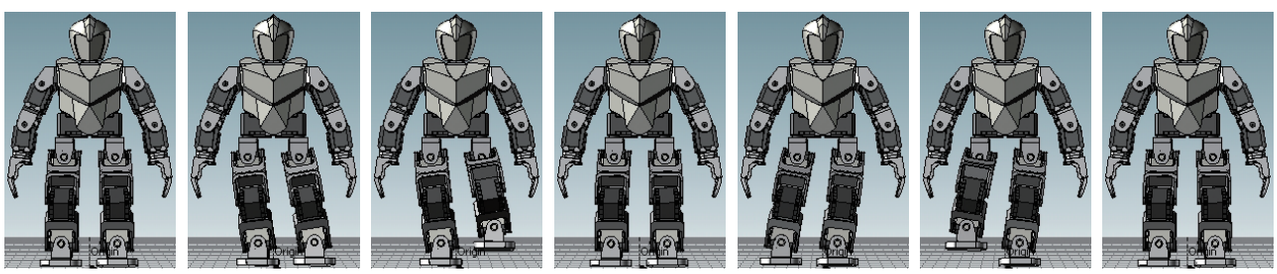
6 Use the created motion

You can call on the created motion with the remote controller (wireless robot control with RC-100B for robot battle or other robot competitions) or use it in the appropriate situation (Avoiding obstacles, other tasks, etc).

4 - 4 - 2 Practice to create " walking-in-place" motion

Walk-in-place

Walk-in-place motion is very important because it is very basic of all walking motions and is needed to decide the position of the center of gravity or the moving speed. Making the motion in symmetry is easier to create and to adjust the motion. As you test the walk-in-place motion continuously you need to adjust its walking posture and all movement to fix wrong movements and postured. Walk-in-place motion can be created in many other ways depending on the user's intention.



Step 0 Step 1 Step 2 Step 3 Step 4 Step 5 Step 6

Basic Cycle

The basic cycle consists of 6 steps like in the image. Initial step 0 is the duplication of step 6 when all the motions are repeated, so let us just begin with 7 steps for now. When it repeats too much, delete one of them.

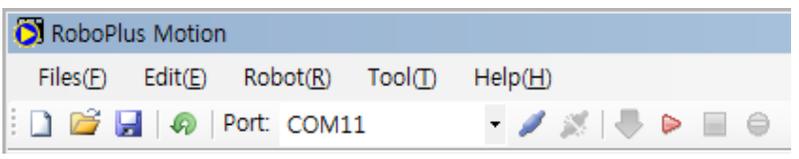
Step 0. Initial Position
Step 1. Move the center of gravity of the body with one foot.
Step 2. Lift the leg that is not supporting the body in Step 1
Step 3. Initial Position
Step 4. Move the center of gravity of the body with the other foot.
Step 5. Lift the leg that is not supporting the body in Step 4
Step 6. Initial Position

By creating 2 steps (Step 1 - change 4 joint values, Step 2 - change 3 joints value) and using the mirror function of the Pose Utility you can easily create the 6 basic steps for the Walk-in-Place motion.

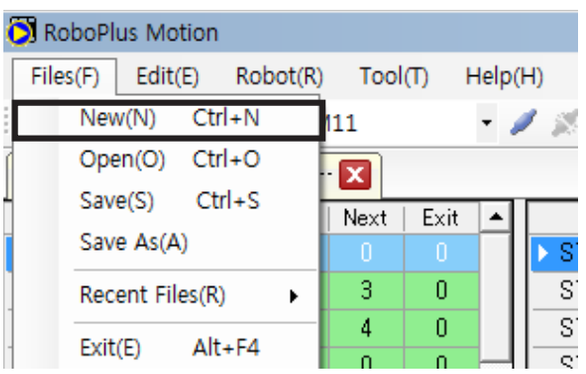
Now, let's generate the motion to walk in place like we explained above. Before starting, we should be prepared to organize and to back up the motions.

1. Run RoboPlus Motion.

2. Turn on the robot and connect it to the PC

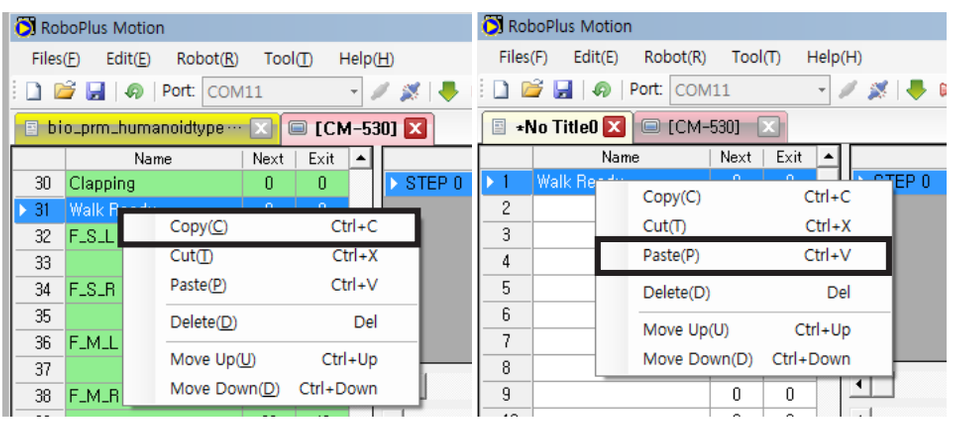


3. Create a new file motion.

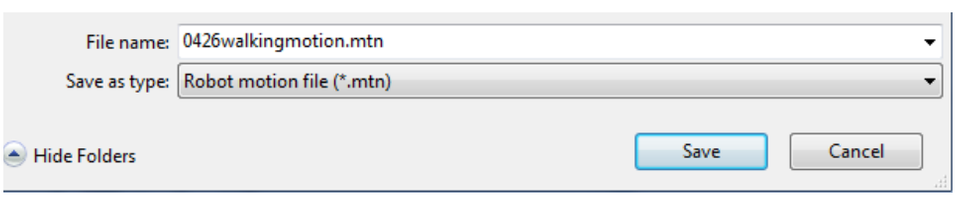


Use file motion to generate basic motions for back up.

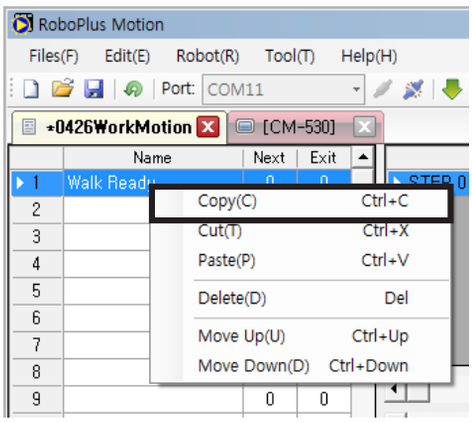
4. Copy page #31 from Robot motion to page #1 of File Motion.

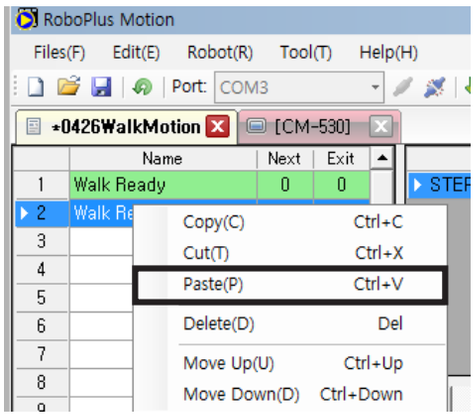


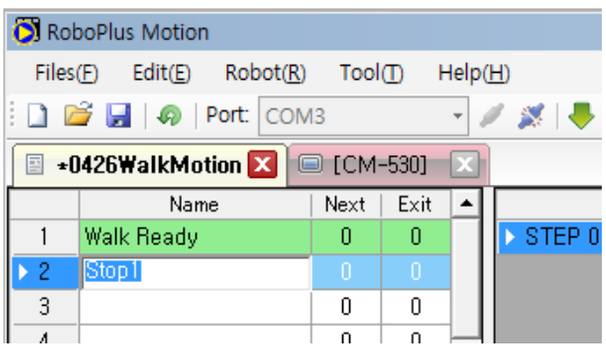
5. Save the file. We recommend naming it after the date and type of the motion. i.e '0426WalkingMotion.mtn"



6. Leave page #1 for back up, copy it to page #2, and change the name to "WalkinPlace_1"





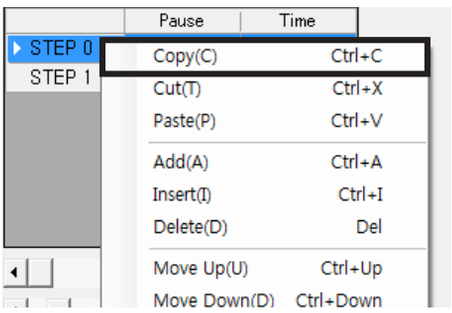


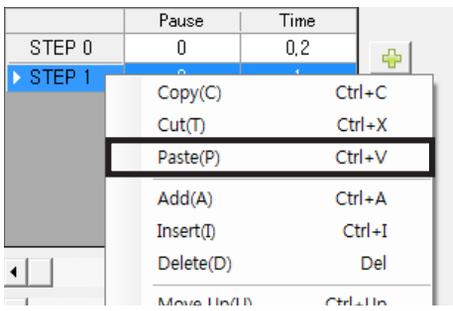
Precaution

The battery weight or the connection of the power supply cable may affect the generation of the walking motion. Always equip your robot with the battery even if you are not using it and support the power supply cable with your hands to minimize the deflection. It is highly recommended to use a fully charged batter instead of the power supply cable.

Now, let's generate the basic walk-in-place motion cycle.
It is better to use the shortcut for fast walking motion generation.

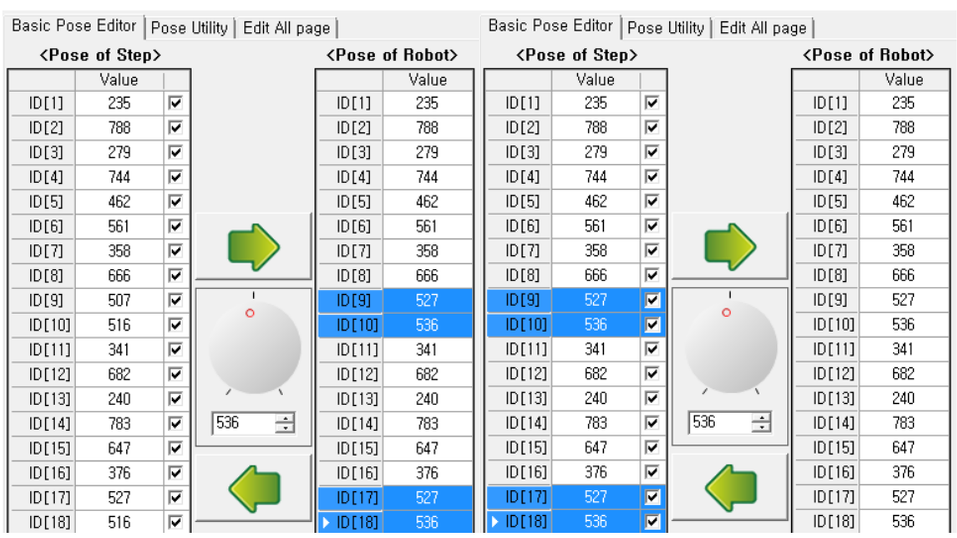
1. Go back to the "WalkinPlace_1" page, add steps and copy step 0 to step 1.





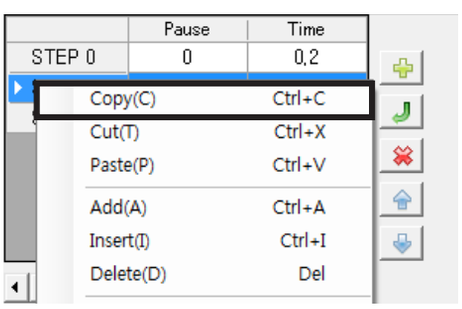
Shortcut Ctrl+C, Ctrl+V

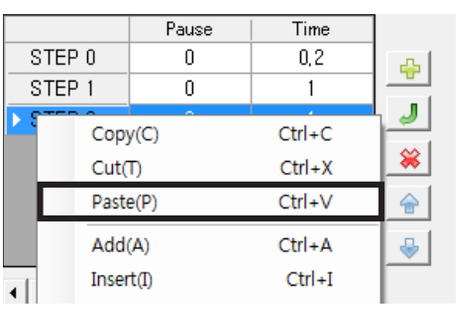
2. Move the center of gravity to one foot by adjusting the hip joints and left/ right ankle joints in Step 1.



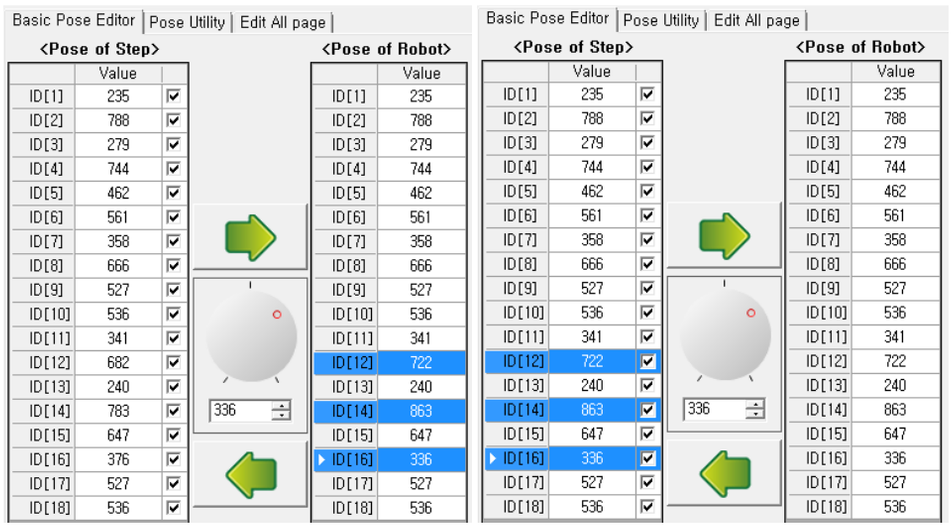
- Move Step 1 to the Pose of Robot
- Press 'Shift +]' twice to increment ID[9], [10], [17], [18] by 20
- Move the Pose of Robot to Step 1

3. Add Step 2 and Copy Step 1.



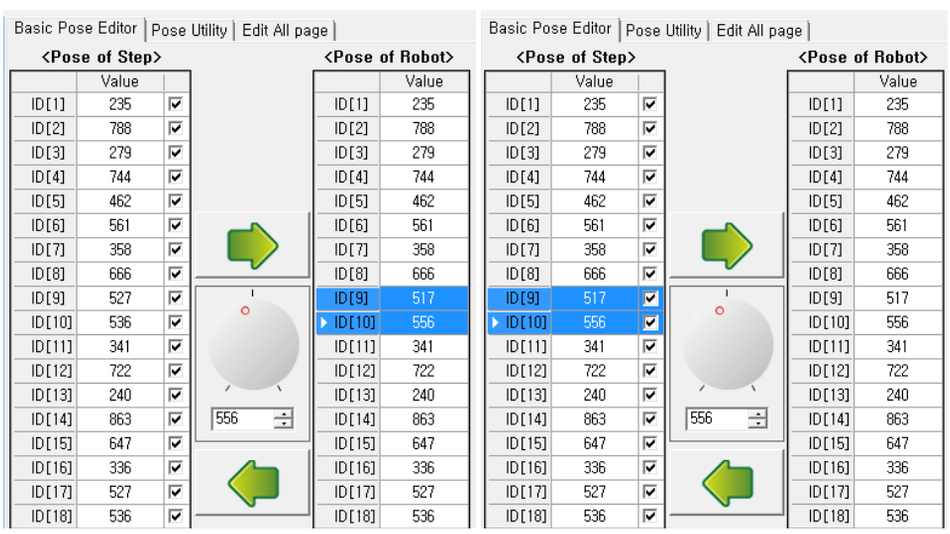


4. Lift one foot by adjusting front and back of the left hip joint, knee, and front and back ankle in Step 2.



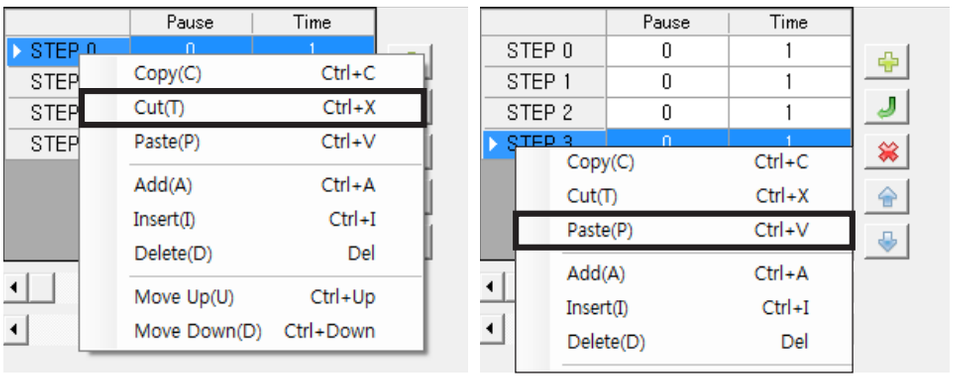
- Move Step 2 to the Pose of Robot
- Use shortcut 'Shift' + '[', ']' to increase ID [12] by 40, ID [14] by 80, and decrease ID [16] by 40
  (To lift the leg horizontally, double the knee joint value).
- Move the Pose of Robot to that of Step 2

5. Move the left/right hip joints towards the direction where it supports the leg in Step 2.



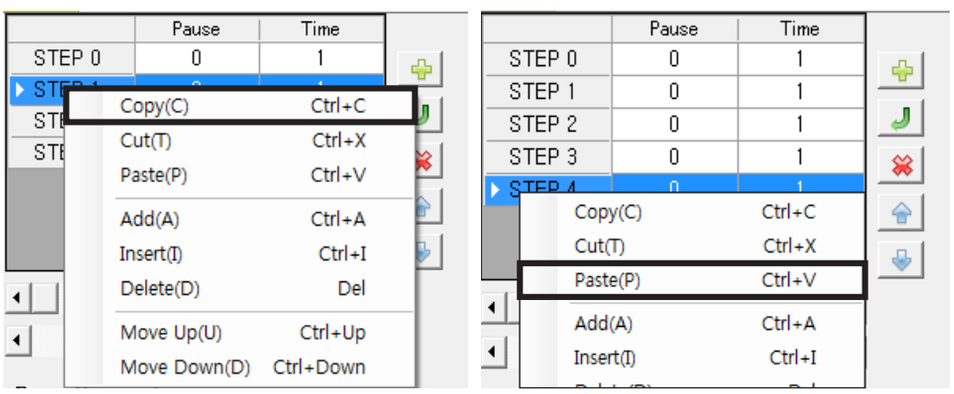
- Because of the deflection due to gravity as it lifts up one leg the robot may step on its own footpad while walking. Adjust the left/right hip joints values against the gravity to compensate this problem.
- Use shortcut 'Shift'+'['.']' to decrease ID[9] by 10 and ID[10] by 20
- Move the Pose of Robot to that of Step 2.

6. Add Step 3 and Copy Step 0.



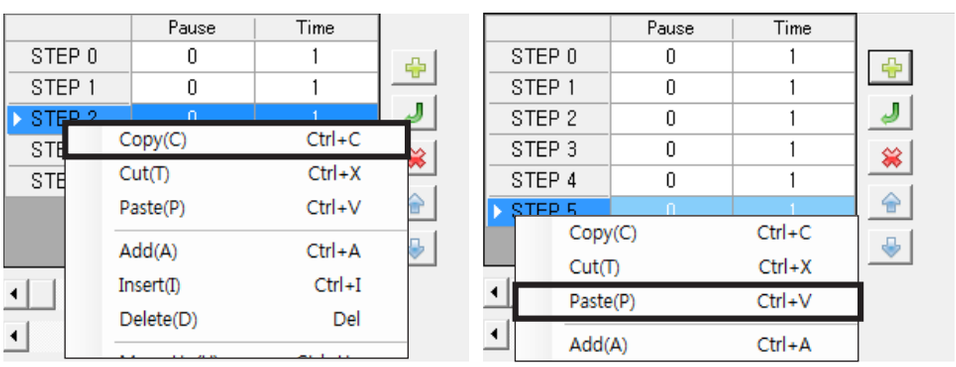
Shortcut Ctrl+C, Ctrl+V

7. Add step 4 and copy step 1.



Shortcut Ctrl+C, Ctrl+V

8. Add step 5 and copy step 2.

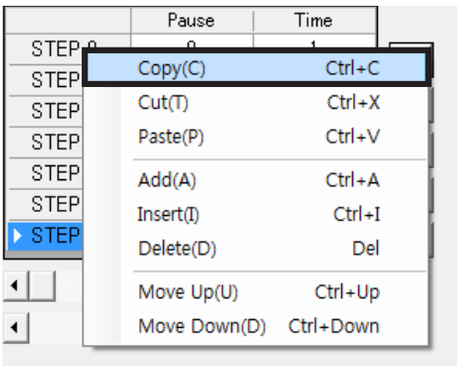


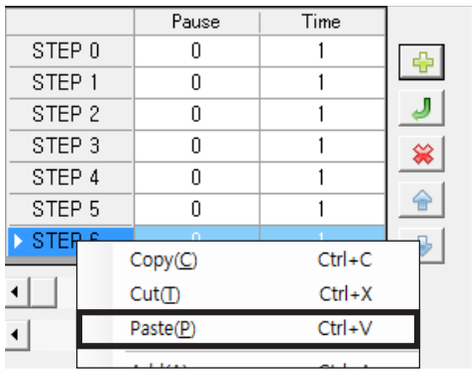
Shortcut Ctrl+C, Ctrl+V

Precaution

When using shortcut 'Shift'+'[',']', please select one joint at a time.

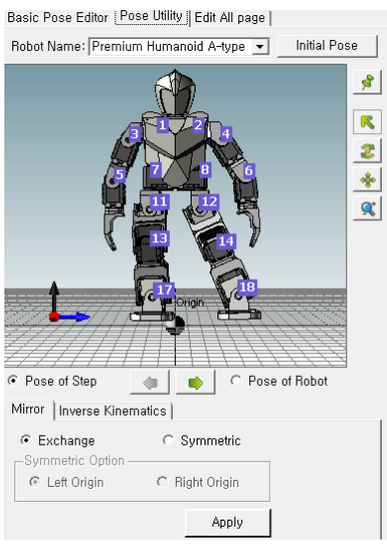
9. Add Step 6 and Copy step 0.

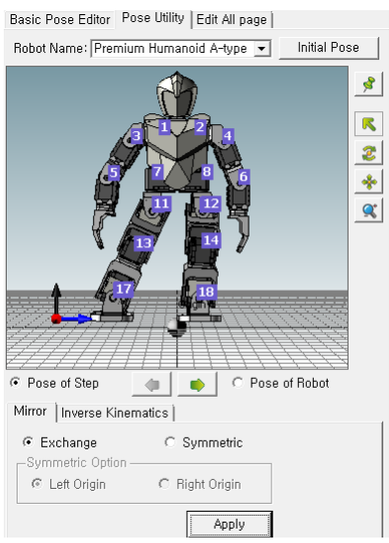




Shortcut Ctrl+C, Ctrl+V

10. Use the Mirror Exchange of Pose Utility to flip Step 4.

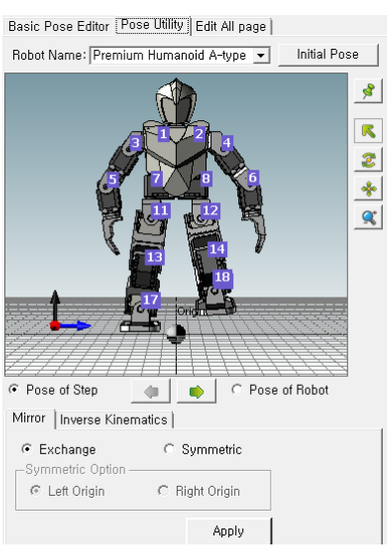


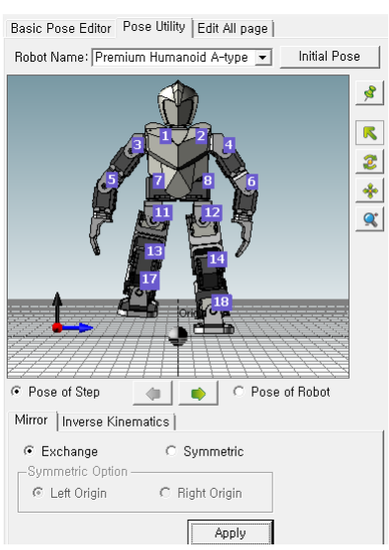


Step 4 - Front

Step 4 - Back

11. Use the Mirror Exchange of Pose Utility to flip Step 5.



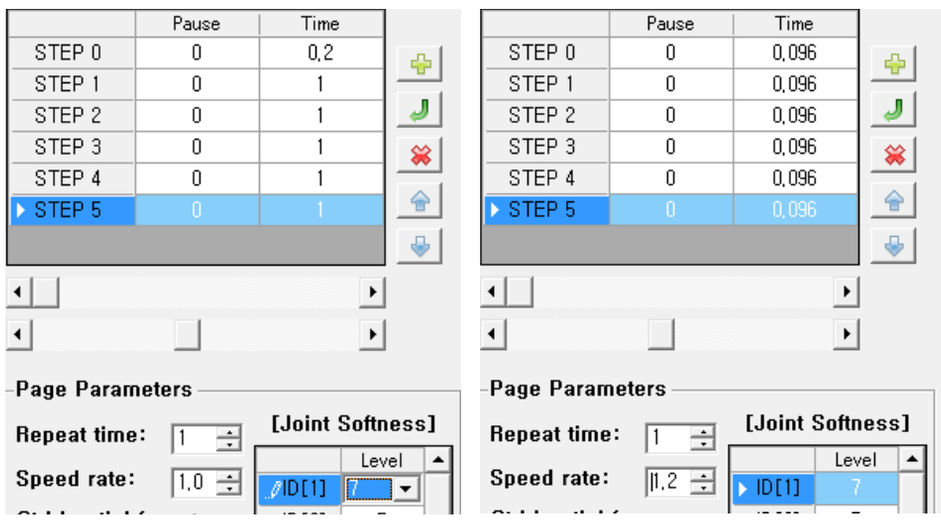


Step 5 - Front

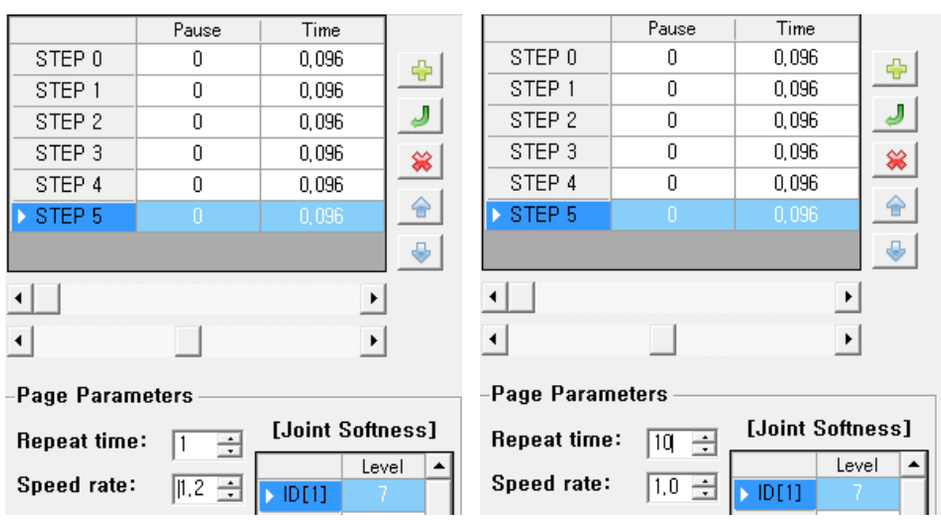
Step 5 – Back

12. Delete Step 0 since it is a duplicate of the last step (Step 6).

Run this page #1 made of 6 steps above to find the appropriate speed value for the each, and entire steps.



13. Run this value continuously(5~10 times) to find the appropriate speed value for the continuous movement.



Let's find your own stable walking motion by adjusting the change of barycenter while walking(the change of Step 0 and 3), the gravity compensation for each leg (the change of Step 1 and 4), the flexibility of the ankle, etc.

Back up the walking pattern(i.e. walkinplace_1, walkinplace_2, etc) as you go and generate additional walking-in-place motion. As you change these values to generate motions you can easily create other desired motions like static gaits, dynamic gaits, running, etc.

The flexibility of the ankle

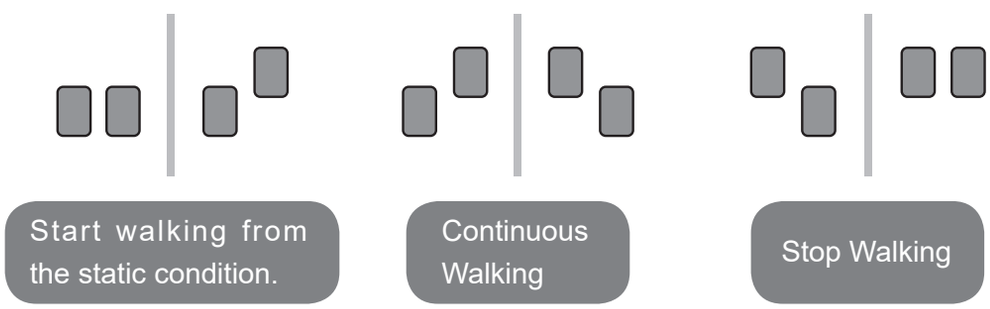
You can give the desired value from 1 to 7 depending on your intention.
It shows the flexibility of the ankle joint when moving and the default value is 5. Because it copied page #31(Walk Ready) to generate the motion current value is set to 6, which is the recommended value for walking. Then you can create much smoother and more stable walking motion. Please refer to the page 149 of this book.

4 - 4 - 3 Practice to generate Walking-Forward motion

Use the Cycle 1 (6 steps) you just created and adjust the joint values appropriately to generate omni-directional walking motion.
This chapter will talk about how to generate Walking-Forward motion which is considered as the most important walking pattern among all kinds of humanoid walking pattern.

Walking-forward motion can be divided into 3 parts - Start Walking, Continuous walking, and Stop walking.

Please refer to the following images for the better explanation.



Start Walking

This is a motion to start walking when walking forward. It would be better if you create a start motion for both feet but it doesn't really matter even if it is just for one foot. Generate 1 cycle of forward walking, another cycle for the other foot with Mirror, then delete the last 3 steps from each foot.

Continuous Walking

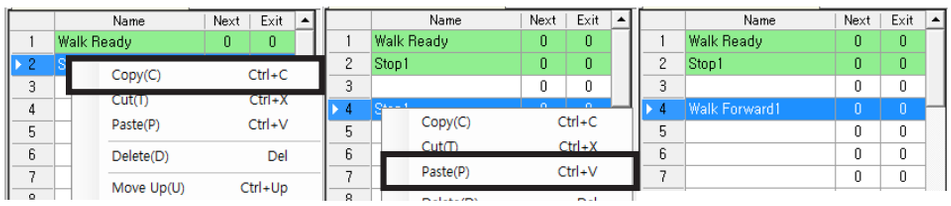
This is where all steps are repeated. You can repeat all 6 steps but the reaction speed will be slower. So give 3 steps for each foot and repeat. Generate 1 Forward-walking cycle and generate for the other foot with Mirror and combine these two appropriately.

Stop Walking

It is required motion for both feet to bring the robot to an initial position after done walking. Generate 1 cycle of Forward-Walking; use the mirror to generate the motion for the other foot and delete the first 3 steps from each foot.

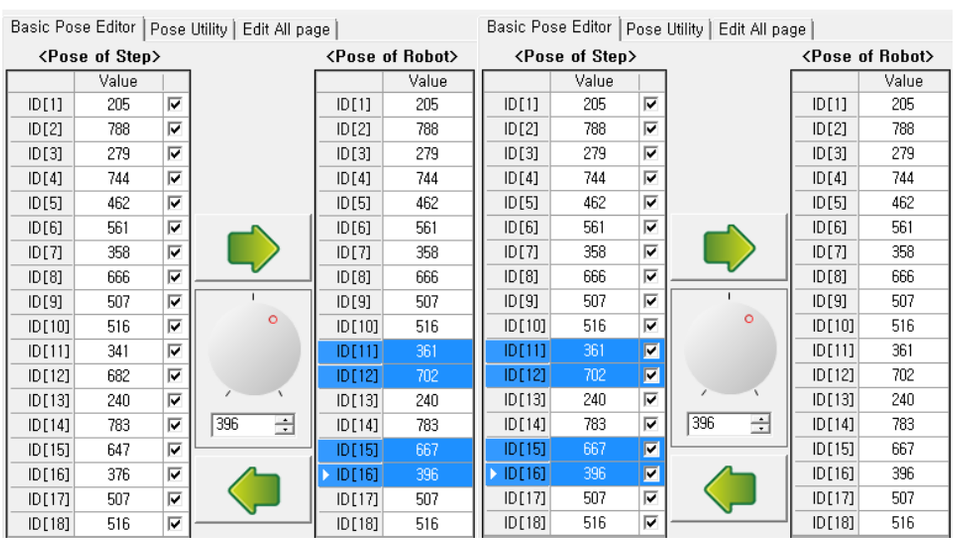
Let's generate "Forward-Walking" pattern using Walking-In-Place Motion.

1. Turn on the robot, connect it to PC and open the Walking-in-Place motion that we just created.
2. On page #4, copy Walking-in-Place motion and save as 'Foward_1'



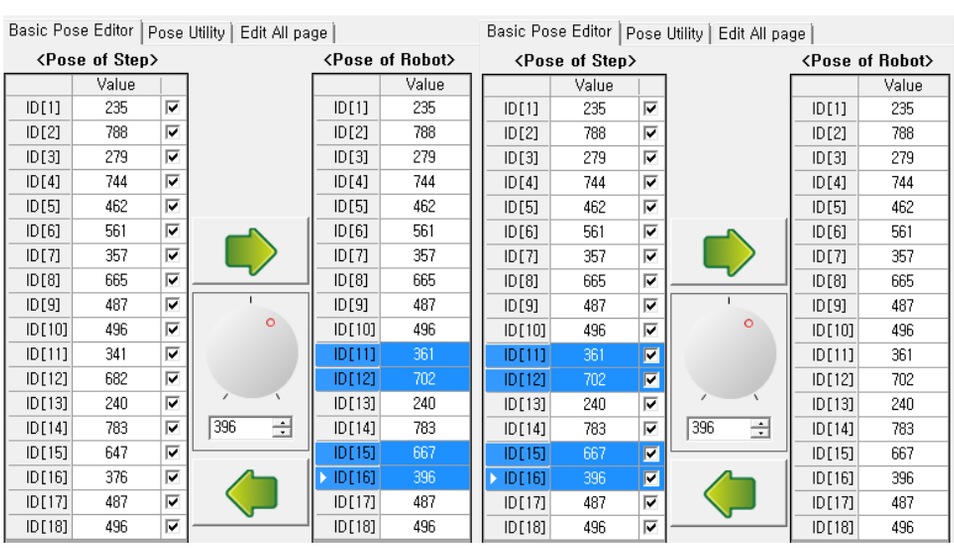
Shortcut Ctrl+C, Ctrl+V

3. Adjust the value of the front/back of hip/ankle joints in step 2.



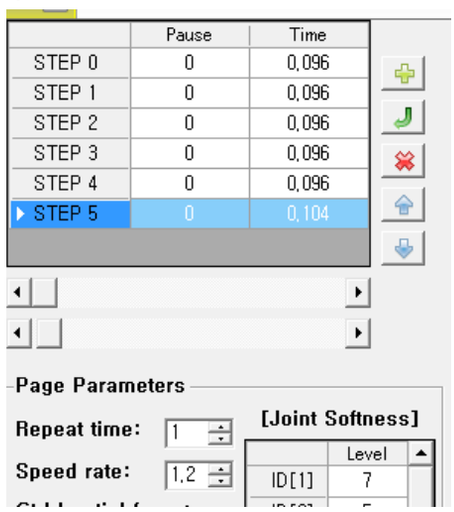
- Move Step 2 to the Pose of Robot
- Press 'Shift +]' twice to increment ID[11], [12], [15], [16] by 20
- Move the Pose of Robot to that of Step 2

4. Adjust the value of the front/back of hip/ankle joints in step 3.

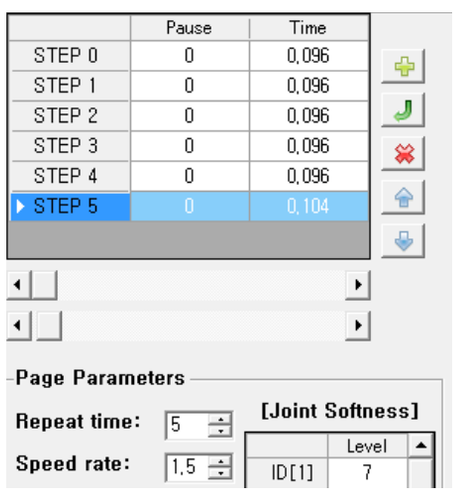


- Move Step 3 to the Pose of Robot
- Press 'Shift +]' twice to increment ID[11], [12], [15], [16] by 20
- Move the Pose of Robot to that of Step 3

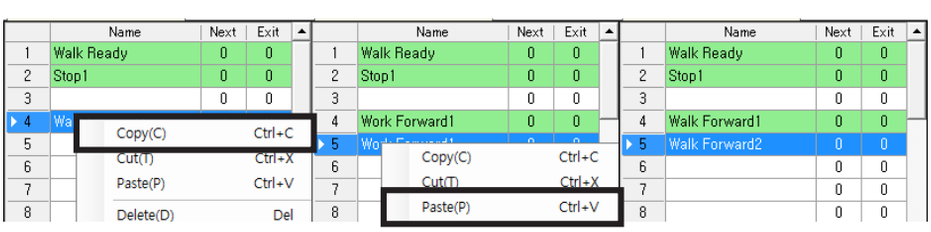
5. Find the most optimized walking motion of 1 cycle by adjusting the speed of the each foot and of the entire step pattern.



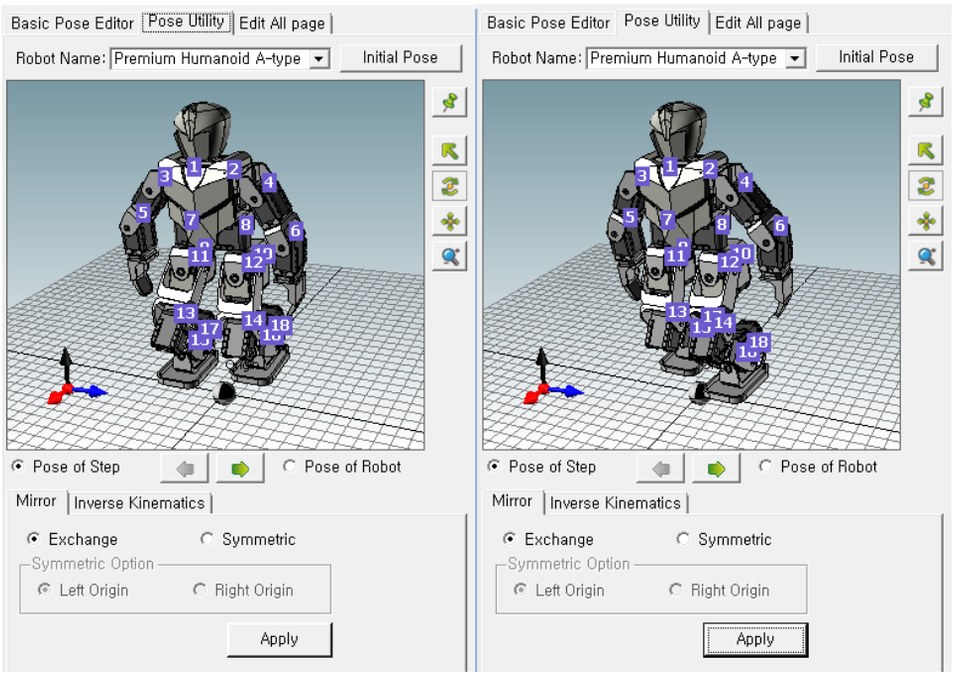
6. Find the most optimized walking motion of continuous walking by adjusting the speed of the each foot and of the entire step pattern.



7. Change the repeat count to 1, copy page #4 to page #5, and save as 'Forward_2'

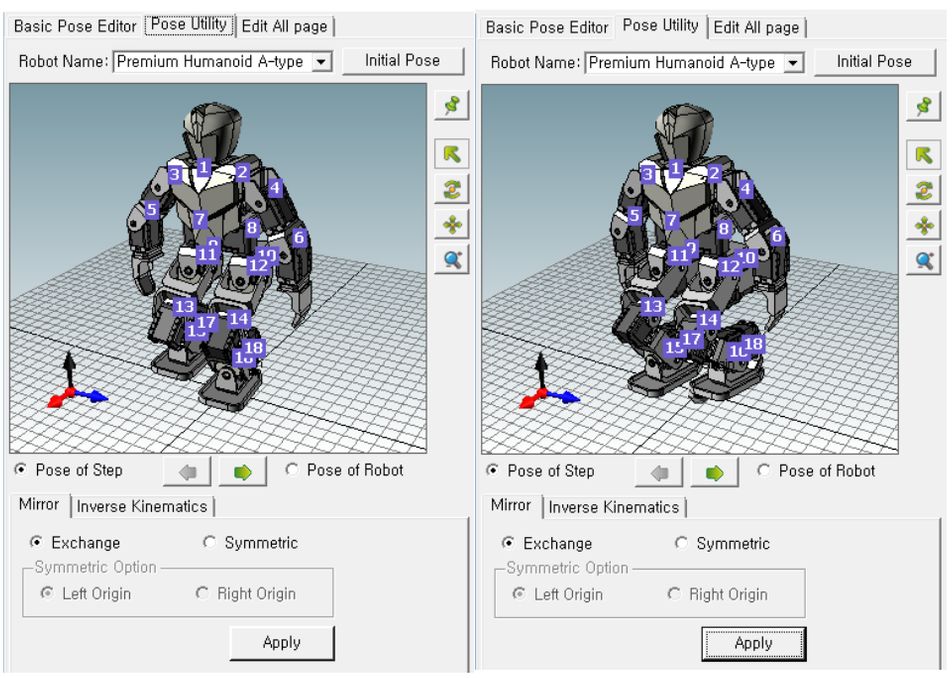


8. Use the mirror function of Pose Utility to flip Step 1~4 of page #5.



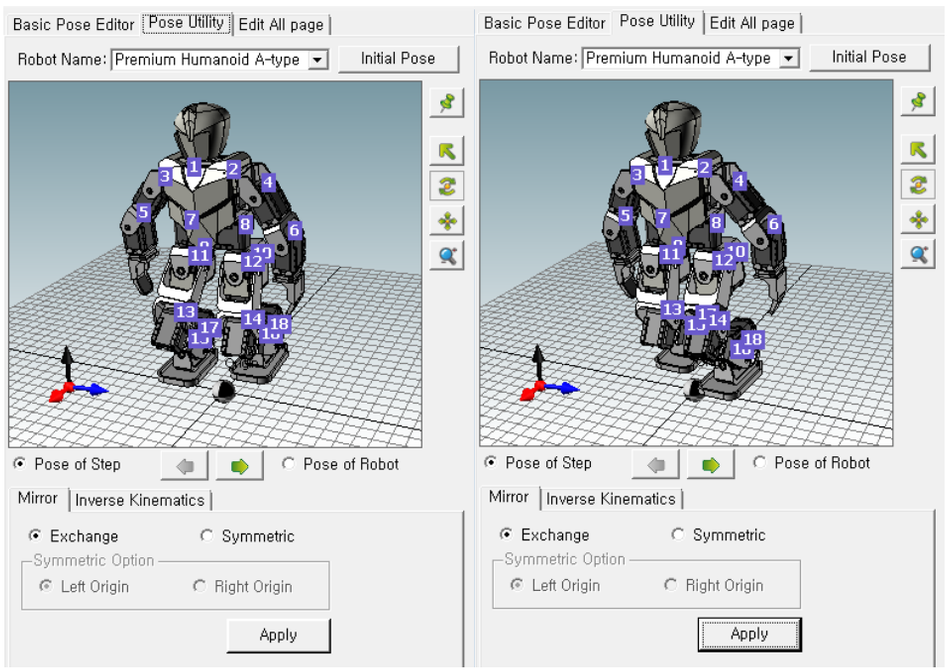
Step 1 - Front

Step 1 - Back



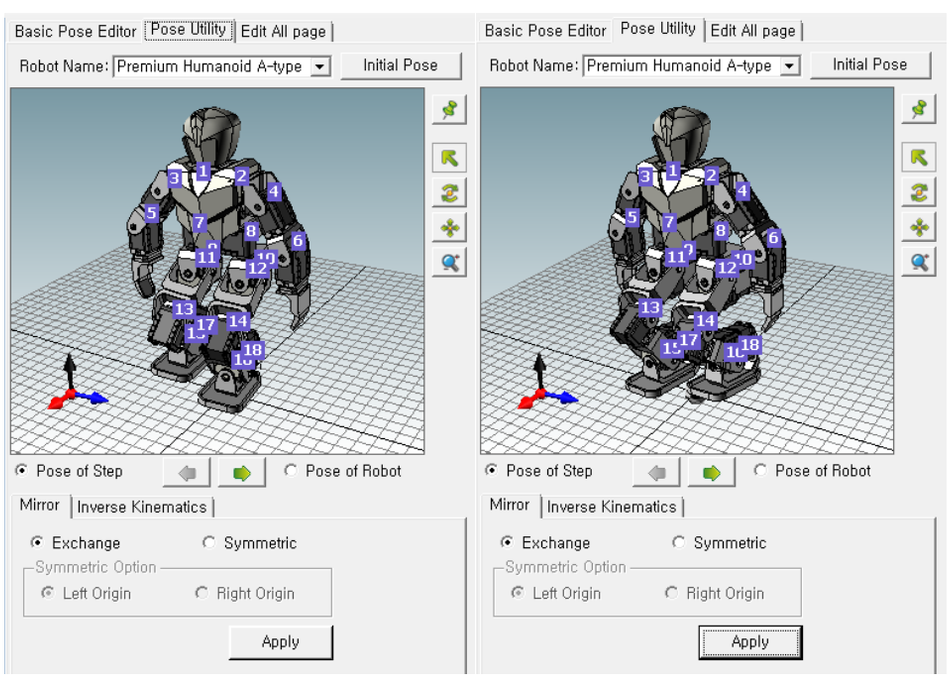
Step 2 - Front

Step 2 - Back



Step 3 - Front

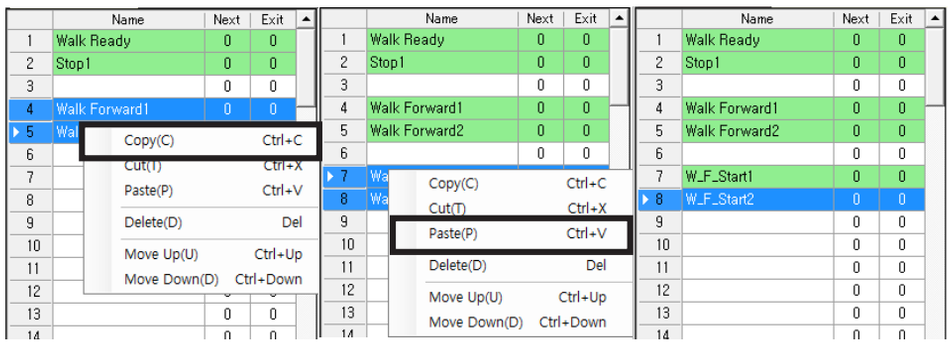
Step 3 - Back



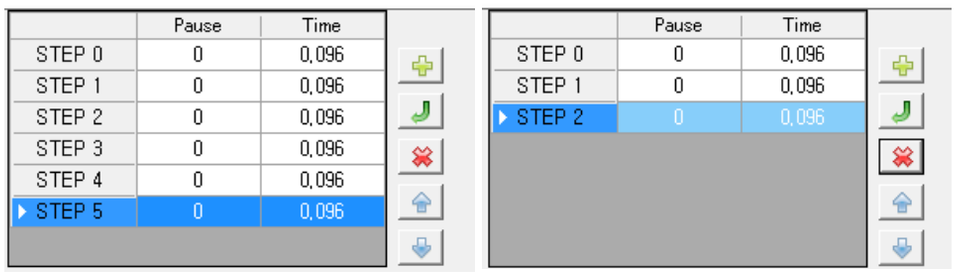
Step 4 - Front

Step 4 - Back

9. Copy Forward_1, Forward_2 to Page 7, 8; Save as Forward_Start_1, Forward_Start_2; and delete Step 3,4,5 from each.

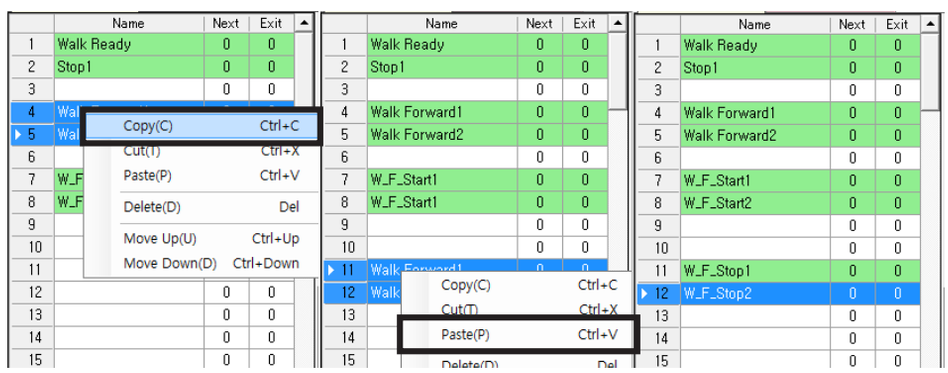


Shortcut Ctrl+C, Ctrl+V

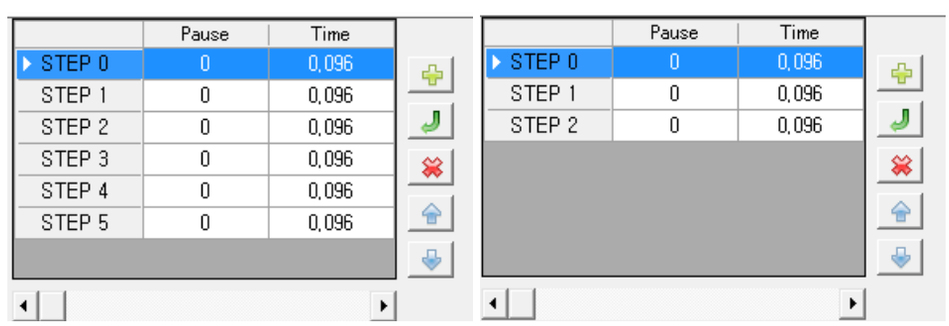


Delete step 3,4,5

10. Copy Forward_1, Forward_2 to Page 11,12; Save as Forward_Stop_1, Forward_Stop_2; and delete Step 0,1,2 from each.

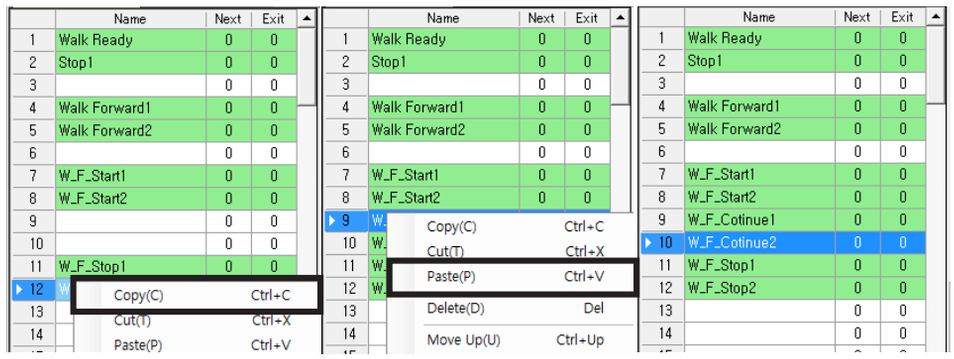


Shortcut Ctrl+C, Ctrl+V



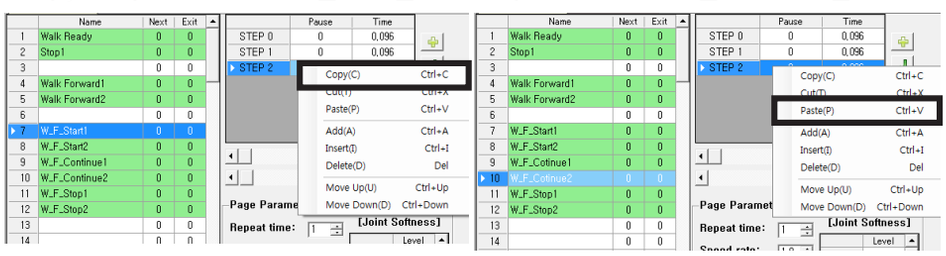
Delete step 0,1,2

11. Copy Forward_Stop 1,2 to Page 9,10; and Save as Forward_ Continue_1, Forward_Continue_2.



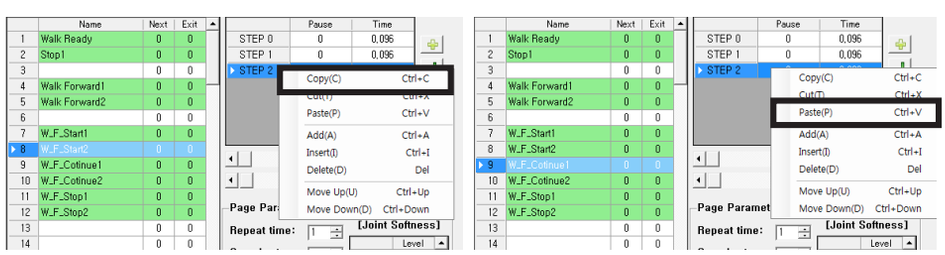
Shortcut Ctrl+C, Ctrl+V

12. Copy Step 2 of Forward_Start_1 to Step 2 of Forward_Continue_2.



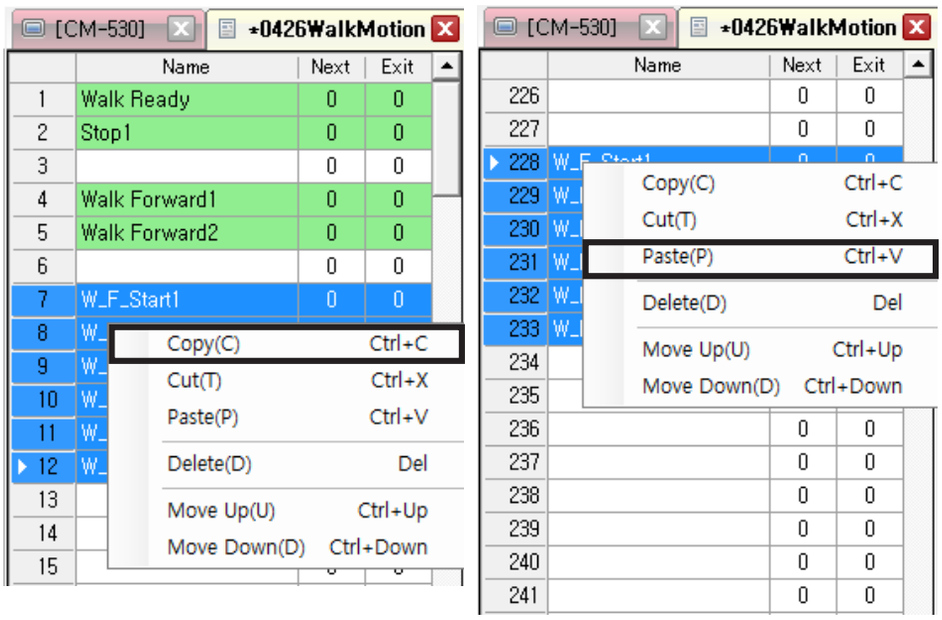
Shortcut Ctrl+C, Ctrl+V

13. Copy Step 2 of Forward_Start_2 to Step 2 of Forward_Continue_1.



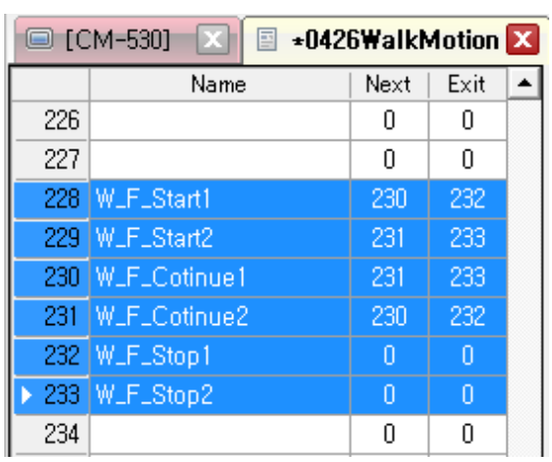
Shortcut Ctrl+C, Ctrl+V

14. Copy Forward_Start_1,2, Forward_Continue_1,2, Forward_Stop_1,2 to Robot Motion.



- Select 6 pages (From page 7~11 from the file motion) and Ctrl+C
- Ctrl+V to page #228 of Robot Motion

15. Assign the appropriate "Next", "Exit" number for each walking pattern and run the final walking test.



- Assign "Next", "Exit" value like in the picture above.
- Execute the motion on page#228 or #229.
- Press Stop button to check page #232, #233.

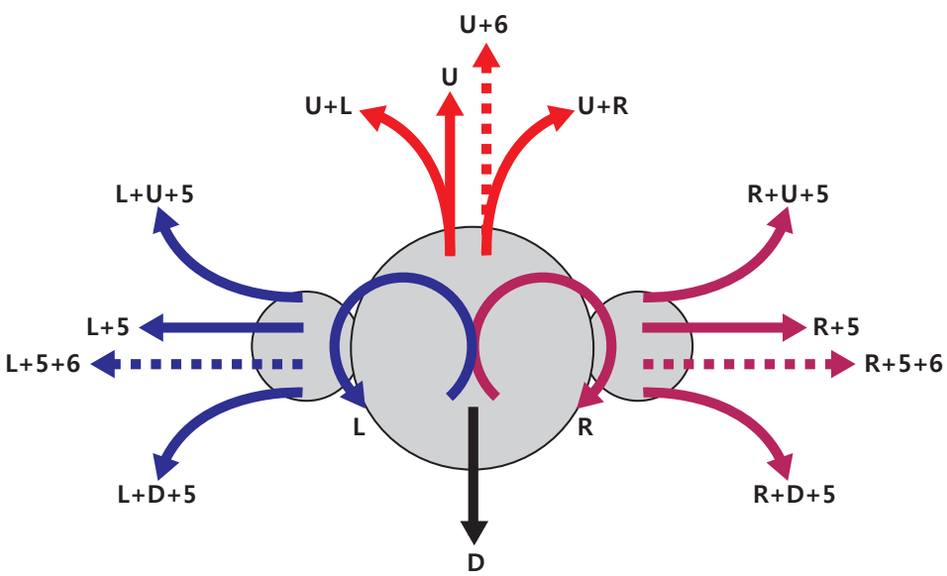
4 - 4 - 4 Advanced Examples of Walking motion

You can find the advanced examples of walking motions for humanoid such as playing soccer or battle from the e-Manual. It is just another walking motion file generated like the above but designed for soccer or battle competition.

Basically, this is an omni-directional, fast-walking pattern with smaller steps for the faster reaction and to minimize the delay while walking while in control by the RC-100B.

You can control the robot to make left/right turn while walking forward, front/back turn while walking sideways as well as the speed of walking forward/sideways.

It is designed to allow the complete control with the RC-100B buttons.



U- Forward	U+L - Turn Left + Forward	U+R - Turn Right + Forward
	U+6 - Forward, Faster	
D - Backward		
L - Turn Left	L+5 - Walk Sideways (Left)	L+5+6 - Walk Sideways (Left, Faster)
	L+U+5 - Forward + Left Diagonally	L+D+5 - Backward + Left Diagonally
R - Right	R+5 - Walk Sideways (Right)	R+5+6 - Walk Sideways (Right, Faster)
	R+U+5 - Forward + Right Diagonally	R+D+5 - Backward + Right Diagonally

Motion Explanation

The walking motion is consisted of page #1~#28.

Page #1 is the initial position - its barycenter is lower than that of basic walking motion. It is designed to fit the rules of various robot walking competitions, which is the angular value of knee joint should be larger than 90 degrees.

	Name	Next	Exit
1	Init	0	0
2	Balance	2	0
3	F_R_S	5	7
4	F_L_S	6	8
5	ff_r_l	6	8
6	ff_l_r	5	7
7	F_R_E	0	0
8	F_L_E	0	0
9	ff_r_l	10	8
10	ff_l_r	9	7
11	FRT_R_M	11	6
12	FLT_L_M	12	5
13	B_R_S	15	17
14	B_L_S	16	18
15	B_R_M	16	18
16	B_L_M	15	17
17	B_R_E	0	0
18	B_L_E	0	0
19	RFT	0	0
20	LFT	0	0
21	RBT	0	0
22	LBT	0	0
23	R	0	0
24	L	0	0
25	Fst_R	0	0
26	Fst_L	0	0
27	RT	0	0
28	LT	0	0

Page #2 is to continuously call the current initial position - is to adjust the posture with the gyro sensor.

Page #3~#12pages are related to moving forward; page #3~#8 are basic walking motions which we just covered, #9~#10 pages are fast-walking, page #11~#12 is to turn left/right as moving forward.

Page #13~#18th pages are to walk in reverse.

Page #19~26 are related to walking-sideways motions; page #23, #24 are basic sideway walking; page #25, #26 are faster sideway walking; page #19, #20 are sideway walking as turning forward; page #21, #22 are sideway walking as turning backward.

Page #27~#28 are turning left/right.

Run all the pages one by one to check the motion.

Task Description

This task is a program to call the necessary motions for the soccer or the battle in accordance with the pressed buttons of the RC-100B.
Because it is more than 400 lines we will explain the certain parts that need detailed explanation.

Look where the program ends (and where functions begin); when no function is called, the Page # calls the 2nd motion. When you press one of the buttons of the RC-100B it stops the motion from page #2 and calls the corresponding motion. When you release the button it stops that motion and no other motion is called, so it goes back to 2nd motion.
Basic turn motion or walking sideways will be repeating the pattern above like other battle motions or soccer related motions.

```
        ELSE IF ( RxData == L )
        {
            UseGyro  = FALSE
            CALL ExitExcuteStop
            Motion Page  = 28
        }
        ELSE IF ( RxData == R )
        {
            UseGyro  = FALSE
            CALL ExitExcuteStop
            Motion Page  = 27
        }
```

...

```
        ELSE IF ( RxData == -- )
        {
            CALL ExitExcuteStop
        }
    }
```

When the robot is walking forward; as you pressed the button to do so, you can change the direction as it moves by pressing an additional button. Once you release that additional button it will just keep moving forward. The task code used here is to move the robot appropriately in a given situation of motion page depending on the pressed button.

That means 'U' button may be used with other buttons; like 'L', 'R,' or '6' (Such as 'U+L','U+R','U+6'). We program this task code to move the robot in the exceptional situations(e.g. turning right/left as moving forward, moving forward as turning left/right, etc) by calling other motion pages when the additional button is pressed/released while executing the motion page.

```
RxData = Remocon RXD
IF ( RxData == U )
{
    UseGyro = TRUE
    IF ( Motion Page >= 3 && Motion Page <= 12 )
    {
        IF ( Motion Page == 9 || Motion Page == 11 )
        {
            CALL ExitWithoutExcuteStop
            Motion Page = 6
        }
        ELSE IF ( Motion Page == 10 || Motion Page == 12 )
        {
            CALL ExitWithoutExcuteStop
            Motion Page = 5
        }
        ELSE IF ( Motion Page == 7 )
        {
            CALL WaitMotion
            Motion Page = 4
        }
        ELSE IF ( Motion Page == 8 )
        {
            CALL WaitMotion
            Motion Page = 3
        }
    }
    ELSE
    {
        CALL ExitExcuteStop
        Motion Page = 3
    }
```

```
    }
    ELSE IF ( RxData == D )
    {
        UseGyro  = TRUE
        IF ( Motion Page >= 13 && Motion Page <= 18 )
        {
            IF ( Motion Page == 17 )
            {
                CALL WaitMotion
                Motion Page  = 14
            }
            ELSE IF ( Motion Page == 18 )
            {
                CALL WaitMotion
                Motion Page  = 13
            }
        }
        ELSE
        {
            CALL ExitExcuteStop
            Motion Page  = 13
        }
    }
```

If the Task code is too complex to understand copy the corresponding walking pattern to the motion page while keeping next/exit the pages is one way to facilitate understanding.

Link with Motion Task

Read the code line by line and see how it works with the motion page. This can help understand the robot system and structure, add/create motions, walk patterns, and tasks; eventually, solve complex missions.

4 - 4 - 5 Walking Machine

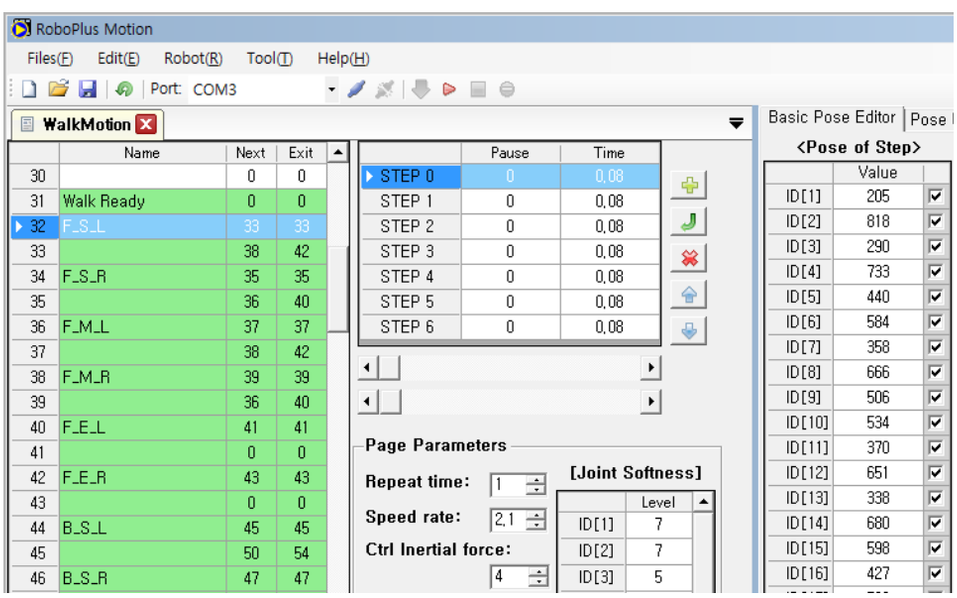
[Learning Objective]

<Waking Machine> is a combination of a walking motion file made up of specific patterns and a task code that plays the role of smoothly connecting the walking motions in this motion file.
Let's use the <Walking Machine> to learn about how walking motions are converted smoothly.

[STEP 1]

Walking Motion File Overview.

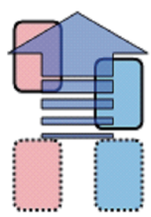
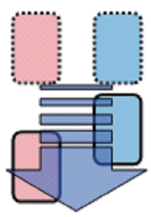
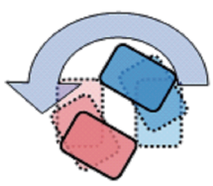
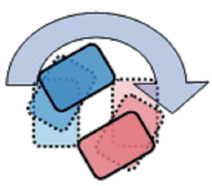
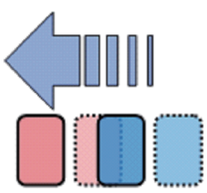
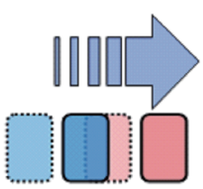
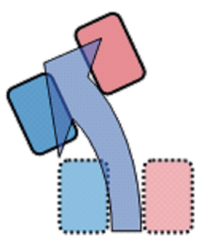
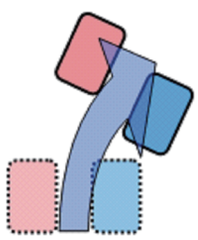
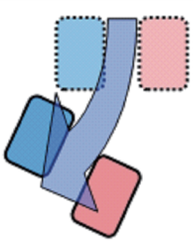
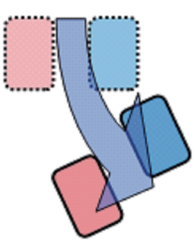
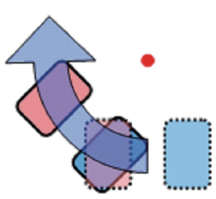
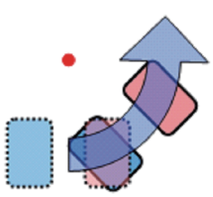
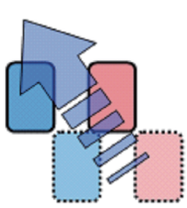
Walking Motion File



The walking motion files used in the Walking Machine can be found on pages 31~224. Several motion pages are gathered and repeatedly played to make up one walking pattern (forward, backward, etc). Each page is made to be conveniently converted to the next walking pattern's motion page.

The walking patterns written in the Walking Motion File

The walking motion files used in the Walking Machine contain 16 different walking patterns shown below.

Forward	Backward	Pivot Left	Pivot Right
			
Walk Sideways + Left	**Walk Sideways + Right**	**Turn Left + Forward**	**Turn Right + Forward**
			
Turn Left + Backward	**Turn Right + Backward**	**Avoid Left**	**Avoid Right**
			
Forward + Left Diagonally	**Forward + Right Diagonally**	**Backward + Left Diagonally**	**Backward + Right Diagonally**
			

[STEP 2]

Walking Machine Task Code Overview

The walking machine task code includes a "Initialization Walk" Function and a "Walk Execute" function. There are examples that use these 2 functions to control the robot with a remote controller.

[Initialization Walk] Function

The "Initialization Walk" function initializes the variables and brings the robot to its default position.

```
620 FUNCTION InitializationWalk
621 {
622     BalancePage  = 224
623     WalkPageStart  = 31
624     WalkCommand  = 0
625     Motion Page  = WalkPageStart
626     CALL WaitMotion
627     WalkState  = 0
628 }
```

[Walk Execute] Function

The "Walk Execute" function executes the walking pattern depending on the corresponding command and smoothly links the conversion between each walking pattern.

```
620 FUNCTION InitializationWalk
621 {
622     BalancePage  = 224
623     WalkPageStart  = 31
624     WalkCommand  = 0
625     Motion Page  = WalkPageStart
626     CALL WaitMotion
627     WalkState  = 0
628 }
```

[Walk Execute] Function

The "Walk Execute" function executes the walking pattern depending on the corresponding command and smoothly links the conversion between each walking pattern.

```
630 FUNCTION WalkExecute
631 {
632     // Using WalkControl Variable
633     //  0 : Stop,       1 : Forward,     2 : Backward,     3 : Turn Left,  4 = Turn Right,
634     //  5 : Left Side,  6 : Right Side,  7 : 1 + 5,        8 : 1 + 6,      9 : 2 + 5,
635     // 10 : 2 + 6,     11 : Avoid Left, 12 : Avoid Right, 13 : 1 +3,      14 : 1+4,
636     // 15 = 2+3,       16 = 2+4
637
638
639     IF ( WalkCommand == WalkState )
640         RETURN
641
642
643     Temp1  = 196 + WalkPageStart
644     IF ( Motion Page < WalkPageStart || Motion Page > Temp1 )
645     {
646         CALL EXITPageWaitMotion
647         IF ( Motion Page != BalancePage )
648         {
```

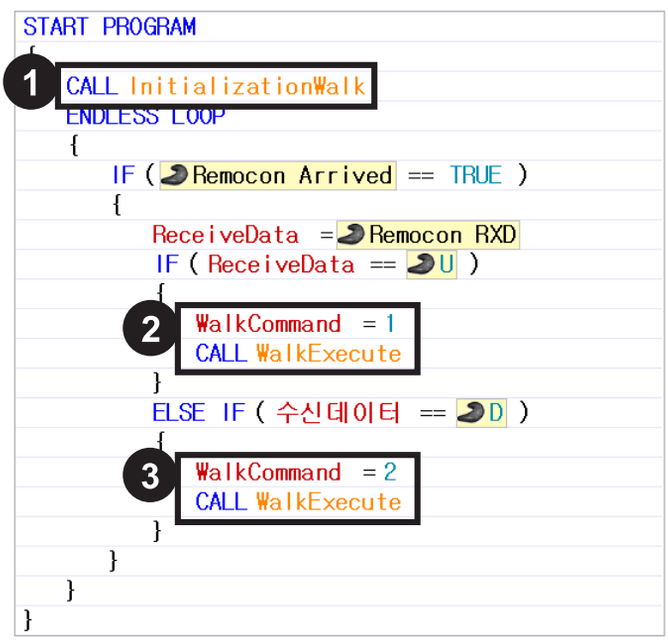
Double click to edit

Walk Command No.

no	'Walking' Pattern	no	'Walking' Pattern	no	'Walking' Pattern	no	'Walking' Pattern
1	Forward	2	Backward	3	Pivot Left	4	Pivot Right
5	Walk Sideways + Left	6	Walk Sideways + Right	7	Turn Left + Forward	8	Turn Right + Forward
9	Turn Left + Backward	10	Turn Right + Backward	11	Avoid Left	12	Avoid Right
13	Forward + Left Diagonally	14	Forward + Right Diagonally	15	Backward + Left Diagonally	16	Backward + Right Diagonally

Start Program

Let's try writing a simple "Start Program" sample using the "Initialization Walk" function and "Walk Execute" functions to smoothly connect walking patterns.



1. First, call the "Initialization Walk" function to initialize the variable and to bring the robot to its default position.
2. Select and input a walking pattern between 1 and 16 for the "Walk Command" variable; then, call the "Walk Execute" function to run the selected walking pattern.
3. When you input a different number in the "Walk Command" variable and call the "Walk Execute" function it will convert to the new walking pattern as naturally as possible.

There are 16 different walking pattern sample codes to control via remote control in the walking machine's task code file. Change the "Start Program" function in the sample codes to suit your needs.

4 - 4 - 6 Adjustment Using the Gyro Sensor

[Learning Objective]

Learn how to adjust a humanoid's posture using a gyro sensor.
A gyro sensor is used to determine angular velocity (angular variation per second). When the angular velocity increases in a specific direction as the robot tilts; the servo motor's value can be adjusted in the opposite direction to straighten the robot.

Things to Prepare

The gyro sensor's X-axis value should be connected to port #3, and Y-axis value should be connected to port #4.
Otherwise, modify the task code to suit your robot.

[STEP 1]

Concept

Callback function

A humanoid with a gyro uses the "callback function" to determine the posture adjustment value. The callback function is a function that runs independently of the main program routine and is automatically executed at fixed intervals. Therefore, by calculating the adjustment value and using the value in a callback function to adjust the posture at regular intervals; the robot can adjust its posture automatically.

Joint Offset

The joint off set is added to the adjustment value from the gyro sensor before being used to adjust the humanoid's posture (More information on Joint Offset). This is a parameter that gives an offset to the specific actuator's joint position value. Thus, the actuator with joint offset executes their motions using the position value + joint offset value, which are designated in the motion file.

[STEP 2]

Task Code Overview.

1. <InitializationGyro> Call Function

```
1  // Bioloid PremiumKit Tutorial(Gyro Revision)
2  START PROGRAM
3  {
4      CALL InitializationGyro
5      CALL InitializationWalk
6
7      ENDLESS LOOP
8      {
9          IF ( Remocon Arrived == TRUE )
10         {
```

2. Execute “Blance/Page” to apply Joint Offset; execute a motion to apply Joint Offset. Therefore, a BalancePage is required to apply the joint offset when no motion is being executed.

```
112
113         IF ( Motion Status == FALSE )
114         {
115             Motion Page = BalancePage
116         }
117
```

3. <InitializationGyro> Function

The "InitializeGyro" function reads the gyro sensor value 10 times in a 0.128 second intervals and saves the average value as a gyro sensor standard value (The standard value is approximate 250 degrees).

If the gyro sensor value is less than 230 or greater than 270 the function assumes there is no gyro sensor; the function does not use adjustment (when there is no gyro sensor or the robot has moved during initialization).

```
342 FUNCTION InitializationGyro
343 {
344     FBBalCenter = 0
345     RLBalCenter = 0
346
347     LOOP FOR ( i = 1 ~ 10 )
348     {
349         FBBalData = PORT[3]
350         RLBalData = PORT[4]
351
352         FBBalCenter = FBBalCenter + FBBalData
353         RLBalCenter = RLBalCenter + RLBalData
354
355         Timer = 0.128sec
356         WAIT WHILE ( Timer > 0.000sec )
357     }
358
359     FBBalCenter = FBBalCenter / 10
360     RLBalCenter = RLBalCenter / 10
361                                   Double click to edit
362     ExistGyro = TRUE
363     IF ( FBBalCenter > 270 || FBBalCenter < 230 )
364         ExistGyro = FALSE
365     IF ( RLBalCenter > 270 || RLBalCenter < 230 )
366         ExistGyro = FALSE
367
368     UseGyro = TRUE
369 }
```

4. The callback function reads the current gyro sensor value and compares it to the standard value to calculate the adjustment value.

If the "UseGyro" variable is false the robot it will not adjust itself. Therefore, set the “UseGyro” variable to false where you do not wish to use the gyro adjustment.

```
371 CALLBACK
372 {
373     IF ( GyroUse == TRUE && ExistGyro == TRUE )
374     {
375         FBBalData  = PORT[3]
376         RLBalData  = PORT[4]
377
378         FBBalError  = FBBalData - FBBalCenter
379         RLBalError  = RLBalData - RLBalCenter
380
381         FBBalErrorScaled  = FBBalError * 4
382         RLBalErrorScaled  = RLBalError * 4
383
384         FinalFBBal1  = FBBalErrorScaled / 54
385         FinalFBBal2  = FBBalErrorScaled / 18
386
387         FinalRLBal0  = RLBalErrorScaled / 16
388         FinalRLBal1  = RLBalErrorScaled / 32
```

5. Apply the calculated adjustment value to the joint offset and adjust the robot's posture.

   To adjust the front/back tilt, you must use the joints in the knees and ankle, which are actuators 13-16. To adjust the left/right tilt, you must use the joints in the ankle and waist, which are actuators 9-10 and 17-18.

```
390         ID[13]:Joint offset  = 0 + FinalFBBal1
391         ID[15]:Joint offset  = 0 + FinalFBBal2
392         ID[14]:Joint offset  = 0 - FinalFBBal1
393         ID[16]:Joint offset  = 0 - FinalFBBal2
394
395         ID[9]:Joint offset  = 0 + FinalRLBal1
396         ID[10]:Joint offset  = 0 + FinalRLBal1
397         ID[17]:Joint offset  = 0 - FinalRLBal0
398         ID[18]:Joint offset  = 0 - FinalRLBal0
```

[STEP 3]

Adjustment Test

- Download the task code to your robot.
- In order to initialize the gyro you must leave your robot on flat ground for at least 1.5 seconds after executing the task code.
- If the gyro sensor is not connected or if there is movement while initializing the gyro sensor there will be no adjustments.
- Observe the robot adjust its posture under the influence of external forces.
- Compare the robot's action when it does and does not make use of the gyro sensor while standing.

4 - 4 - 7 User-defined Motions — Create Motion

[Learning Objective]

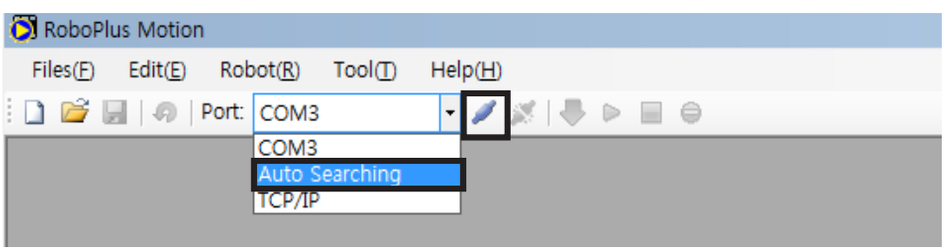
Let's learn how to add the following 4 motions using the RoboPlus Motion program.

Page No.	Movement Descriptions	Page No.	Movement Descriptions
14	Block ball on the right	27	Get up while lying on stomach
16	Block ball on the left	28	Get up while lying on back

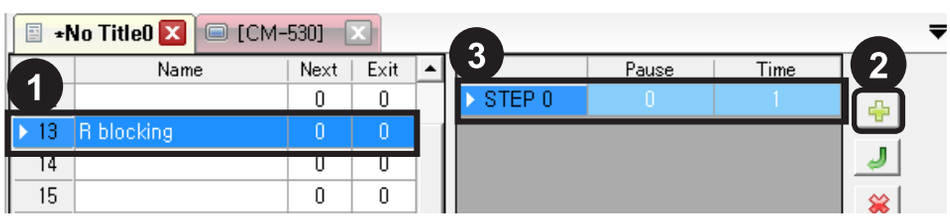
[STEP 1]

Make a motion to block the ball on the right.

1.Execute RoboPlus Motion, and connect it to the CM-530

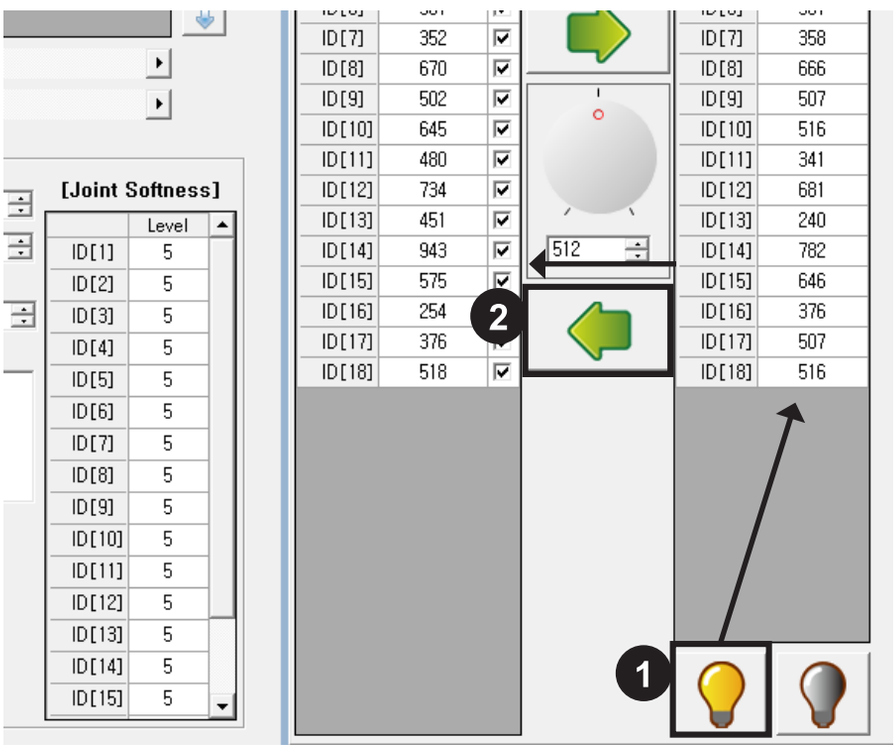


2. To add the "Block Ball" motion in page 14 input a name and step.

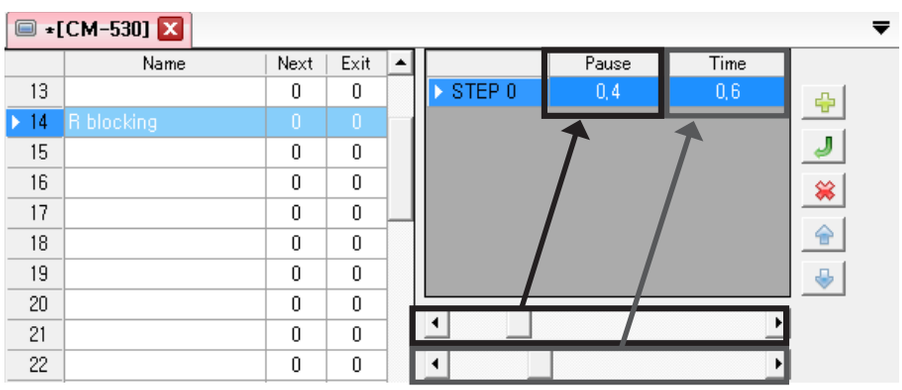


3. Turn off all motors then set the robot's pose. The "Block ball on the right" pose is shown below.

4. Set the robot's pose as above. When you press the "torque on" button the robot's current actuator values will automatically be saved. When you press the left arrow button you can read the robot's current input actuator values into Step's Pose column.

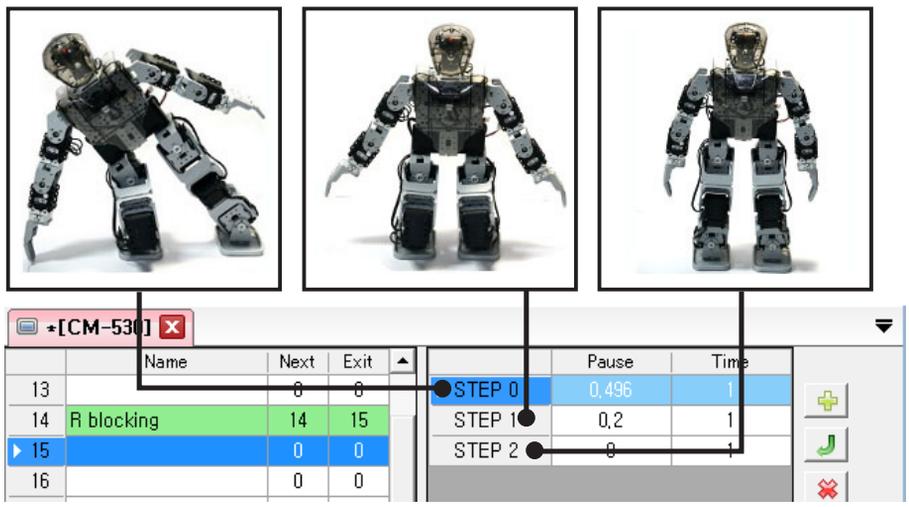


5. You can adjust the pause and play durations.

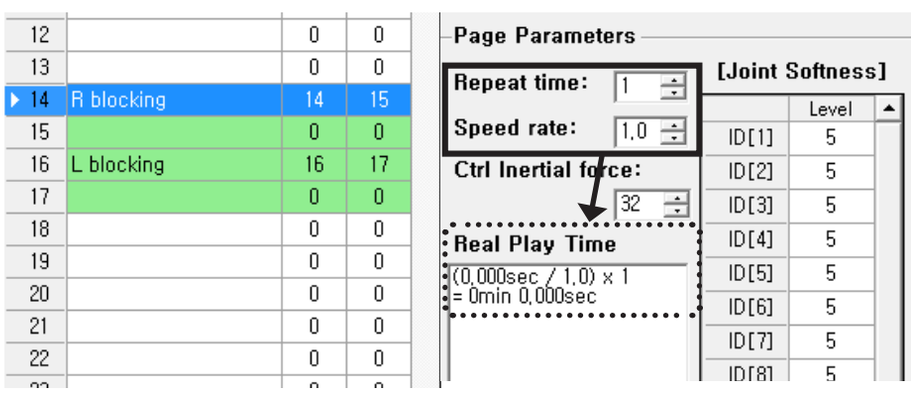


6. Assign the Next page as it (14) will cause the robot to enter an infinite loop and maintain its pose. Also, designate an EXIT page for the robot to smoothly transition into when it exits the infinite loop.

7. Make a motion to return to the standard position on the page designated as the Exit page (15). Add the following 3 steps and poses to page 15.



8. Additional settings (Number of Repeats, Play Speed, etc...). You can also set the number of repeats, entire speed, etc. for each page.



[STEP 2]

Make a motion to block the ball on the left.

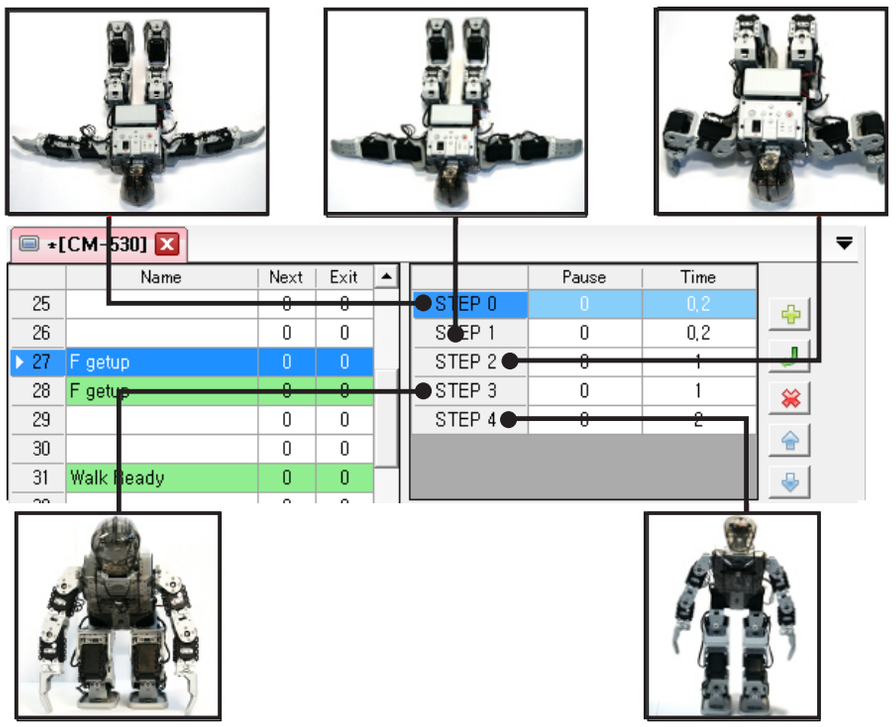
Repeat STEP 1 to make a motion to block a ball on the left on pages16 and 17.

[STEP 3]

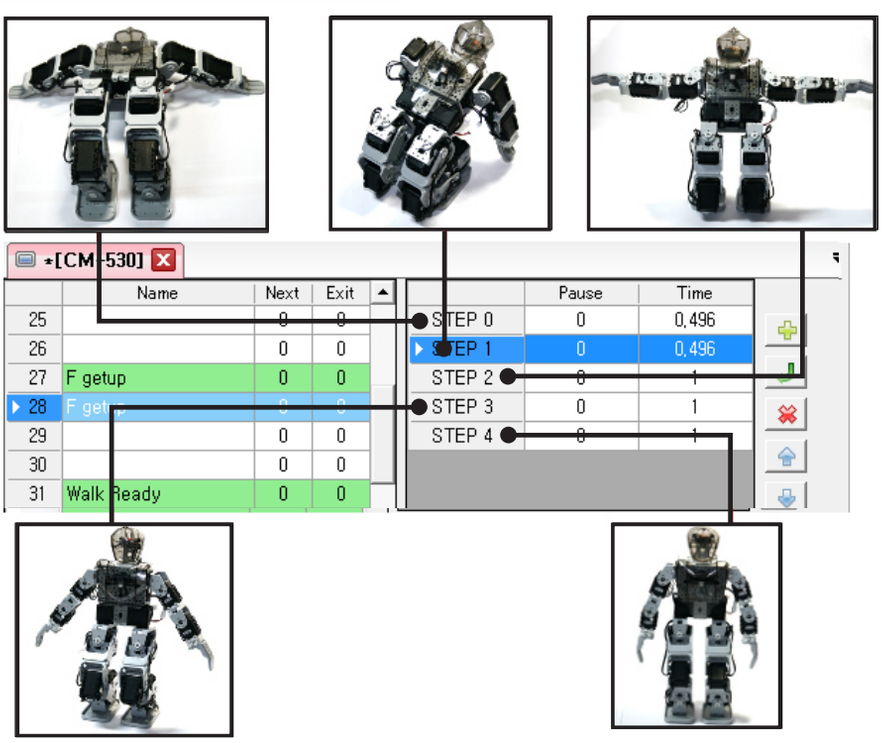
Create motions to get up when the robot is lying on its back and chest.

Add the following steps in motion page 27 and 28 to make a "Get up while lying on stomach" and "Get up while lying on back" motions.

Get up while lying on stomach



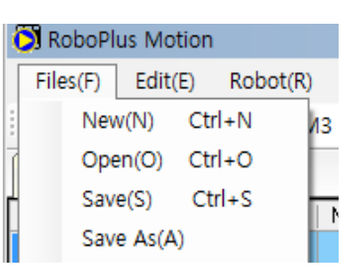
Get up while lying on back



[STEP 2]

Save

Use the "Save (S)" command to save your work As (A)" to save as an .mtn file in your PC.



4 - 4 - 7 User defined Motions Task Code

[Learning Objective]

Write task code to execute user-defined motions.
Let's learn how to run the motion added on "User-defined Motions 1 : Create Motion" with RC-100B.

[STEP 1]

Task Code Overview.

Code to execute user-defined motions has been added to the task code written in "Adjusting using the Gyro Sensor."

```
ELSE IF ( RxData == -- )
{
    Double click to edit  = 0
    CALL WalkExecute
}
ELSE
{
1   WalkCommand  = 0
    CALL WalkExecute
}
2 GyroUse  = FALSE
  CALL EXITPageWaitMotion

3 IF ( RxData == U+2 )
  {
      Motion Page  = 27
      CALL WaitMotion
  }
  ELSE IF ( RxData == D+2 )
  {
      Motion Page  = 28
      CALL WaitMotion
  }
4 ELSE IF ( RxData == U+3 )
  {
      Motion Page  = 14
      WAIT WHILE ( RxData == Remocon RXD )
      CALL EXITPageWaitMotion
  }
  ELSE IF ( RxData == D+3 )
  {
      Motion Page  = 16
      WAIT WHILE ( RxData == Remocon RXD )
      CALL EXITPageWaitMotion
  }
5 GyroUse  = FALSE
}
```

1. Set "WalkCommand" as 0 to make the robot stop.
2. If the motion added by the user does not require the gyro sensor to maintain the posture you must turn off the gyro sensor adjustment to prevent motion variation due to the offset. Set the "UseGyro" variable to FALSE. Then call the "ExitPageWaitMotion" function and wait for the robot to come to a complete stop.
3. The “Getting Up” motions added in motion pages 27 and 28 can be played just once.
4. The “Block Ball” motions added in motion page 14 and 16 are endlessly repeating motions; to end the motion, an EXIT page is needed. Using the "WAIT WHILE" command so that if the button is not pressed and held, the "ExitPageWaitMotion" function will execute the EXIT to end the motion.
5. After the user's motion ends, set the "UseGyro" variable back to TRUE to restore gyro adjustment.

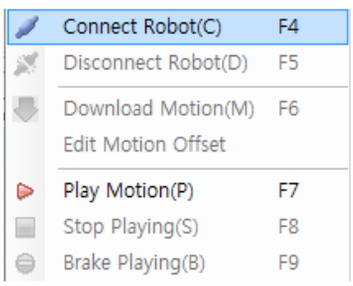
4 - 5 Other information

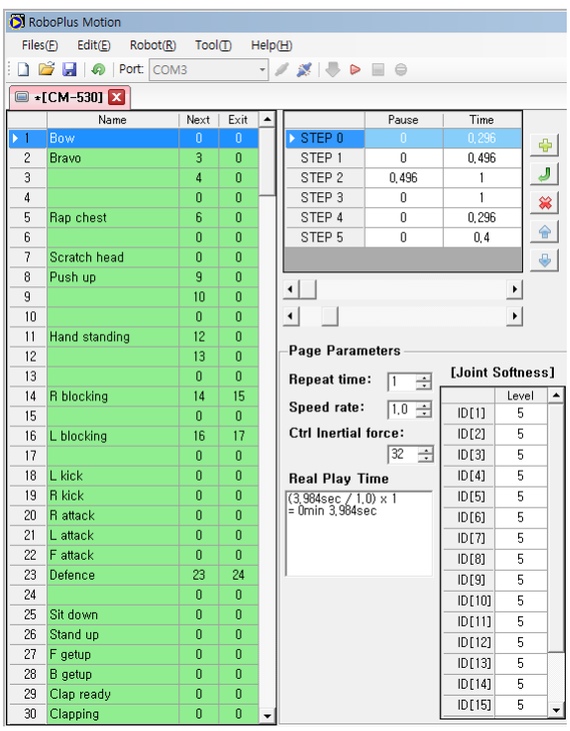
Upload Robot Motion

Transferring motion data from the controller to the PC is called "uploading."

[STEP 1]

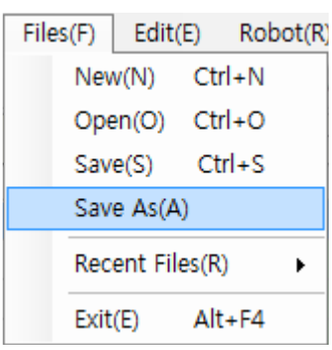
Connect the robot to the PC to see the Robot Motion window.





[STEP 2]

After selecting the Robot Motion window click on "Save As."

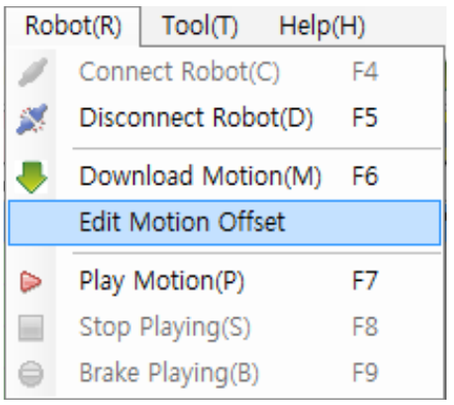


Motion Offset

Offset is the difference from the standard value. Motion offset refers to the difference from the standard motion and the robot that performs the standard motion is called "Master Robot." Even when robots of the same type are performing the same motions there will be differences in their poses. This is due to the discrepancies in motor locations and errors in assembly. These differences may even cause some robots to fall down. "Motion Offset" is used to resolve these differences.

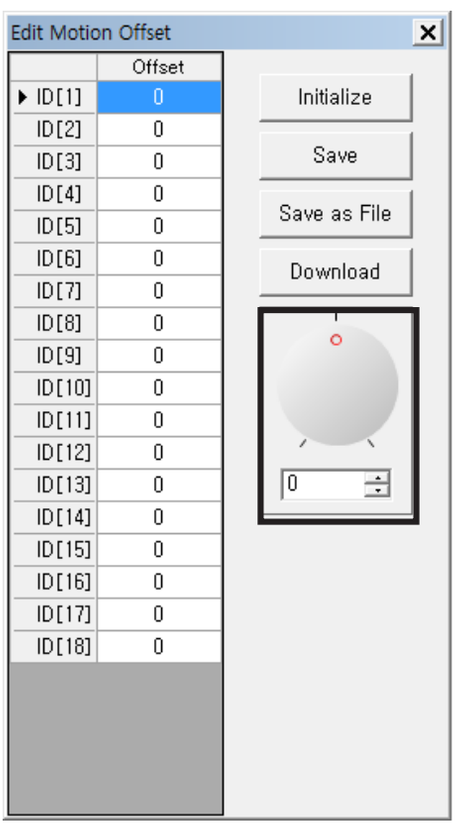
Edit Motion Offset

Discrepancies in the location of robot joints can be fixed using the "Edit Motion Offset" function.



Select the joint to edit its value with the editor.

- Positive values indicate movement in the CCW direction.
- Negative values indicate movement in the CW direction.

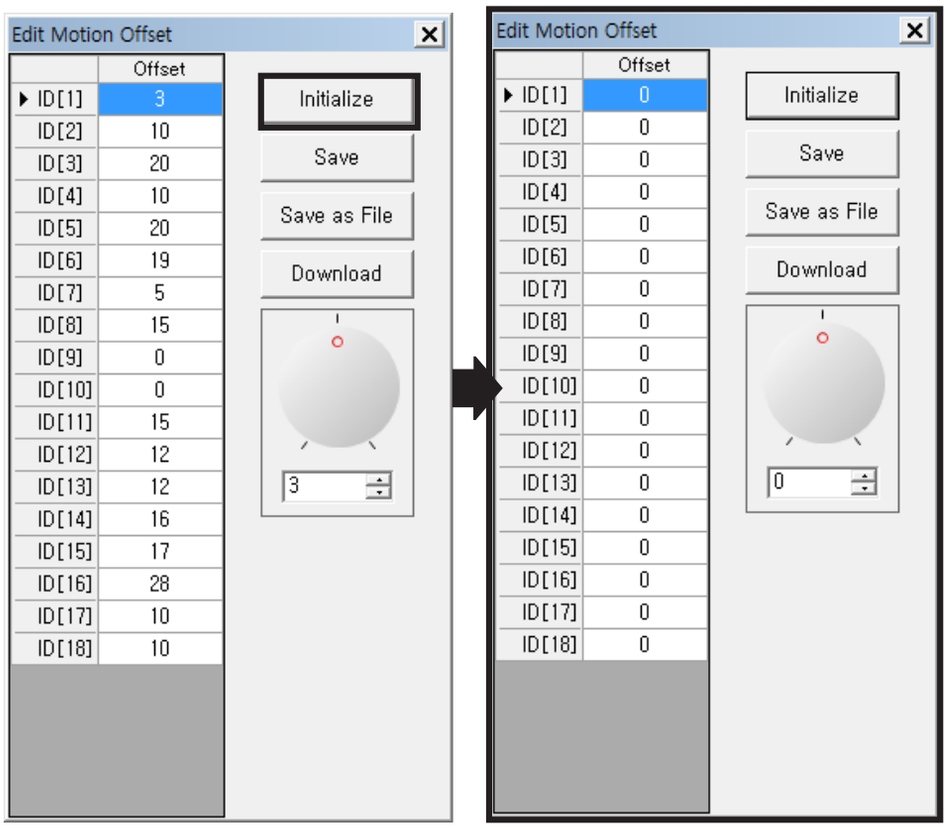


Precaution

Usually, the motion offset is a very small value that you can ignore. However, for the robot like Humanoid, which is difficult to keep the balance, this small value can cause the big problem. When the motion offset menu is selected the torque of all joints will be turned on to sustain its current position. Therefore, it would be beneficial to execute this function when the robot is in a pose where the differences can be easily distinguished.

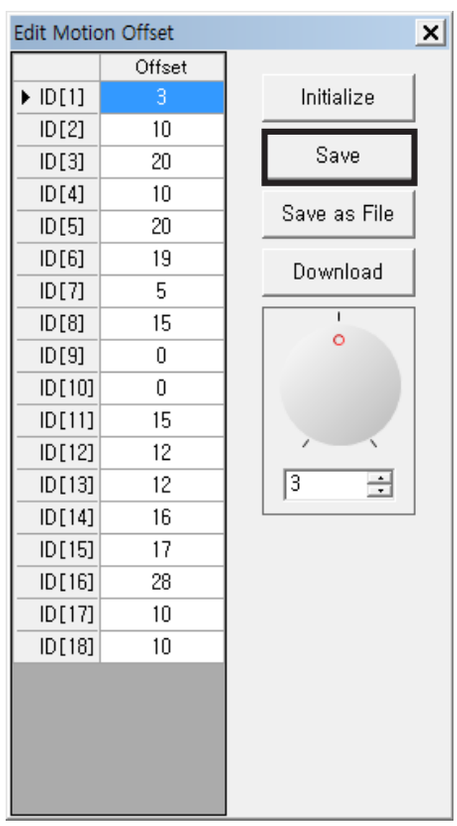
Initialize Motion offset

Initializes all motion offset values to 0.



Save Motion Offset

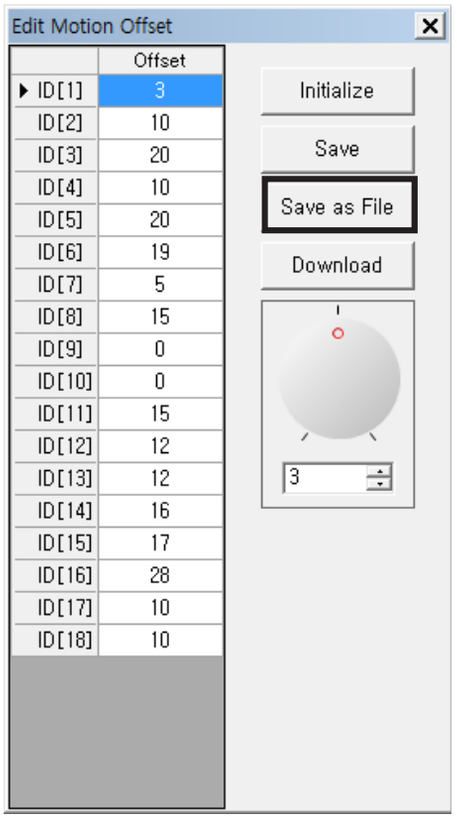
Saves the current offset values. This function saves the values in the controller.



Save as Motion offset File

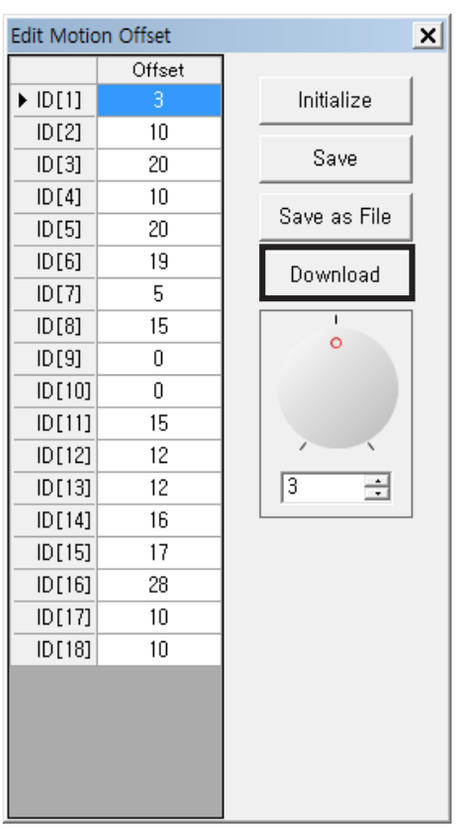
Saves the robot's current offset values as a file in the PC.

The file extension of motion offset files is .ofs.



Download Motion Offset

Motion Offset files(*.ofs) in the PC can be downloaded to the robot.



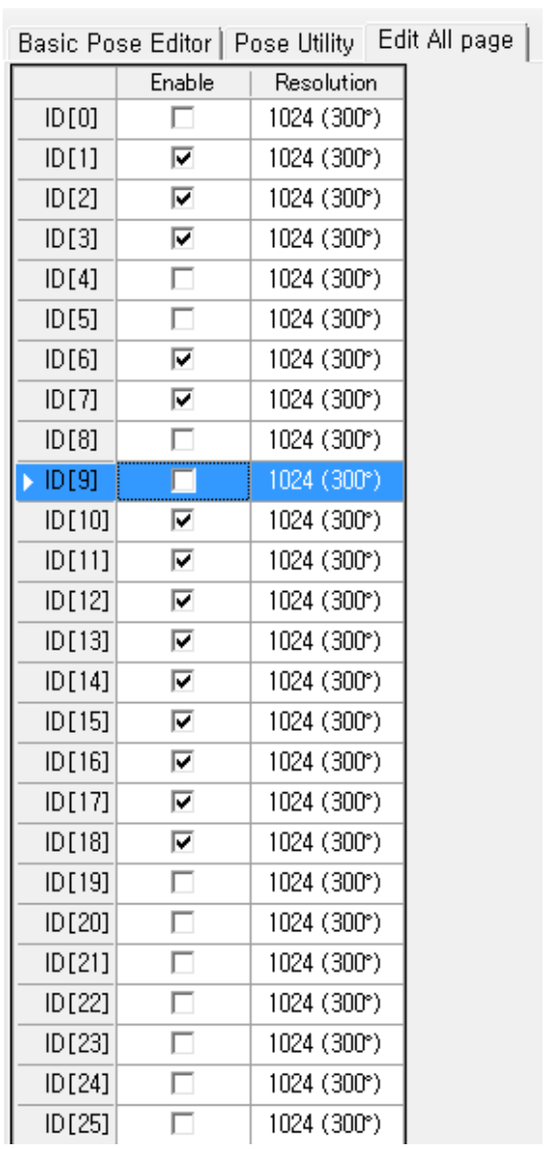
Edit All Page

"Edit All Page" is used to duplicate revisions on all pages.
This function is required in the following situations

- To change all motor values simultaneously
- To change ID usage status (whether it is being used or not)

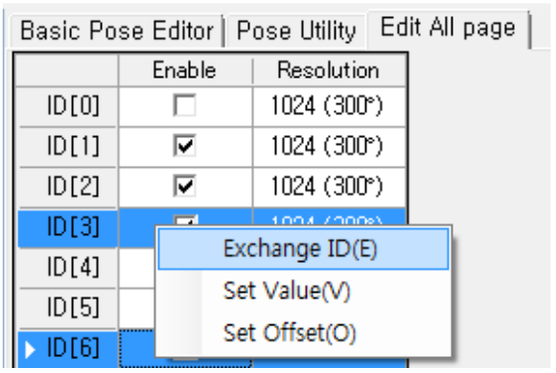
Set ID Usage Status

RoboPlus Motion can handle the motions of robots with up to 26 AX-12A (ID between 0 and 25). Set whether an ID is being used to edit only the necessary ID's.



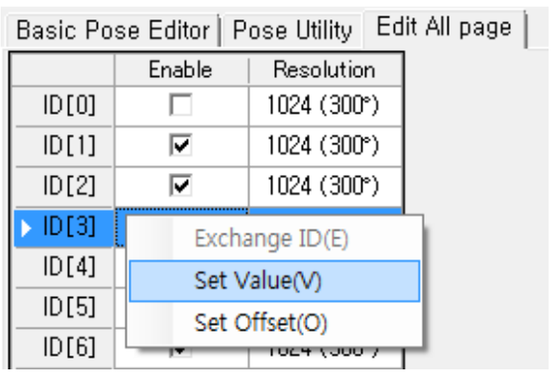
Exchange ID

The position values of the robot's joints can be easily exchanged.
Select the 2 ID's to exchange and then click "Exchange ID."



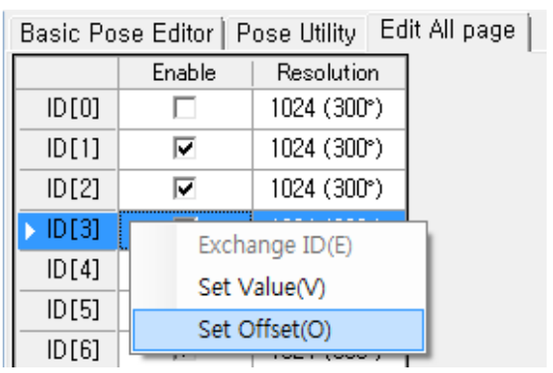
Change All Values

Use this function to change the value of the selected ID.



Apply All Values

Offset is the difference from a standard value. Use this function to add or subtract a value from all joints with the selected ID.

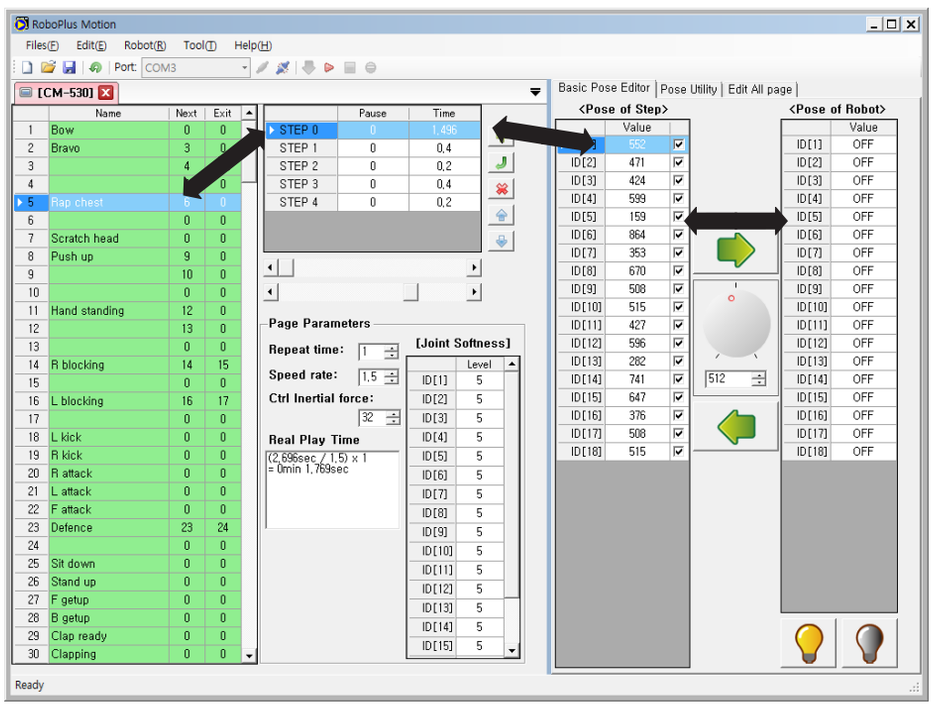


Keyboard Shortcuts

Creating motions via mouse and keyboard can be challenging and tedious while one hand is occupied holding the robot. We introduce some tips to facilitate motion creation.

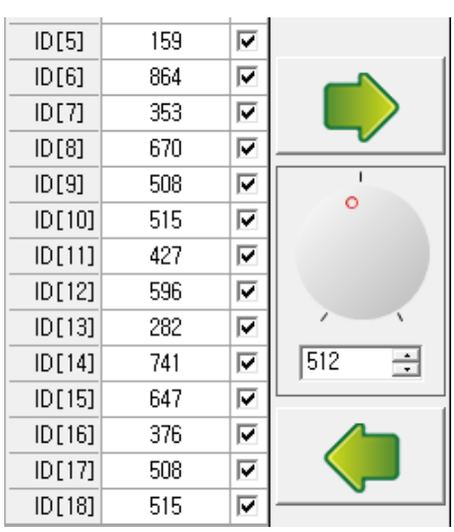
Use arrow keys to move within the program

- Arrow keys can be used to move the focus between the Page Edit Window, Step Edit Window, and Pose Edit Window.



Change the Joint Values

- Press the [and] keys to increase or decrease the joint value by 1.
- Press the { and } keys (Shift + [,]) to increase or decrease the joint value by 10.
- Press Enter to move the focus to the setting window. When you are done changing the value press Enter again to return the focus.

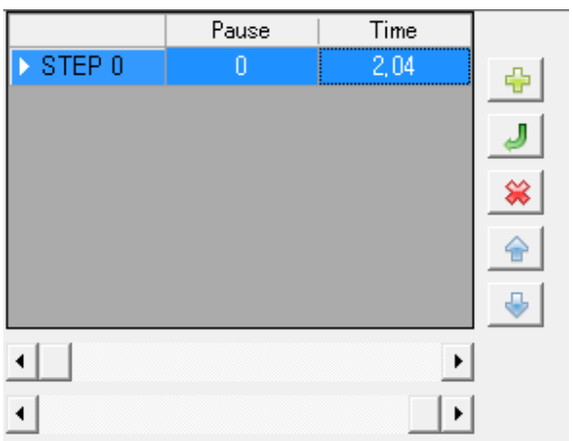


Turn the torque on/off

- After selecting the joint press the space bar to turn the torque on or off.

Move robot step by step

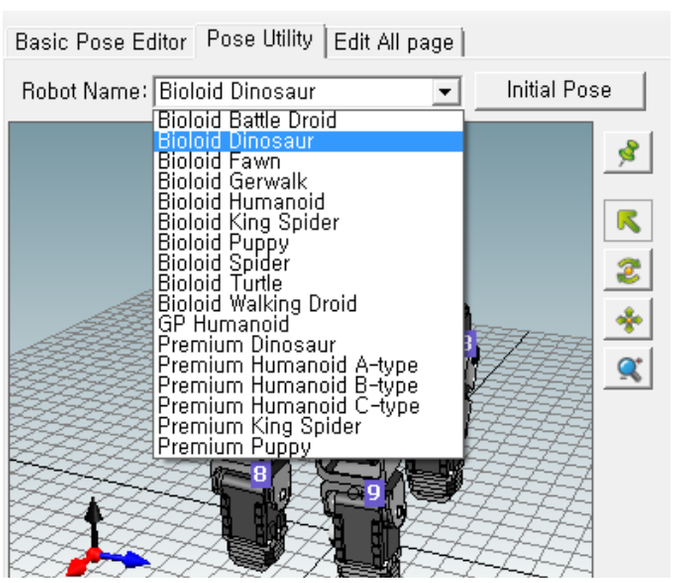
- Please choose the step that you want and press the enter key. Your robot will take the pose of selected step (This is available only at robot motion window).



Making Robots

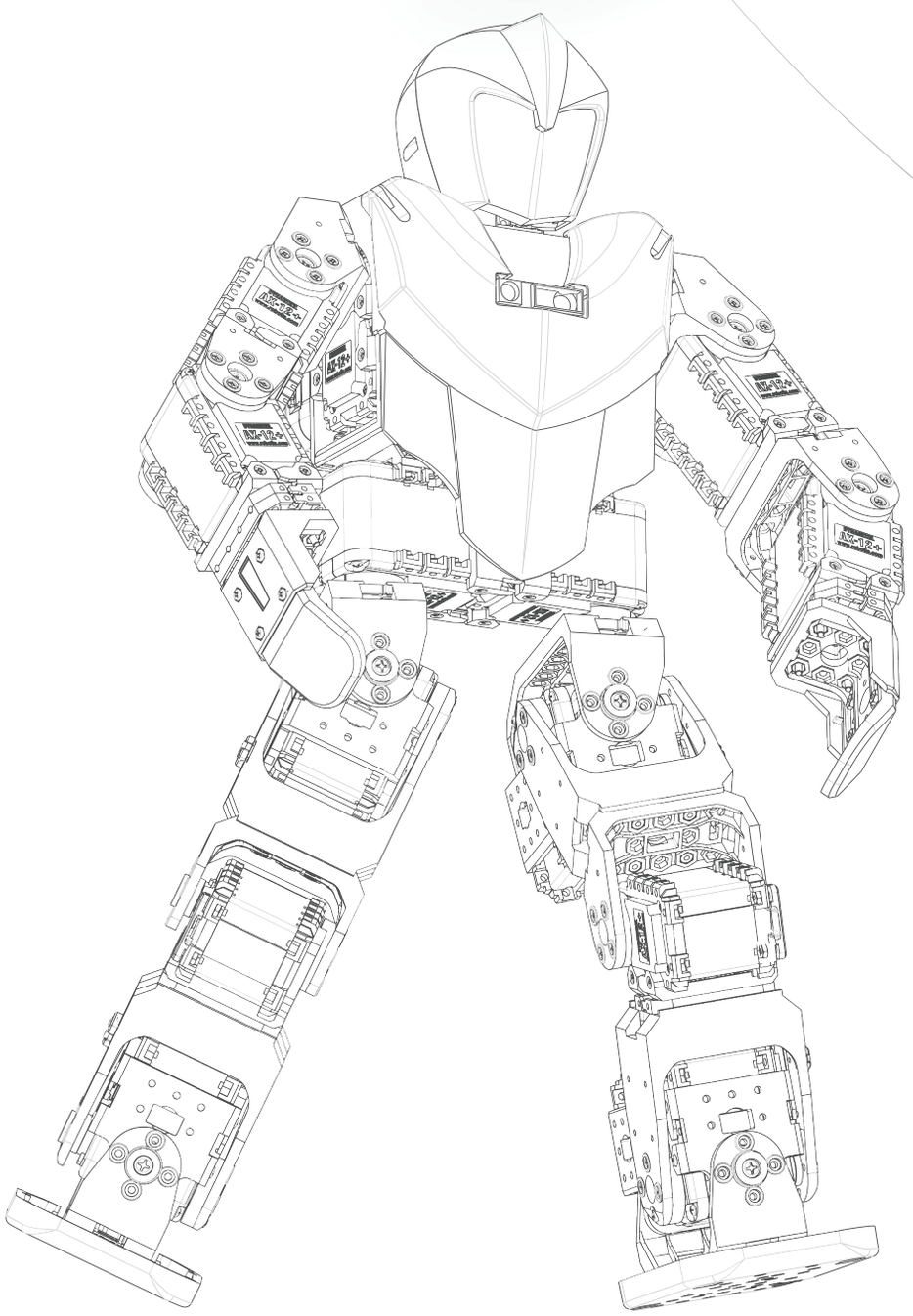
Users can make and operate their own robots using RoboPlus Motion.

- To control the robot in 3D you need 3D Model Data and its position information. You need CAD tools such as Inventor, Pro/E, etc.
- For additional information of “Mirror” and Robot know how to write the basic structure in .txt file.
- Plug-In SDK Programming Users can use Plug-In SDK to add inverse kinematics computing module from "Pose Utility" (C# Programming).



Building your own robot

This part is for the advanced users only and needs more information if you are using Pose Utility. For more information please refer to the e-Manual.



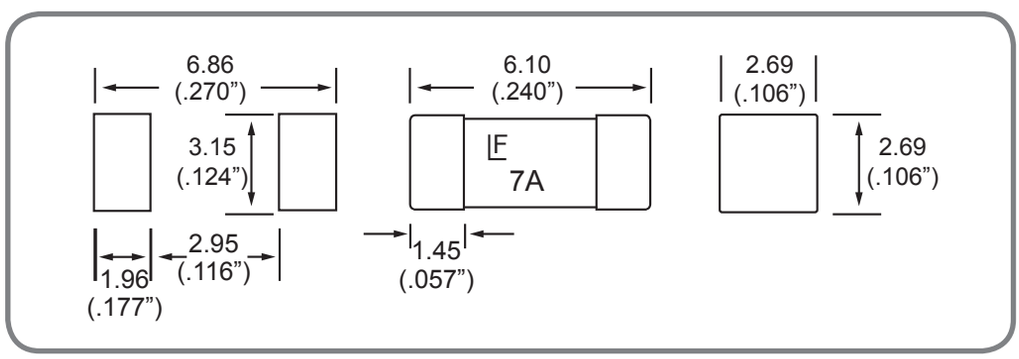
5. Useful Information

5 - 1 Major Settings and Management

5 - 1 - 1 Fuse Replacement

The use in the CM-530 prevents damage from current overload. If the CM-530 does not turn on under battery power but turns on under SMPS power replace the fuse.

The size of the fuse is shown below. Use a 125V/5A~10A fuse.



How to replace the fuse

1. Find the fuse on the back of the CM-530.

2. Use a pincette to replace the fuse with a new one.

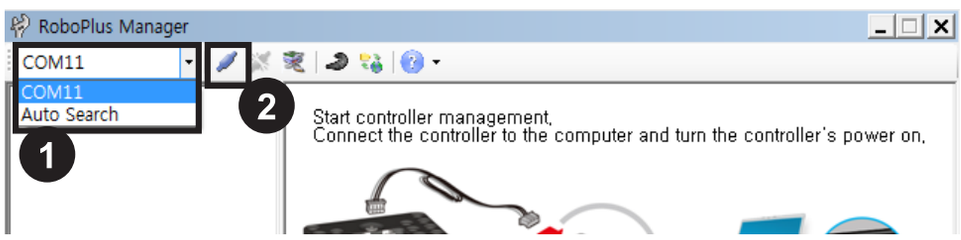
5-1-2 AX-12A Management

The AX-12A is equipped with many functional settings. This section explains how to change the dynamixel's settings.

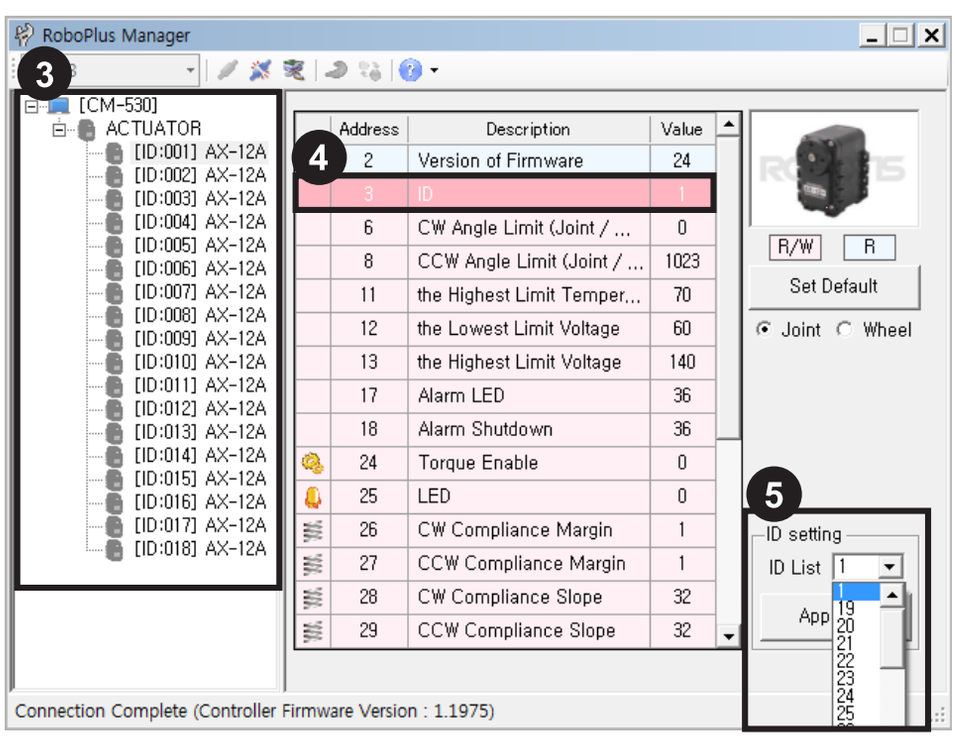
Change the ID

Here's how you can change the ID.

1. Select the port that CM-530 is connected to.
2. Click "Connect".



3. A list of connected AX-12A is shown on the left. Click on AX-12A you wish to change the ID.
4. Click on the ID row in the Control Table.
5. Click on the ID List combo box to see a list of possible ID's; Select the ID and click Apply.



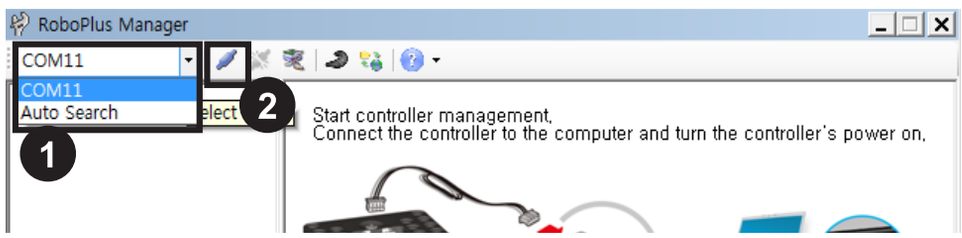
* To use in RoboPlus Motion and RoboPlus Task the ID must be between 0 and 25.

Change the Movement Mode

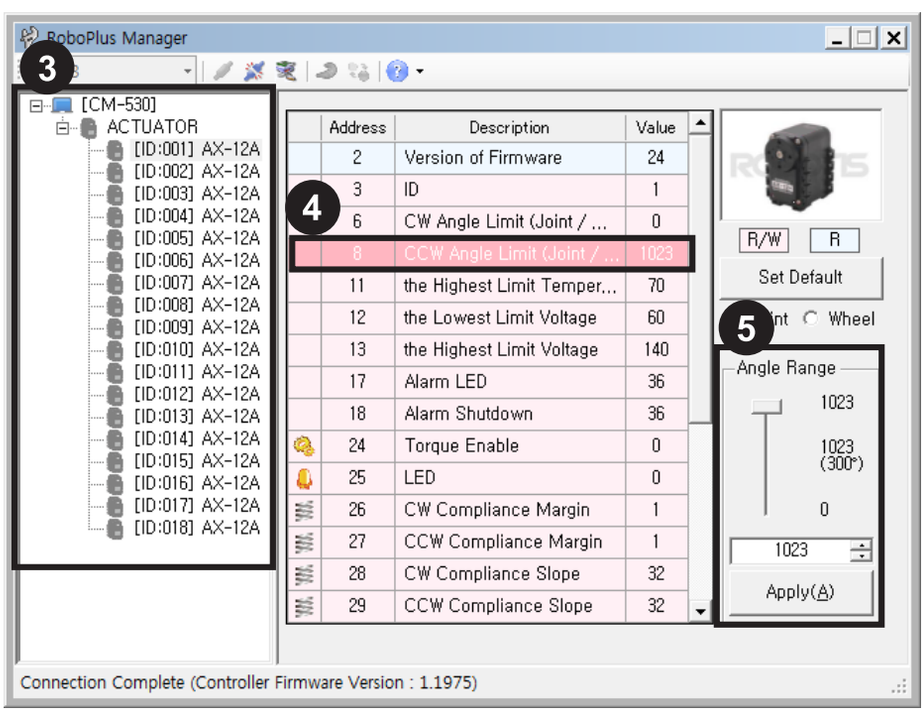
AX-12A can operate in 2 different modes.

- Wheel Mode : Rotates 360 degrees like a regular motor.
- Joint Mode : Moves at a set angle with normal servo motors.
- The mode can be changed using RoboPlus Manager. Once mode is set it will remain set (even after powering off).

1. Select the port that the CM-530 is connected to.
2. Click “Connect”.



3. A list of connected AX-12As is shown on the left. Click on the AX-12A you wish to change the mode of. Then, click on the CW/CCW Angle Limit line in the Control Table.
4. To set to Wheel Mode change the CW/CCW Angle Limit value to “0;” Or, simply click on the "Wheel Mode" button.



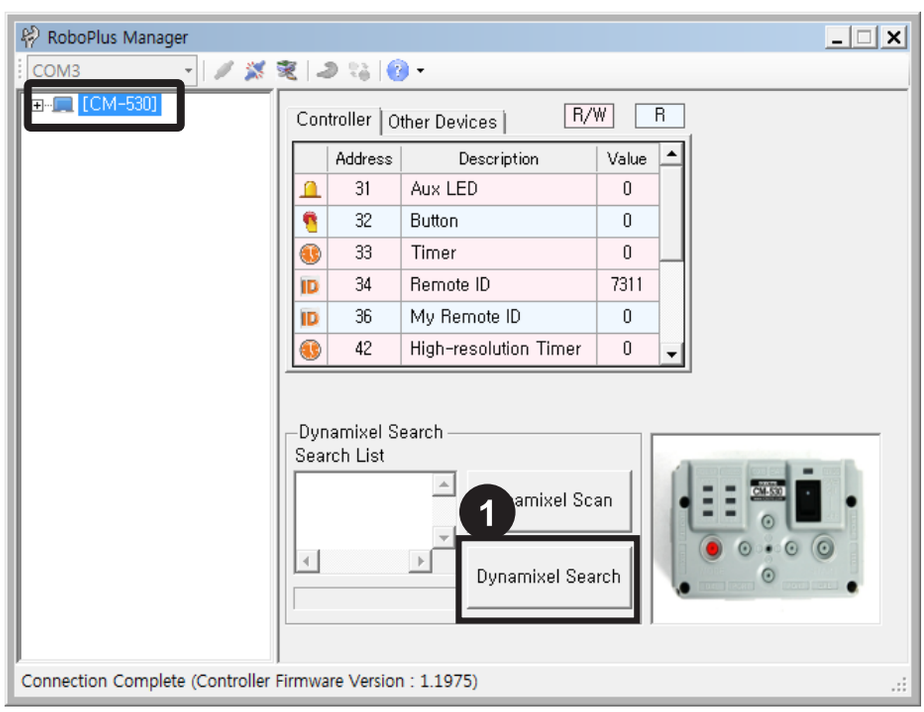
5. To set to Joint Mode again set the CW/CCW Angle Limit value to any number other than "0". The initial values for Joint Mode are "0" for CW Angle Limit, and "1023" for CCW Angle Limit.
   The initial values for Joint Mode are "0" for CW Angle Limit and "1023" for CCW Angle Limit.

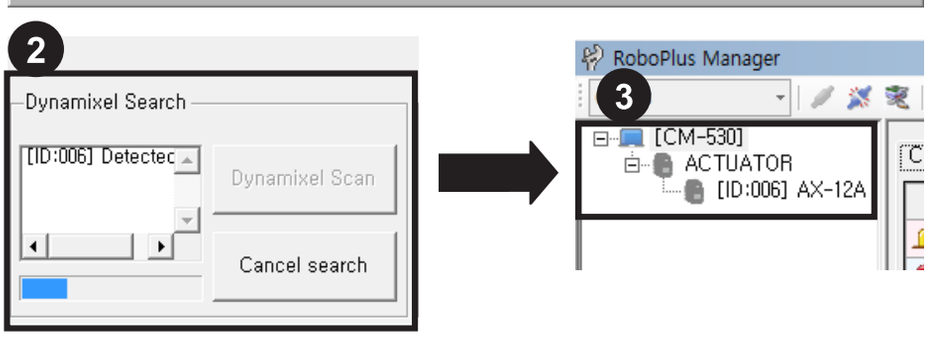
Troubleshooting

If you cannot find the AX-12A you are looking for using RoboPlus Manager, try the following :

- Connect just 1 Dynamixel and check if there are any duplicate IDs. If you see a Dynamixel on the left even though only 1 Dynamixel is connected there is a high probability of a duplicate ID. Change the ID immediately.
- If you are unable to find any AX-12A as in the image below click on "Detailed Search".

If the communication speed is not set to 1Mbps the "Detailed Search" function automatically resets the CM-530's communication speed to 1Mbps to enable it to be recognized.





If the problem persists the AX-12A may need repair. Please contact the service department of the company you purchased from.

5 - 2 Wireless Control

5 - 2 - 1 PAN Communication Wireless Control

Zigbee/ Bluetooth

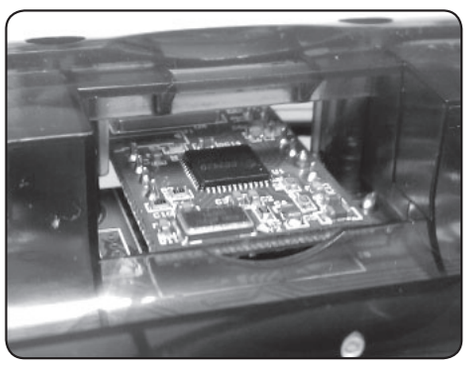
Zigbee and Bluetooth communication are one of the technologies which increased the reliability of short distance communication, which are also called as PAN (Personal Area Network) communication.

The PAN communication technology has better communication quality than the IR method, and is almost free of interruptions that may be caused when multiple devices are present. This allows for smooth robot controlling even in the presence of various wireless devices, such as in robot competitions.

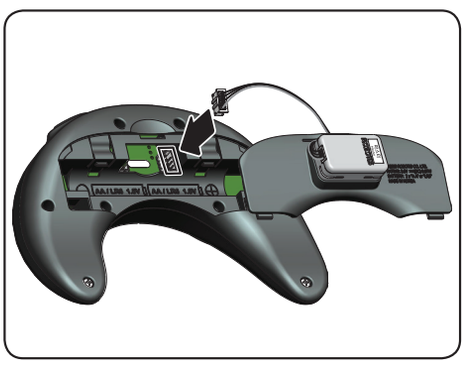
CM-530 & Zigbee/Bluetooth

The standard setting for controlling the ROBOTIS Premium with the RC-100B uses IR wireless communication.
To use a Zigbee or Bluetooth module, you must purchase the wireless modules that are sold separately and mount them on the robot and RC-100B.

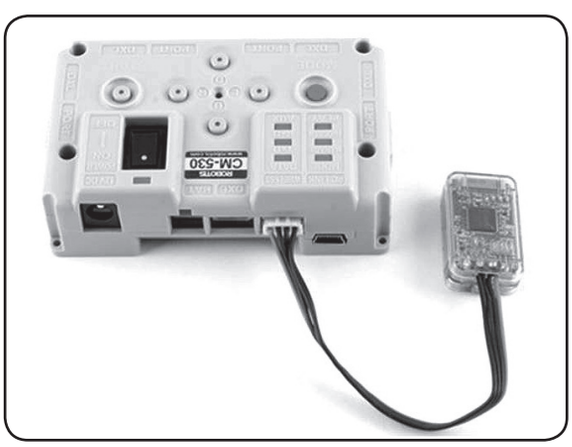
> The modules in a single Zigbee/Bluetooth set have been preconfigured to communicate with each other. Therefore, a module from one set may not work with a module from another set. Please be careful not to mix them up.



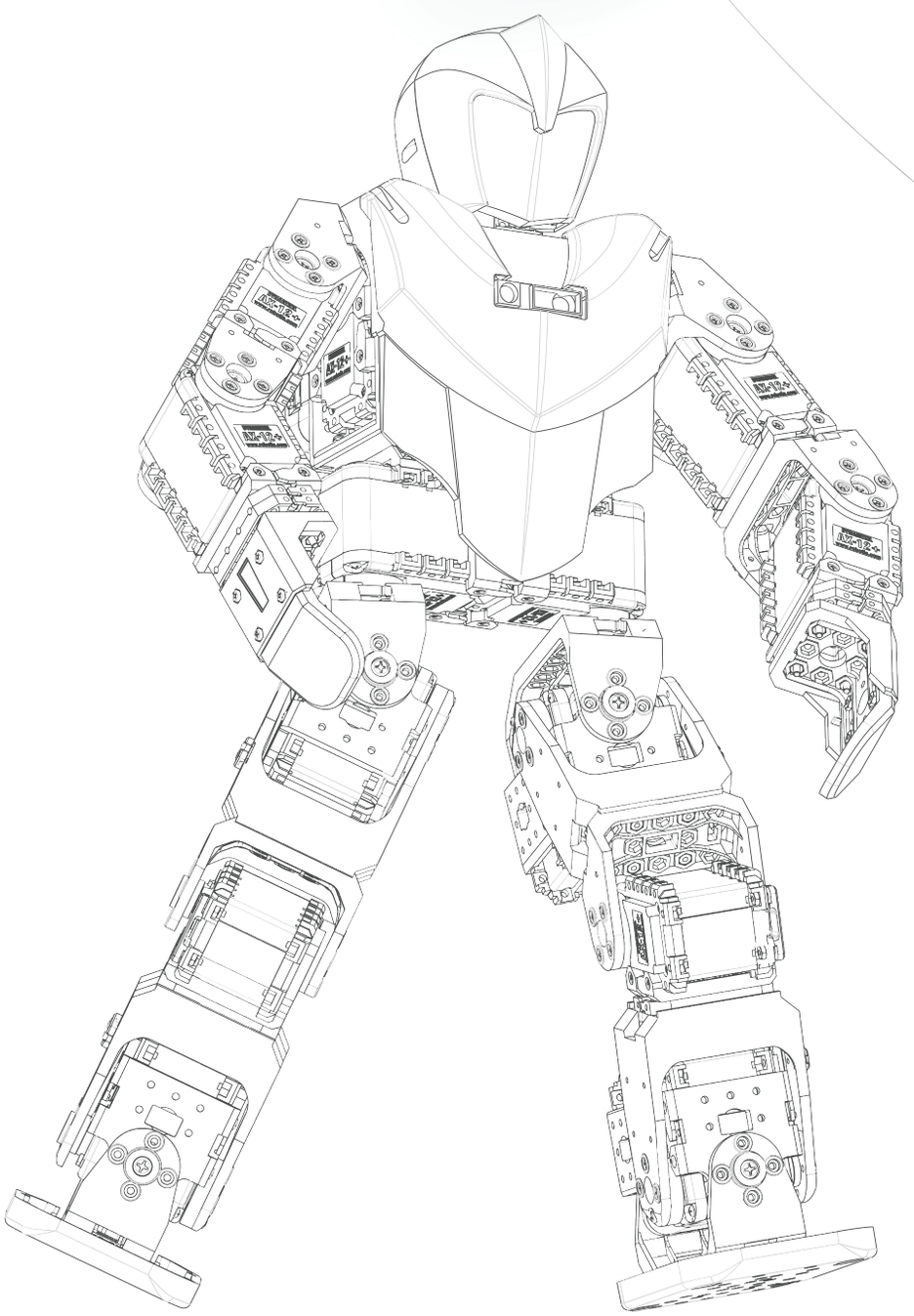
An image of the ZIG-100 installed in the RC-100B



An image of the BT-410 installed in the RC-100B



An image of the ZIG-110 (BT-410)installed in the CM-530



6. Customizing your robot

Customizing your robot

Customizing your robot

ROBOTIS PREMIUM has been designed to allow the user to create customized robots. We recommend you to build various shapes of robots other than provided 26 examples.

You can plan your own robot from now on!

Step 1	Decide on the Shape and the Functions of the Robot
Step 2	Assembly
Step 3	Programming with RoboPlus Motion/Task
Step 4	Test, Adjust, and Add

Step 1 Decide on the Shape and the Functions of the Robot

This step is to decide what kind of robot you want to build. You may use the examples as reference or begin a new design.

Step 2 Assembly

Assemble the robot around the CM-530. Assemble the frames, AX-12As, and sensors around the CM-530. You may use the examples for reference. Once your robot is assembled and wired move the joints and ensure the wires have proper slack.

Step 3 Programming with RoboPlus Motion/Task

Customize and adjust the settings of your robot with RoboPlus Manager.
Create unique movements with Motion.
Create your robot's behavior with Task.
For simpler robots Task should be enough to create movements.
For more complex robots (with more than 4 joints) Motion is recommended in conjunction with Task.

Step 4
Test, Adjust, and Add

Once finished with Motion and Task codes test the robot.

Revise and add parts/programming to make this robot completely yours.

ROBOTIS PREMIUM User's Guide

First Edition	May 13, 2013
Revision 4	February 11, 2019
Published by	ROBOTIS Co., Ltd.
Address	37, Magokjungang 5-ro 1-gil, Gangseo-gu, Seoul, 07594, Republic of Korea TEL : +82-70-8671-2609 FAX : +82-70-8230-1336
Website	www.robotis.com
Written and Edited by	ROBOTIS Planning Contents Department